МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»
Кафедра систем информационной безопасности

«ДОПУЩЕН К ЗАЩИТЕ»
Зав. кафедрой к.п.н., доцент Р. Ш. Бердибаев
_____ «_____» _____2019 г.
(подпись)

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Технологии разработки современных Web-приложений
Специальность: 5В100200 - «Системы информационной безопасности»
Выполнил Мамашев Салих Ильдарович Группа СИБ-15-2
Научный руководитель Турганбаев Ерик Сулейменович

Консультант:
по экономической части:
к. э. н., профессор Аренбаева Т. Г.
(ученая степень, звание, Ф.И.О)
(подпись) «22» мая 2019 г.

по безопасности жизнедеятельности:
д. т. н., ст. преподаватель Бекбасаров Ш. Ш.
(ученая степень, звание, Ф.И.О)
(подпись) «21» мая 2019 г.

по применению вычислительной техники:
д. ф-м. н., профессор ВАК, доцент Турганбаев Е. С.
(ученая степень, звание, Ф.И.О)
(подпись) «24» мая 2019 г.

Нормоконтролер:
ст. преподаватель, м. т. н. Аскарова Н. Б.
(ученая степень, звание, Ф.И.О)
(подпись) «28» 05. 2019 г.

Рецензент:
_____
(ученая степень, звание, Ф.И.О)
_____ «_____» _____2019 г.
(подпись)

Алматы 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»

Институт систем управления и информационных технологий

Кафедра систем информационной безопасности

Специальность «Системы информационной безопасности»

ЗАДАНИЕ
на выполнение дипломного проекта

Студенту Мамашеву Салиху Ильдаровичу

Тема проекта Технологии разработки современных Web-приложения

Утверждена приказом по университету №_124_ от «_26_» _октября_ 2018 г.

Срок сдачи законченного проекта «_28_» _____мая_____2019 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): проект подразумевает разработку веб-приложения, с использованием новейшего стека технологий и надежной системы безопасности для него. Помимо этого будет проведен детальный анализ эффективности примененных технологий и его системы защиты.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта: дипломный проект включает в себя 5 глав, разделенных на подглавы, каждая из которых освещает определенную тематику, используемую при разработке современного веб-приложения.

В первой главе дипломного проекта представлена общая информация о веб-приложении: что из себя представляет, его архитектура, из каких частей состоит, как эти части между собой взаимодействуют, что означает безопасность приложения и описание одного из современных подходов организации архитектуры приложения – паттерн MVC.

Во второй главе дипломного проекта приводится список используемых технологий с их детальным описанием: в основу приложения ляжет стек MEAN (MongoDB, Express, Angular, Node.js), для аутентификации и поддержки состояния сессиии пользователя будет использоваться JWT-токен, модуль crypto для доступа к различным методам шифрования, технология

безопасности http-сервера helmet, passport для создания стратегий авторизации пользователей и более мелкие модули и библиотеки.

В третьей главе подробно описывается процесс разработки приложения: от проектирования до построения системы защиты для уязвимых мест, а так же анализ эффективности выбранных технологий, и эффективность созданной системы защиты перед различными типами уязвимостей.

В четвертой главе приводится технико-экономическое обоснование, показывающее актуальность разработки приложения с выбранными технологиями с финансовой точки зрения.

В пятой главе рассматриваются необходимые условия для комфортной разработки программного продукта.

Перечень графического материала (с точным указанием обязательных чертежей):

1 схема архитектуры приложения с использованием MVC;

2 скрины исходного кода приложения;

3 скрины готового приложения;

4 скрины с результатами работы различных сканнеров уязвимостей;

5 скрины со статистическими данными по выбранной теме

Основная рекомендуемая литература:

1 «Node.js в действии, 6-е издание» / Д. Майк Кантелон, Марка Хартер. – 2018. – 432 с., ил

2 «MongoDB в действии, 4-е издание» / Кэйл Банкер. – 2017. – 539 с., ил.

3 «Стек MEAN, 2-е издание» / Саймон Холмс. – СПб: Символ-Плюс, 2018. – 622 с., ил.

4 «Angular для проффессионалов» / Адам Фримен. – СПб: Символ-Плюс, 2017. – 980 с., ил.

5 Курс «Angular7» / «WebForMySelf.com Team»; — 2019. URL: udemy.com/

Конструкции по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
4	Аренбаева Ж. Г.	04.03. – 22.05.19	
5	Бекбасаров Ш. Ш.	04.03 – 21.05.19	
1	Турлыбаев Е. С.	04.03 – 06.04	
2	Турлыбаев Е. С	06.04 – 21.04	
3	Турлыбаев Е. С.	21.04 – 24.05	

График
подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Изучение принципов работы Web-прил.	07.04.2019	
Изучение архитектуры Web-прил	10.04.2019	
Изучение работы паттерна MVC	16.04.2019	
Выбор стека совреем. технологий	18.04.2019	
Разработка серверной части	25.04.2019	
Разработка клиентской части	05.05.2019	
Исправление ошибок	07.05.2019	
Построение системы защиты Web-прил.	12.05.2019	
Анализ эффективности тех-ий	14.05.2019	
Анализ надежности системы защиты	16.05.2019	
Экономическая часть	20.05.2019	
Безопасность жизнедеятельности	21.05.2019	

Дата выдачи задания «04» марта 2019 г.

Заведующий кафедрой _____ (_____)
(Подпись) (Ф.И.О)

Научный руководитель проекта _____ (Тургамбаев Е.С.)
(Подпись) (Ф.И.О)

Задание принял к исполнению студента _____ (Мамашев С.И.)
(Подпись) (Ф.И.О)

АННОТАЦИЯ

Тема дипломного проекта: «Технологии разработки современных Web-приложений».

Целью данной дипломной работы является создание современного веб-приложения с использованием новейшего стека технологий, которые бы соответствовали сегодняшнему положению вещей в сфере Интернет технологий, а так же построение системы защиты для созданного приложения. С последующим анализом эффективности примененных технологий и оценки качества созданной системы защиты.

АҢДАТПА

Дипломдық жобаның тақырыбы: «Заманауи Web-қосымшаларын әзірлеу технологиялары».

Дипломдық жұмыстың мақсаты Интернет-технологиялар саласындағы ағымдағы жағдайға сәйкес келетін жаңа технологияларды пайдалану арқылы қазіргі заманауи веб-қосымшаларды құру, сондай-ақ құрылған қосымшаға арналған қауіпсіздік жүйесін құру болып табылады. Қолданбалы технологиялар тиімділігін және құрылған қорғаныс жүйесінің сапасын бағалауды кейіннен талдау.

ANNOTATION

Theme of the graduation project: «Technologies of development of modern Web-applications».

The purpose of this work is to create a modern web application with using the latest stack of technologies that would correspond to the current state of affairs in the sphere of Internet technologies, as well as building a security system for the created application. With the subsequent analysis of the effectiveness of the applied technologies and the assessment of the quality of the created protection system.

Содержание

Введение 7
1 Современное Web-приложение 9
1.1 Понятие Web-приложения 9
1.2 Архитектура Web-приложения 9
1.2.1 Клиентская часть 12
1.2.2 Серверная часть 13
1.2.3 Технологии взаимодействия клиента и сервера 14
1.3 Надежность и безопасность Web-приложений 16
1.4 Концепция MVC для создания веб-приложений 17
2 Применяемые технологии 20
2.1 Базовые технологии 20
2.2 Стек MEAN 20
2.2.1 База данных MongoDB 20
2.2.2 Серверный модуль Express 22
2.2.3 Клиентский фреймворк Angular 23
2.2.4 Платформа Node.js 27
2.3 Модуль crypto для шифрования 28
2.4 JWT-токен 28
2.5 Технология безопасности Helmet 30
2.6 Используемый редактор текста 280
3 Создание приложения с помощью выбранных технологий 302
3.1 Описание создаваемого проекта 302
3.2 Проектирование базы данных 303
3.3 Проектирования архитектуры приложения 324
3.4 Создание серверной части с помощью на Node 335
3.5 Добавление клиентской части с помощью Angular 424
3.6 Построение защиты для созданного приложения 535

3.7 Преимущества выбранного стека технологий 581
3.8 Анализ безопасности веб-приложения 664
4 Безопасность жизнедеятельности.....736
4.1 Исходные данные 736
4.2 Рассчитать тепловые нагрузки в помещении: внутренние и наружные.736
4.3 Рассчитать количество воздуха, необходимое для подачи в помещение 770
4.4 Привести основные характеристики выбранного кондиционера 781
4.5 Привести схему расположения кондиционера в помещении.....781
5 Технико-экономическое обоснование.....814
5.1 Определение трудоемкости разработки ПО.....814
5.2 Расчет затрат на разработку ПО 825
5.3 Расчет затрат на электроэнергию 836
5.4 Расчет затрат на оплату труда.....8487
5.5 Расчет затрат по социальному налогу.....88
5.6 Амортизация основных фондов и прочие затраты 89
5.7 Определение возможной (договорной) цены ПО 90
Заключение 892
Список литературы 914
Приложение А 936
Приложение Б.....99
Приложение В.....1203

Введение

По данным Международного союза электросвязи (МСЭ), количество пользователей Интернета за последний год впервые в истории достигло половины населения планеты, а количество активных сайтов равно около 1,8 млрд. На основании этого можно говорить о том, что на сегодняшний день наличие собственного сайта считается критерием современного предприятия, фирмы или даже магазина. Параллельно, вместе с этим возросли требования к создаваемым сайтам – заказчики требуют удобное, быстрое и безопасное решение для своего вопроса. Для разработчиков это усложняется тем, что технологии, включая языки программирования, фреймворки, CMS и т.д., с каждым годом изменяются и развиваются. Для тех, кто сегодня разрабатывает веб-приложения важно выбрать не просто хорошую технологию, а предугадать тренды развития так, чтобы созданное приложение было эффективным и надежным еще несколько лет вперед.

Несмотря на появление новых, интересных трендов около 70-80% сайтов используют старые серверные технологии на базе LAMP(Linux, Apache, MySQL, PHP). Например, в сети есть много хороших и стабильно работающих сайтов, которые разработаны с применением базового комплекса технологий (HTML, CSS, JavaScript) и LAMP. Однако, если бы при их разработке применялся бы более новый подход, использовались бы более прогрессивные и функциональные технологии, а также уделялось особое внимание надежности и безопасности – можно было бы увеличить не только удобство и скорость разработки, но и добиться большей безопасности, более качественного взаимодействия компонентов системы, увеличить скорость работы и отклика сервера. Именно этому направлению будет посвящен данный дипломный проект.

Целью данной дипломной работы является создания современного веб-приложения с использованием новейшего стека технологий, которые бы соответствовали сегоднешнему положению вещей в сфере Интернет технологий с учетом особенностей создаваемого сайта. С последующим анализом их эффективности.

Исходя из предметной области, выбранной темы, а так же цели дипломной работы, можно выделить следующие задачи проекта:

- дать описание современному веб-приложению;
- выбрать и охарактеризовать набор выбранных технологий;
- использовать выбранные технологии для создания приложения;
- провести анализ эффективности разработанного веб-приложения;
- провести анализ безопасности разработанного веб-приложения.

При этом – каждой задаче посвящена отдельная глава данной дипломной работы (в основной части будет 4 главы).

В основу использованного набора технологий пойдет стек MEAN, в который входит – платформа Node.js, на которой будет располагаться наше приложение, Angular для организации клиентской части, Express для создания

серверной части, MongoDB, как хранилище данных, а так же более детальные технологии для создания сервисов аутентификации, загрузки файлов и т.д. Помимо всего этого будет уделено внимание аспектам обеспечения безопасности – будут применены средства защиты к особо «чувствительным» частям системы.

Данная тема является актуальной, поскольку веб-технологии развиваются с большой скоростью. И очень важно, если вы начинающий или уже опытный разработчик, чтобы использовались самые современные и прогрессивные технологии web-разработки, которые бы обеспечивали наибольшее удобство, быстроту, функциональность и безопасность на несколько лет вперед.

Результатом дипломной работы станет современное многопользовательское веб-приложение, которое представляет собой несколько веб-станиц, с функциями аутентификации, загрузки файлов и работой с элементами базы данных с использованием функций просмотра и добавления данных. Помимо этого будет произведен подробный анализ безопасности данного приложения и эффективности примененных технологий разработки.

1 Современное Web-приложение

В нашей работе мы стремимся создать современное веб-приложение (Modern Web App) – приложение, придерживающееся всех современных веб-стандартов. Среди них Web App Security – безопасность приложения и надежность хранения пользовательских данных. Так же это максимально удобное взаимодействие с пользователем, идеальная поисковая оптимизация; и естественно — высокая скорость загрузки и отклика. Так же учтем условие скорости и удобство процесса разработки.

Исходя из этого определения, а также поставленных задач необходимо создать веб-приложение, отвечающий следующим требованиям:

- удобство и скорость при разработке;
- безопасность приложения;
- функциональный интерфейс;
- удобство загрузки и взаимодействия с сайтом;
- высокая скорость работы приложения.

Для достижение этих целей нужно выбрать подходящий современный стек технологий, но перед этим необходимо описать что представляет из себя веб-приложение.

1.1 Понятие Web-приложения

Веб-приложения представляет собой приложение, обычно с клиент-серверной архитектурой, где основная часть находится на удаленном сервере, а клиентский интерфейс находится в окне браузера. Основная часть называется сервером приложения, а то, что показывается в браузере – клиентом.

Для запуска таких приложений нет необхдимости применять сторонние программы. Все, что необходимо – браузер и доступ к сети Интернет. Так же может использоваться разные типу устройств – от обычного смартфона до персонального компьютера.

В отличие от обычных программ, веб-приложению не нужна определенная операционная система, поэтому нет необходимости подстраивать разработанную систему для Linux, Windows, Mac OS и других операционных систем.

Среди преимуществ веб-приложения можно выделить: кроссплатформенность, экономия, адаптивность, отсутствие стороннего программного обеспечения и масштабируемость.

1.2 Архитектура Web-приложения

Как уже говорилось ранее веб-приложение состоит из клиентской и серверной частей, тем самым реализуя технологию «клиент-сервер».

Основные функции клиентской части – обеспечить графический интерфейс взаимодействия с пользователем, отправлять запросы к серверу, а так же получать ответы от него.

Основные функции серверной части – получать запросы от клиента, далее выполнять определенные вычисления, в дальнейшем формировать ответ и обратно отправлять клиенту. Все это осуществляется с помощью протокола HTTP.

Само веб-приложение может выступать в качестве клиента других служб, например, базы данных или другого веб-приложения, расположенного на другом сервере. В этом случае сервер должен установить специальные модули взаимодействия и превратится в «middleware» для клиентской части и другими сервисами.

В таких приложениях сервер и клиент работают в режиме «запрос-ответ». Поэтому организация такого взаимодействия требует специальных модулей, которые бы обеспечивали функцию посредника между браузером и сервером.

На сегодняшний день для клиент-серверного взаимодействия актуально применять следущие средства связи – технологию Ajax или Web-socket. Эти элементы веб-приложение будут рассмотрены в главе 1.2.3.

Так же следует отдельно упомянуть о сервере БД, поскольку это, пожалуй, самый часто используемый прикладной сервис в веб-приложениях расположенных в сети Интернет.

База данных, или зачастую ее называют СУБД – программное обеспечение, основная функция которого – создать структуру для хранения и, в случае необходимости отображения данные. В случае форума или блога, хранимые в БД данные – это посты, комментарии, новости, и так далее. База данных располагается на серверной стороне. Серверная часть веб-приложения обращается к базе данных, извлекая данные, которые необходимы для формирования страницы, запрошенной пользователем.

Таким образом, веб-приложение, которое работает по принципу клиент-сервер, с использованием БД можно представить в следующем виде (Рисунок 1.2.1).

Далее будет более подробно описаны компоненты веб-приложение – клиентская и серверная часть.

1.2.1 Клиентская часть

Исходя из данных главы 1.2, можно сделать вывод, что клиентская часть веб приложения – это графический интерфейс или то, что вы видите на странице. Пользователь взаимодействует с веб-приложением через браузер, нажимая на ссылки и кнопки.

Для описания графического интерфейса страниц используется основной язык разметки HTML. Этот язык создает структуру веб-страницы. Для художественного оформления полученной разметки используют специальную таблицу стилей – CSS.

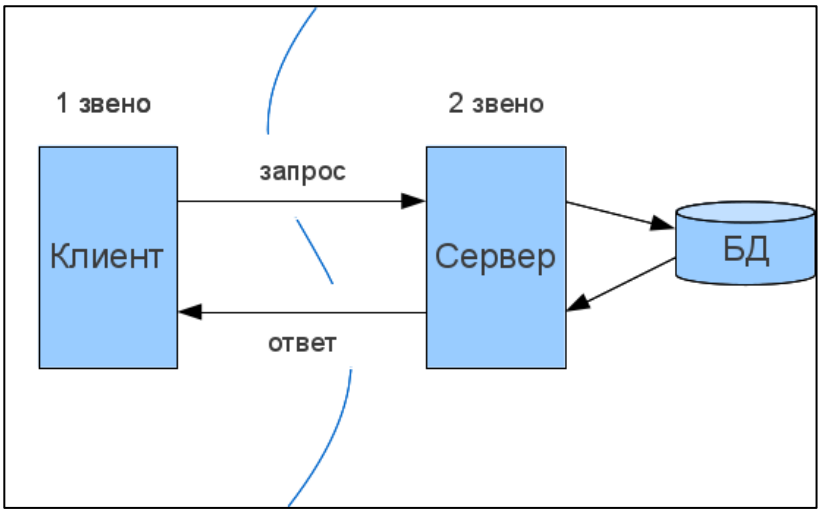

Рисунок 1.2.1 – Архитектура клиент-серверного приложения

Для создания взаимодействия с графическим интерфейсом веб-приложения используется специальный скриптовый язык – JavaScript. Написанный таким образом скрипт встраивается в веб-страницу, затем он может управлять встроенными в страницу компонентами, тем самым реализуя пользовательский интерфейс с богатыми возможностями.

Однако применение только «чистого» JavaScripta немного устарело, поэтому сейчас применяются специальные фрэймворки. Фреймворк это платформа, которая предоставляет программистам основу для разработки приложений. Он содержит заранее определенные и реализованные классы и функции, которые упрощают и ускоряют работу программисту. Фреймворки весьма эффективны при создании сайтов, богатыми функциональностью. В этом случае это намного проще реализовать на Angular, например, чем на старом jQuery. Это обеспечивается за счет применения определенной структуры страницы. Такой код получается меньшего объема, а поэтому его легче исправлять и масштабировать. Еще одно преимущество заключается в том что мы используем обычный JavaScript, но значительно расширяем его при использовании фреймворка. Это дает возможность быстро создать мобильное или настольное кроссплатформенное приложение из веб-версии, используя такие системы, как PhoneGap. Самые известные веб-фреймворки включают React JS, Angular и Vue JS.

1.2.2 Серверная часть

Серверная часть веб-приложения - это программа или скрипт на сервере, обрабатывающая запросы пользователя (точнее, запросы браузера). Чаще всего серверная часть веб-приложения создается, на каком либо языке программировании. При каждом переходе пользователя по ссылке браузер отправляет запрос к серверу. Сервер обрабатывает этот запрос, вызывая некоторый скрипт, который формирует веб-страничку или запрос и отсылает клиенту по сети. Браузер тут же отображает полученный результат.

Для разработки веб-приложений можно использовать практически любые современные языки программирования:

- PHP, Perl;
- Ruby;
- Java;
- платформа .NET;
- Python;
- JS.

Независимо от языка, на котором написана серверная часть веб-приложения, способы обработки запросов и взаимодействия с пользователем остаются те же.

В целом можно выделить следущие функции, которые берет на себя сервер:

- эффективное хранение и доставка информации-рограммирование серверной части позволяет вместо этого хранить информацию в базе данных и динамически создавать, и возвращать HTML и другие типы файлов (например, PDF, изображения, и т.д.);
- контролируемый доступ к контенту-Программирование серверной части позволяет сайтам ограничивать доступ авторизованным пользователям и предоставлять только ту информацию, которую пользователю разрешено видеть;
- хранение информации о сессии/состоянии-Программирование серверной части позволяет разработчикам использовать сессии – изначально это механизм, позволяющий серверу хранить информацию о текущем пользователе сайта и отправлять разные ответы, основанные на этой информации. Это позволяет, например, сайту знать, что пользователь был предварительно авторизован и отображает ссылки на их адрес электронной почты или историю заказов, или возможно сохранить прогресс простой игры, так чтобы пользователь мог при следующем заходе на сайт продолжить оттуда, где он закончил;
- уведомления и средства связи-Серверы могут отправлять общие или пользовательские уведомления непосредственно через сайт или по электронной почте, смс, мгновенные сообщения, видеосвязь или другие средства связи.

1.2.3 Технологии взаимодействия клиента и сервера

Как уже упоминалось в главе 1.2 на данный момент рекомендуется применять две технологии взаимодействия клиента и сервера:

- AJAX (аббревиатура от Asynchronous JavaScript and XML). Это технология взаимодействия с сервером без перезагрузки страницы. Поскольку не требуется каждый раз обновлять страницу целиком, повышается скорость работы с сайтом и удобство его использования [1]. Понять основной принцип работы AJAX помогает рисунок 1.2.2. Для обмена данными на странице создается объект XMLHttpRequest, он будет выполнять функцию посредника между браузером и сервером. Запросы могут отправляться в одном двух типов – GET и POST. В первом случае обращение производится к документу на

сервере, в роли аргумента ему передается URL сайта. Для предотвращения прерывания запроса можно воспользоваться функцией JavaScript Escape. Для больших объемов данных применяется функция POST. AJAX применяет асинхронную передачу данных. Такой подход позволяет пользователю совершать различные действия во время «фонового» обмена информации с сервером. Действует оповещение пользователя о протекающих процессах, чтобы он не подумал, что сайт «завис» либо на нем произошел какой-то сбой. В качестве ответа сервер использует простой текст, XML и JSON. В первом случае результат можно сразу же отобразить на странице. При получении XML-документа его обычно конвертируют в HTML и выводят на экран. Если ответ получен в формате JSON, клиенту следует выполнить полученный код. После этого будет сформирован объект JavaScript;

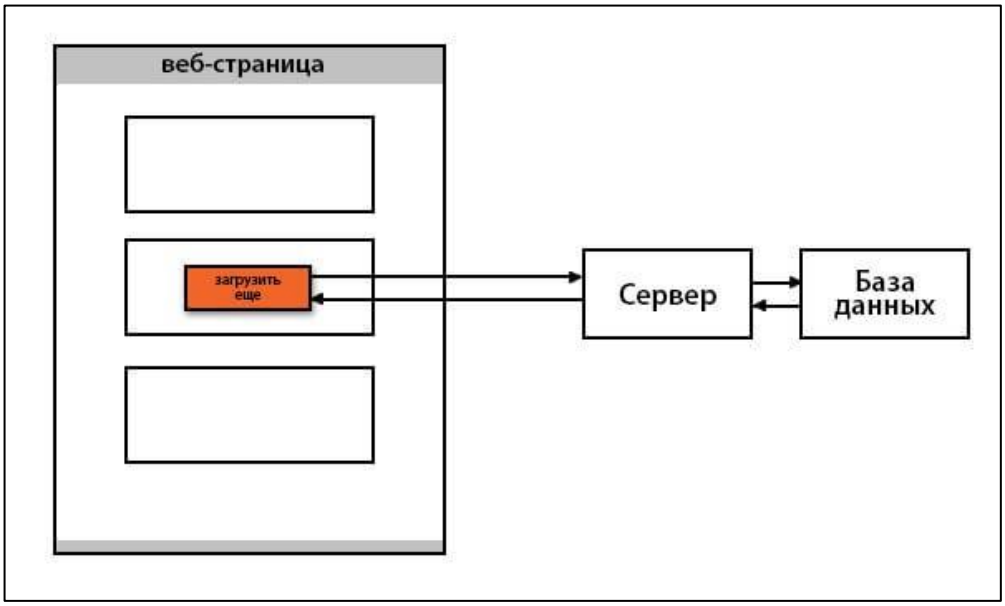

Рисунок 1.2.2 – Работа технологии AJAX

- веб-сокеты (Web Sockets) – это передовая технология, которая позволяет создавать интерактивное соединение между клиентом (браузером) и сервером для обмена сообщениями в режиме реального времени. Веб-сокеты, в отличие от HTTP, позволяют работать с двунаправленным потоком данных, что делает эту технологию совершенно уникальной. Давайте разберемся, как работает эта технология и чем она отличается от HTTP (рисунок 1.2.3). Браузер постоянно спрашивает у сервера, есть ли для него новые сообщения, и получает их. Веб-сокетам же для ответа не нужны повторяющиеся запросы. Достаточно выполнить один запрос и ждать отклика. Предоставляется возможность просто слушать сервер, который будет отправлять вам сообщения по мере готовности.

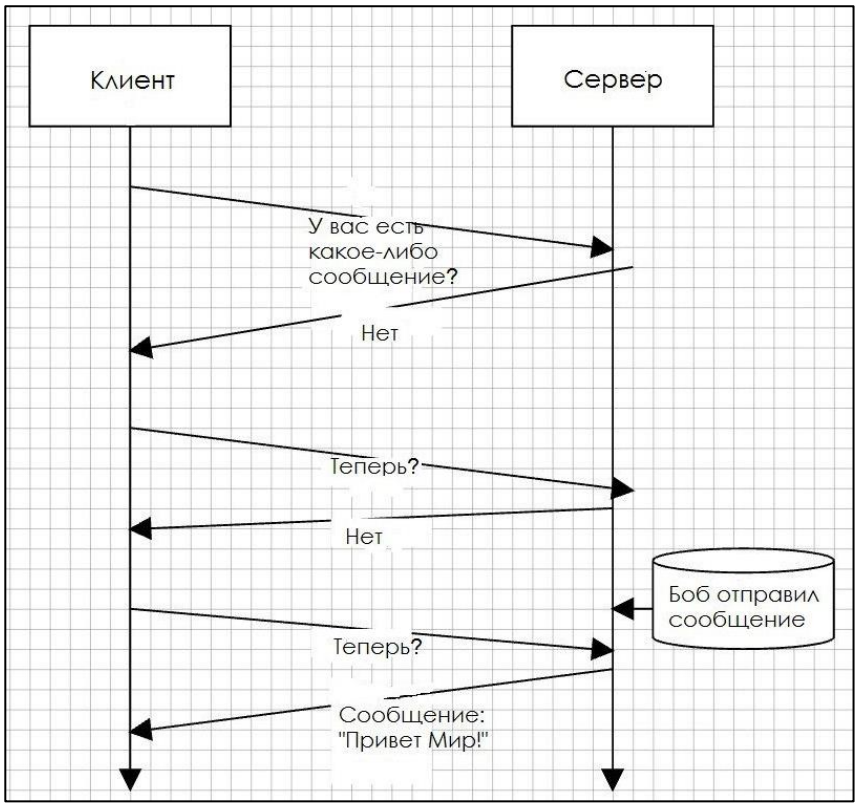

Рисунок 1.2.3 – Работа технологии Socket

1.3 Надежность и безопасность Web-приложений

Безопасность веб-приложений – один из наиболее острых вопросов в контексте информационной безопасности. Как правило большинство веб-сайтов, доступных в Интернете, имеют различного рода уязвимости и постоянно подвергаются атакам. В этой главе будут рассмотрены основные угрозы информационной безопасности веб-приложений.

Насколько безопасным должно быть приложение? Для кого-то этот вопрос не имеет смысла. "Настолько, насколько это возможно. Чем безопасней, тем лучше". Но это не исчерпывающий ответ. И он не помогает сформировать security политику в проекте. Более того, если придерживаться только этой директивы ("чем больше безопасности, тем лучше"), мы можем создать большое количество ненужных проблем [2].

Для начала нужно разобраться с тем, что может послужить угрозами безопасности для веб-приложения. Основные типы угроз информационной безопасности веб-приложения:

- угрозы конфиденциальности – несанкционированный доступ;
- угрозы доступности – ограничение или блокирование доступа;
- угрозы целостности – искажение данных.

Основным источником угроз информационной безопасности веб-приложения являются внешние нарушители. Внешний нарушитель – лицо, мотивированное, как правило, коммерческим интересом, имеющее возможность доступа к сайту компании, не обладающий знаниями об

исследуемой информационной системе, имеющий высокую квалификацию в вопросах обеспечения сетевой безопасности и большой опыт в реализации сетевых атак на различные типы информационных систем.

Говоря простыми словами – основная угроза безопасности сайта – хакерская атака. Она может иметь конечную цель, быть т.н. целевой атакой, либо атака носит бессистемный характер, по принципу – атакую все подряд, что-нибудь да сломается. Виды атак бывают целевыми или нецелевыми.

Целевые атаки – это атаки, специально нацеленные на один сайт или их группу, объединенную одним признаком (сайты одной компании, либо сайты, относящиеся к определённой сфере деятельности, либо объединенные рядом признаков). Опасность таких атак заключается именно в «заказном» характере. Исполнителями таких атак становятся, как правило, злоумышленники, обладающие высокой квалификацией в области безопасности веб-приложений.

Нецелевые атаки – это атаки, которые проводится фактически “на удачу”, а ее жертвами становятся случайные веб-сайты независимо от популярности, размера бизнеса, географии или отрасли. Нецелевая атака на сайт – это попытка получения несанкционированного доступа к веб-ресурсу, при которой злоумышленник не ставит целью взломать конкретный сайт, а атакует сразу сотни или тысячи ресурсов, отобранных по какому-то критерию. Например, сайты, работающие на определенной версии системы управления сайтом. Такого рода атаки бьют по «площадям», стараясь охватить максимальное количество сайтов при минимуме затрат.

К сожалению, не существует какого-то общего решения безопасности для всех сайтов, поскольку они имеют разные методы построения, реализации и т.д. Конечно, существуют базовые меры безопасности при разработке и поддержке работы сайта: обновлять CMS и ее компоненты; регулярно менять пароли; отказаться от использования устаревших протоколов; настроить и использовать HTTPS/HSTS. Но, тем не менее, систему защиты нужно строить для каждого веб-приложения индивидуально учитывая все аспекты и свойства данной системы.

1.4 Концепция MVC для создания веб-приложений

Создание веб-сайта – очень кропотливый и трудоемкий процесс, поэтому для упрощения и ускорения данного процесса необходимо использовать определенный поэтапный подход (методологию).

В данной дипломной работе использовалась модель - «MVC»(Рисунок 1.4.1), как наиболее понятная и удобная в рамках небольшого проекта.

MVC – это не шаблон проекта, это концепция, которая описывает способ организации структуры нашего приложения и взаимодействие каждой из частей данной системы [3].

Идея, которая лежит в основе конструкционного шаблона MVC, очень проста: нужно чётко разделять ответственность за различное функционирование в наших приложениях.

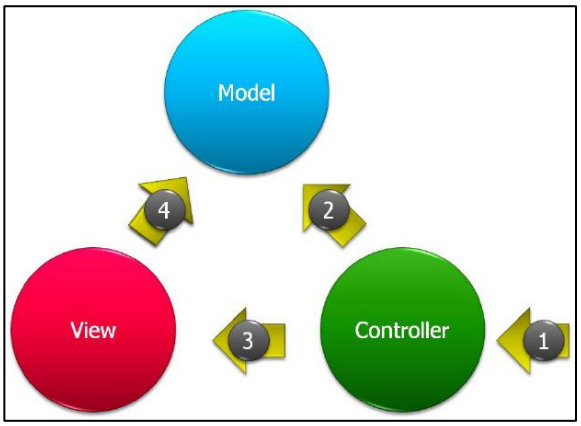

Рисунок 1.4.1 – Работа паттерна MVC

Приложение разделяется на три основных компонента, каждый из которых отвечает за различные задачи:

Контроллер – управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция – вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

Модель – это данные и правила, которые используются для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определённым образом. Что такое пользователь для приложения – сообщение или книга? Только данные, которые должны быть обработаны в соответствии с правилами (дата не может указывать в будущее, e-mail должен быть в определённом формате, имя не может быть длиннее X символов, и так далее).

Вид – обеспечивает различные способы представления данных, которые получены из модели. Он может быть шаблоном, который заполняется данными. Может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации. Веб приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

Самое очевидное преимущество, которое мы получаем от использования концепции MVC – это чёткое разделение логики представления (интерфейса пользователя) и логики приложения.

Поддержка различных типов пользователей, которые используют различные типы устройств является общей проблемой наших дней. Предоставляемый интерфейс должен различаться, если запрос приходит с персонального компьютера или с мобильного телефона.

Модель возвращает одинаковые данные, единственное различие заключается в том, что контроллер выбирает различные виды для вывода данных.

2 Применяемые технологии

Изучив тенденции и требования, которые сложились на сегоднешний день [4], а так же имеющийся опыт – были выбраны следущие технологии:

- основа - JS, HTML и CSS (3 базовые технологии без которых не обойтись при создании сайта);
- стек технологий – MEAN (это относительно новый стек технологий для разработки клиент-серверного приложения), в этот пакет входит: Сервер БД – MongoDB, Express и Node.js для серверной стороны, Angular 7 как клиентский JS фреймворк;
- технологии JWT – для аутентификации и авторизации пользователя в сети;
- модуль multer – для загрузки файлов на сервер;
- технология helmlet для обеспечения безопасности сервера;
- модуль crypto для применения различных инструментов криптографии;
- библиотеки passport и mongoose, как промежуточное ПО;
- более мелкие библиотеки и модули для реализации незначительных частей создаваемой системы;
- для создания архитектуры приложения будет использован паттерн MVC.

2.1 Базовые технологии

К базовым технологиям веб-разработки относятся – язык разметки страниц HTML, язык стилей CSS, язык функционирования и взаимодействия JavaScript.

HTML (язык разметки гипертекста) – это специальный язык для разметки веб-страниц, используемый для создания сайтов в сети Интернет. Хоть развитие HTML началось еще в 90-х годах прошлого века, в настоящее время трудно представить Интернет без него [5].

В 2014 году работа над новым стандартом была официально завершена - HTML5, который фактически произвел революцию, привнес много нового в HTML.

Что именно принес HTML5:

- HTML5 определяет новый алгоритм синтаксического анализа для создания структуры DOM;
- добавление новых элементов и тегов, таких как видео, аудио и несколько других элементов;
- переопределение правил и семантики уже существующих элементов HTML.

Фактически, с добавлением новых функций, HTML5 стал не только новой версией языка разметки для создания веб-страниц, но и фактически платформой для создания приложений, и его использование вышло далеко за пределы веб-среды Интернета: HTML5 также используется для создания мобильных приложений для Android, iOS, Windows Mobile и даже для

создания настольных приложений для обычных компьютеров (в частности, в ОС Windows 8 / 8.1 / 10).

CSS (от англ. «Cascading Style Sheets», каскадные таблицы стилей) – список стилей для сайта в формате html [6].

Что такое стиль? Грубо говоря, стиль - это то, как выглядит элемент сайта. Например, любой текст может быть написан размером 10 пикселей, а возможно 14 пикселей. Вы можете написать черным, вы можете синим. Вы можете подчеркнуть / наклонить / вычеркнуть и т. Д. Все, что связано с форматированием текста, выполняется с помощью CSS.

Но это только малая часть возможностей. За всю визуализацию всех тэгов html отвечает CSS. Даже для динамических изменений: выпадающие меню, выделение ссылок при наведении.

Список всех стилей на языке веб-мастеров часто называют «таблицей стилей CSS».

Каковы преимущества использования CSS:

- легко изменить дизайн. Достаточно изменить стиль только в одном месте, и он будет меняться на каждой странице сайта;
- это единственный способ изменить дизайн сайта;
- простой синтаксис языка.

В последнее время часто можно найти концепцию CSS3. По сути, это все тот же CSS, но с набором новых аргументов, которые предоставляют дополнительные возможности с точки зрения различных эффектов. Например, свечение текста.

JavaScript – это язык, который позволяет элементы функциональности на веб-странице [7]. Каждый раз, когда на веб-странице происходит какие-либо динамические изменения, переходы, взаимодействие с графикой и т.д. – можете быть уверены, что скорее всего, не обошлось без JavaScript. Это третий слой многослойного набора стандартных веб-технологий, два из которых (HTML и CSS) были подробно описаны в предыдущей главе.

Ядро языка JavaScript состоит из ряда общих функций, которые позволяют выполнять следующие действия [8]:

- хранить данные внутри переменных;
- операции с фрагментами текстов (известные в программировании как «строки»);
- запуск кода в соответствии с определенными событиями, происходящими на веб-странице.

Код JavaScript выполняется в браузере после обработки кода HTML и CSS и его преобразования в веб-страницу. Это гарантирует, что структура и стиль страницы уже сформированы к моменту запуска JavaScript. Такой порядок выполнения применяется, потому что часто JavaScript используется для динамического изменения HTML и CSS для обновления пользовательского интерфейса. Если JavaScript будет запущен до загрузки HTML и CSS, это приведет к ошибкам [9].

Как уже упоминалось, существует серверный и клиентский код, особенно в контексте веб-разработки. В этой части дипломной работе мы говорим о клиентском использовании JavaScript. Однако JS также можно использовать в качестве языка сервера, например, в популярной среде Node.js. Мы вернемся к этой технологии позже, когда будем говорить о стеке MEAN.

2.2 Стек MEAN

Название данного стека является аббревиатурой, где каждая буква отвечает за определенную технологию стека – три JavaScript-фреймворка и документно-ориентированная база данных NoSQL (рисунок 2.2.1).

M – MongoDB. MongoDB - это бесплатная документно-ориентированная СУБД (база данных), построенная таким образом, чтобы обеспечить высокую масштабируемость, так и гибкость в разработке. При записи данных в базу не используются стандартные таблицы и связи между ними, как в реляционной базе данных. Вместо этого MongoDB хранит JSON-подобные документы с динамическими схемами.

E – это ExpressJS. Express.js - это инфраструктура сервера приложений Node.js для создания одностраничных, многостраничных и гибридных веб-приложений. Фактически это базовая серверная структура для node.js.

A – это Angular. Angular - это фреймворк для создания быстрых и динамических веб-приложений. Он может использовать HTML в качестве языка шаблонов, а также расширить синтаксис HTML для четкого и краткого описания компонентов приложения. Привязка данных и внедрение зависимостей в Angular позволяет избавиться от большого количества кода, используя специальные классы и регулярные выражения.

N – является NodeJS. Node.js – это бесплатная кроссплатформенная среда выполнения для разработки приложений веб-сервера. Приложения Node.js написаны на JavaScript и могут выполняться практически любых операционных систем. это сокращение от MongoDB, ExpressJS, AngularJS и Node.js.

На уровне клиента, сервера и базы данных весь стек MEAN написан на JavaScript. Angular используется для разработки клиентской части. Представления будут разработаны с использованием Angular 7, чтобы они все отображались на одной странице. На стороне сервера я буду использовать Node.js, то есть инфраструктуру Express.js. Используя Express, API будет написан для связи с базой данных. Наконец, MongoDB для хранения данных [10]. Все это показано на диаграмме.Для того чтобы понять суть работу давайте детально рассмотрим каждую технологию из данного стека.

2.2.1 База данных MongoDB

MongoDB – это система управления базами данных, «заточенная» под веб-приложения и инфраструктуру Интернета. Модель данных и стратегия их постоянного хранения спроектированы для достижения высокой пропускной способности чтения и записи и обеспечивает простую масштабируемость с малой степени отказа. Сколько бы узлов ни требовалось приложению - один

или сто, – MongoDB сумеет обеспечить поразительно высокую производительность [11].

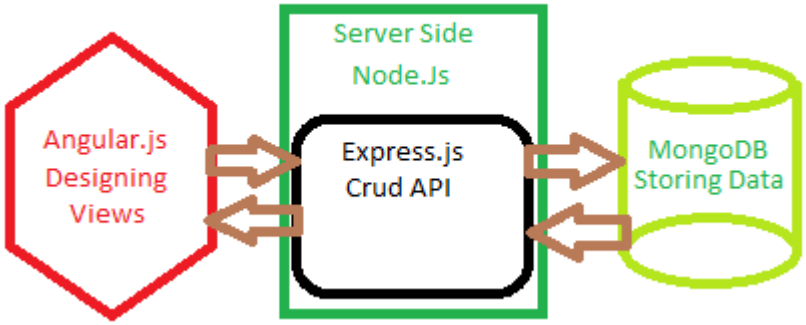

Рисунок 2.2.1 – Стек MEAN

Популярность MongoDB объясняется в первую очередь не стратегией масштабирования, а простой и удобной моделью данных. Учитывая, что с помощью документа-ориентированной модели можно представить развитые иерархически организованные структуры данных [12].

Для понятия работы этой СУБД рассмотрим пример – пусть моделируется товары для сайта интернет магазина. В полностью нормализованной базе данных информация об одном товаре может быть разбросана по десяткам таблиц. Чтобы получить его представление в интерактивной оболочке СУБД, придется написать сложный SQL-запрос с кучей соединений. Поэтому разработчики, как правило, обращаются к дополнительным программам, когда хотят собрать разрозненные данные в нечто осмысленное. Но если мы говорим о документной модели, то большая часть информации о товаре может быть представлена в виде одного документа. В оболочке MongoDB, построенной на основе языка javaScript, нетрудно получить полное представление о товаре в виде иерархически организованной JSON-подобной структуры. К ней можно предъявлять запросы, ей можно манипулировать. Средства составления запросов в MongoDB специально спроектированы для работы со структурированными документами, но так, чтобы пользователь, имеющий опыт работы с реляционными базами, располагал сравнимой выразительной мощью. К тому же, сегодня большинство разработчиков пишут на объектно-ориентированных языках, поэтому им нужно такое хранилище, которое было бы проще отобразить на объекты. В случае MongoDB объект, определенный на языке программирования, сохраняется «как есть» - без дополнительной сложности, привносимой системами объектно-реляционного отображения.

База данных в значительной мере определяется своей моделью данных. Модель данных MongoDB является документа-ориентированной. Продемонстрировать ее проще всего на примере (Рисунок 2.2.2).

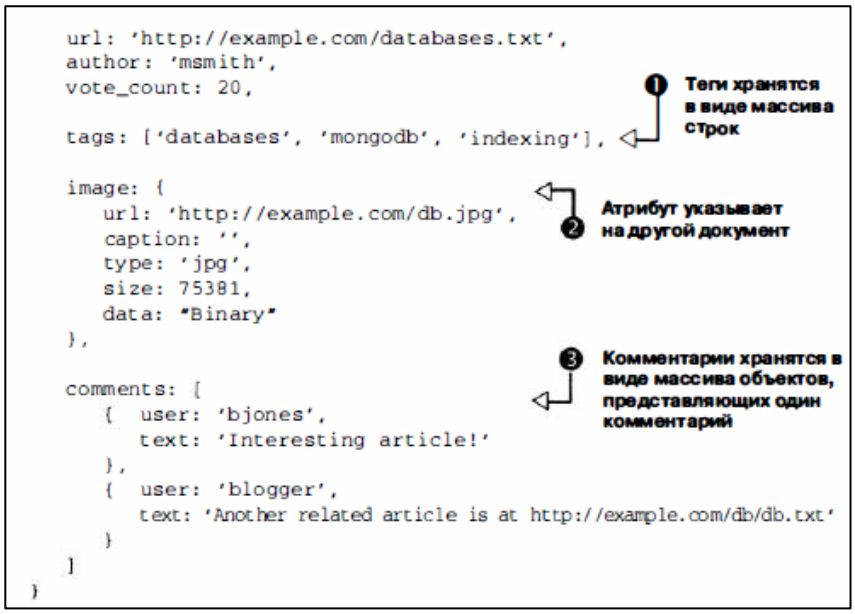

Рисунок 2.2.2 – Пример документо-ориентированной базы данных

В листинге 1.1 приведен пример документа, представляющего статью на социальном новостном сайте. Как видите, документ - это по существу набор, состоящий из имен и значений свойств. Значение может быть представлено простым типом, например: строки, числа и даты. Но может быть также массивом и даже другим документом. С помощью таких конструкций можно представлять весьма сложные структуры данных. Так, в нашем примере имеется свойство tags - массив, в котором хранятся ассоциированные со статьей теги. Но еще интереснее свойство comments O, которое ссылается на массив документов, содержащих комментарии.

2.2.2 Серверный модуль Express

Node – это платформа, а не среда разработки JavaScript-приложений. Распространенной ошибкой является отождествление Node со средой Rails или Django for JavaScript, так как на самом деле Node является гораздо более низкоуровневым инструментом. А если говорить о среде, в которой создается веб-приложение, то здесь нужно рассматривать более высокоуровневые технологии. В нашем стеке такой технологией является – Express.

Среда веб-разработки Express представляет собой надстройку над Connect (модуль который взаимодействует с клиентской частью), предлагая инструменты и структуру, которые делают разработку веб-приложений более быстрой, простой и увлекательной. В Express поддерживается унифицированная система представлений, позволяющая использовать практически любой шаблонизатор и простые утилиты для работы с различными форматами данных, передаваемыми файлами, маршрутными URL-адресами и т.п.

По сравнению с другими средами разработки приложений, такими как Django или Ruby на платформе Rails, Express гораздо компактнее. Философия, заложенная в Express, заключается в том, что приложения значительно

отличаются по требованиям и реализациям, а легковесная среда разработки позволяет смастерить именно то, что требуется, ничего лишнего. Как сама среда Express, так и сообщество Node-разработчиков ориентированы на компактные и модульные кирпичики функциональности, а не на монолитные среды разработки. В этой главе мы постепенно от начала до конца разработаем приложение для обмена фотографиями, чтобы на этом примере научиться создавать приложения с помощью Express.

Express не навязывает разработчику жесткую структуру приложения – вы можете размещать маршруты в любом количестве файлов, публиковать ресурсы в произвольной папке и т.п. Простейшее Express-приложение может быть весьма компактным, являясь тем не менее полнофункциональным HTTP-сервером (Рисунок 2.2.3).

2.2.3 Клиентский фреймворк Angular

Angular заимствует некоторые из лучших аспектов серверной разработки и использует их для расширения HTML-разметки в браузере. Таким образом, закладывается фундамент, который упрощает и облегчает создание приложений с расширенными функциональными возможностями. Угловые приложения строятся на основе шаблона проектирования MVC («модель - представление - контроллер»), который ориентирован на создание приложений, имеющих следующие характеристики [13]:

```
var express = require('express');
var app = express();
// Ответ на любой веб-запрос в папке /
app.get('/', function(req, res){
// Передаем "Hello" в качестве текста ответа
res.send('Hello');
});
// Слушаем порт 3000
app.listen(3000);
```

Рисунок 2.2.3 – HTTP-сервер на Express

- простота расширения: если вы понимаете основы, это будет легко понять даже в самом сложном приложении Angular, что означает, что вы можете легко расширить приложения для поддержки новых полезных функций;
- удобное сопровождение: угловые приложения легко отлаживаются, они легко исправляют ошибки, а это значит, что сопровождение кода легко в долгосрочной перспективе;
- удобство тестирования: Angular имеет хорошую поддержку для модульного и сквозного тестирования. Следовательно, вы можете найти и устранить дефекты до того, как ваши пользователи их обнаружат;

- стандартизация: Angular работает на основе внутренней функциональности браузера, не создавая никаких препятствий для вашей работы. Это позволяет создавать веб-приложения, отвечающие стандартам, в которых задействованы новейшие функции (например, различные API-интерфейсы HTML5), популярные инструменты и платформы.

Многие инструменты, используемые для разработки приложений на Angular, зависят от Node.js.

Для создания приложения работающего на Angular используют специальный модуль angular-cli [14]. Этот модуль стал стандартным инструментом для создания и управления пакетами Angular во время разработки. Создание угловых пакетов с нуля - довольно длительный и ненадежный процесс, который упрощается благодаря angular-cli. Чтобы установить angular-cli, откройте новую командную строку и введите следующую команду «npm install - global @ angular / cli @ 1.0.0».

После того, как вы установите Node.js, angular-cli, редактор и браузер, у вас есть все необходимое для запуска процесса разработки.

Выберите соответствующую папку и выполните следующую команду в режиме командной строки, чтобы создать новый проект с именем project: «ng new project».

Команда ng предоставляется пакетом angular-cli, а подкоманда ng new создает новый проект. В процессе установки создается папка с именем project, которая содержит все файлы конфигурации, необходимые для разработки Angular, некоторые временные файлы, упрощающие начальный этап разработки, и пакеты NPM, необходимые для разработки, запуска и развертывания приложений Angular.

NPM использует файл package.json для чтения списка программных пакетов, необходимых для проекта. Файл package.json создается angular-cli как часть инфраструктуры проекта, но он описывает только базовые пакеты, необходимые для разработки на Angular [15].

Инструментарий и базовая структура на месте; пришло время убедиться, что все работает нормально. Для этого нужно выполнить следующую команду из папки project: «npm serve --port 4200 –open».

Команда запускает сервер HTTP для разработки, который был установлен angular-cli и настроен для работы с инструментарием разработчика Angular.

Запуск занимает немного времени для подготовки проекта, а вывод выглядит примерно так (Рисунок 2.2.4).

2.2.4 Платформа Node.js

Попробуем ответить на вопрос, что такое Node.js. Хотя эта платформа появилась в 2009 году, она уже стала очень популярной среди разработчиков. Проект Node является вторым по популярности в GitHub, у него много последователей в группе Google и в IRC [16].

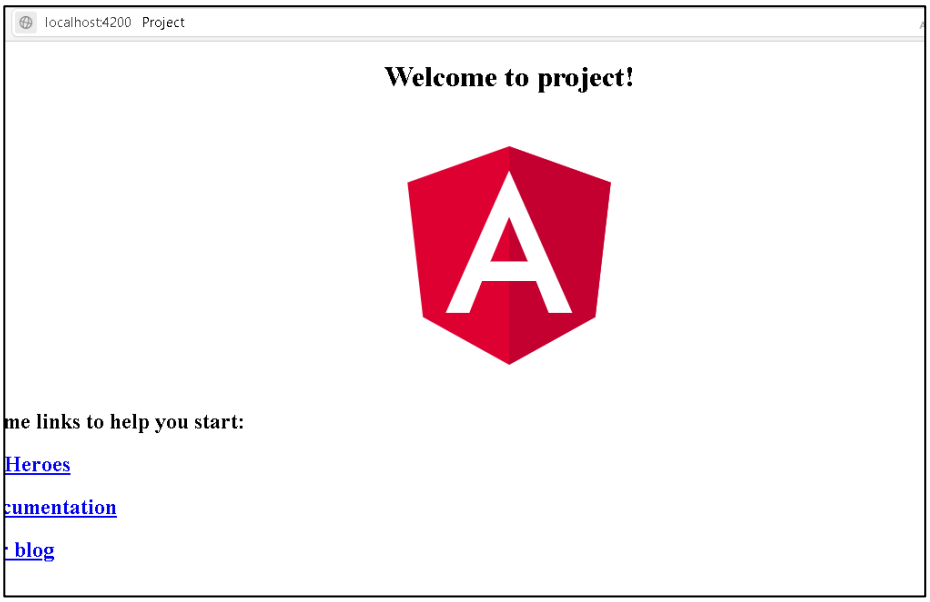

Рисунок 2.2.4 – Запуск приложения на Angular

На официальном веб-сайте Node определяется как «платформа, основанная на исполняемой библиотеке JavaScript Chrome, которая позволяет легко создавать быстрые, масштабируемые сетевые приложения. Node.js использует управляемую событиями неблокирующую модель ввода-вывода, легкую и эффективную, которая идеально подходит для разработки приложений реального времени, которые обрабатывают большие объемы данных и работают на распределенных устройствах.

Node имеет асинхронную платформу, управляемую событиями, для серверных JavaScript-приложений. Код JavaScript выполняется на сервере так же, как в браузере клиента. Если вы хотите понять, как работает Node, начните с изучения работы браузера.

Операции ввода-вывода в Node обрабатываются асинхронно и не блокируют выполнение скрипта, позволяя реагировать циклом событий на другие взаимодействия или запросы, возникающие на странице. В результате повышается чувствительность браузера, кроме того, становится возможным обрабатывать большое количество интерактивных взаимодействий на странице.

Платформа Node также включает базовый набор модулей для многих типов сетей и файлового ввода-вывода. В этот комплект входят модули для H TTP, TLS, H TTPS, файловой системы (POSIX), дейтаграммы (UDP) и NET (TCP). , Этот базовый набор (ядро) намеренно сделан компактным, низкоуровневым и несложным по структуре; он включает только «строительные блоки» для приложений ввода / вывода [17].

Модули независимых производителей, созданные на основе этих блоков, являются более абстрактными и позволяют решать задачи общего характера.

Node – это платформа, а не среда разработки приложений JavaScript. Распространенной ошибкой является отождествление Node с Rails или Django

для JavaScript, поскольку Node на самом деле является инструментом более низкого уровня. Если вы заинтересованы в средах разработки веб-приложений, прочитайте соответствующий раздел книги, в котором обсуждается одна из самых популярных сред разработки Node, называемая Express [18].

Для установки Node.js достаточно зайти на главный сайт и скачать последнюю версию, после чего установить (рисунок 2.2.5).

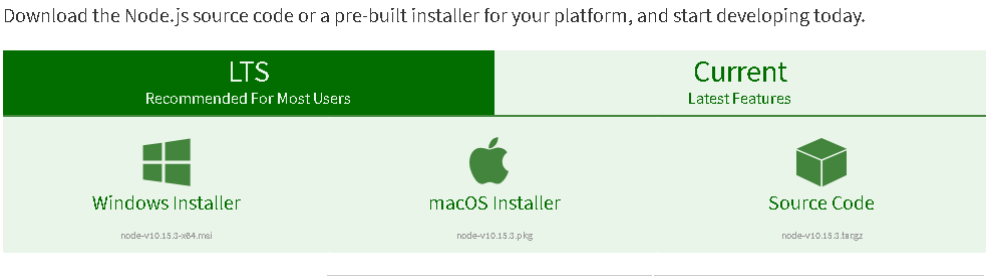

Рисунок 2.2.5 – Скачивание платформы Node.js

2.3 Модуль crypto для шифрования

Шифрование – настолько распространенная потребность, что даже в Node.js существует встроенный модуль под названием crypto. Он включает несколько методов для управления шифрованием данных, мы рассмотрим следующие два:

- randomBytes – метод генерации криптографически стойкой строки данных для использования в качестве соли;
- pbkdf2Sync – метод создания из пароля и соли хеша. pbkdf2 расшифровывается как основанная на пароле функция формирования ключа 2 (password-basedkey derivation function 2), это промышленный стандарт.

2.4 JWT-токен

Для создания аутентификации пользователей и определенич состояния сессии используется технология JWT-токена.

JWT состоит из трех выглядящих случайными разделенных точками частей (см. рисунок 2.4.1). Они могут быть довольно длинными, вот реальный пример.Для человеческого взгляда он довольно бессмыслен, но вы могли заметить две точки, а значит, три отдельные части [19]. Вот эти части.

- заголовок (Header) – закодированный объект JSON, содержащий тип и алгоритм хеширования;
- содержимое (Payload) – закодированный объект JSON, содержащий данные – собственно тело маркера;
- подпись (Signature) – зашифрованный хеш заголовка и содержимого, использующий секрет, известный только формирующему его серверу.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1 Auth.service.ts:42
Y2QzY2Y3YzRmOWE0YzQ1ODg2ZDg5MDUiLCJ1c2VybmFtZSI6InNhbG9pZyIsImV4cCI6M
TU1ODM0ODYyNCwiaWF0IjoxNTU3NzQzODI0fQ.7BINKqtsnYvBSiPuxSXSZdHbf2zDmD6
UtmnoCLafj6c
```

Рисунок 2.4.1 – Содержание JWT-токена

Обратите внимание на то, что первые две части не зашифрованы, они закодированы. Это значит, что декодировать их браузеру – и, конечно, другим приложениям – несложно. В наиболее современных браузерах имеется функция atob() для декодирования строки Base64. Родственная ей функция btoa() выполняет кодирование *в* формат строки Base64.

Третья часть, подпись, зашифрована. Для расшифровки ее вам необходимо воспользоваться секретом, задаваемым на сервере. Этот секрет следует хранить на сервере и никогда не обнародовать.

Хорошая новость заключается в том, что существуют библиотеки для всех сложных частей этого процесса. Так что установим одну из этих библиотек в наше приложение и создадим метод схемы для генерации JWT.

Первый шаг к генерации JWT – включение из командной строки модуля npm под названием jsonwebtoken: «npm install jsonwebtoken –save»

Далее нам необходимо выполнить его запрос (см. рисунок 2.4.2).

```
var jwt = require('jsonwebtoken');
```

Рисунок 2.4.2 – Запрос на JWT-токена

Наконец, необходимо создать метод схемы, который мы назовем generateJwt. Для генерации JWT нужно предоставить содержимое, то есть данные, и значение секрета.

Также необходимо задать для маркера дату истечения срока действия, после которой пользователю вновь придется выполнить вход для генерации нового JWT. Мы воспользуемся зарезервированным для этого в JWT полем exp, в котором дата истечения срока действия должна указываться в формате числового значения Unix (см. рисунок 2.4.3).

```
return jwt.sign({
  _id: this._id,
  username: this.username,
  exp: parseInt(expiry.getTime() / 1000),
```

Рисунок 2.4.3 – Объявление JWT-токена

2.5 Технология безопасности Helmet

Helmet помогает защитить приложение от некоторых широко известных веб-уязвимостей путем соответствующей настройки заголовков HTTP.

Helmet, по сути, представляет собой набор из девяти более мелких функций промежуточной обработки, обеспечивающих настройку заголовков HTTP, связанную с защитой:

- csp задает заголовок Content-Security-Policy для предотвращения атак межсайтового скриптинга и прочих межсайтовых вмешательств;
- hidePoweredBy удаляет заголовок X-Powered-By;
- hpkp добавляет заголовки Public Key Pinning для предотвращения атак посредника (атак “человек посередине”) с поддельными сертификатами;
- hsts задает заголовок Strict-Transport-Security, принудительно активирующий защиту соединений с сервером (по протоколу HTTP с использованием SSL/TLS);
- ieNoOpen задает заголовок X-Download-Options для IE8+;
- noCache задает заголовок Cache-Control и заголовки Pragma для отключения кеширования на стороне клиента;
- noSniff задает заголовок X-Content-Type-Options для защиты браузеров от прослушивания (сниффинга) MIME ответов с отличным от объявленного типом содержимого (content-type);
- frameguard задает заголовок X-Frame-Options для защиты от кликджекинга;
- xssFilter задает заголовок X-XSS-Protection для активации фильтра XSS (фильтра межсайтового скриптинга) в большинстве современных веб-браузеров.

2.6 Используемый редактор текста

В разработке приложений может использоваться любой редактор для программистов. Выбор возможных вариантов огромен. В некоторых редакторах предусмотрена расширенная поддержка работы с Angular, включая выделение ключевых терминов и хорошую интеграцию с инструментарием.

Один из важнейших критериев при выборе редактора – возможность фильтрации содержимого проекта, чтобы вы могли сосредоточиться на работе с некоторым подмножеством файлов. Проект Angular может содержать кучу файлов, многие из которых имеют похожие имена, поэтому возможность быстро найти и отредактировать нужный файл чрезвычайно важна. В редакторах эта функция может быть реализована разными способами: с выводом списка файлов, открытых для редактирования, или возможностью исключения файлов с заданным расширением. Среди самых популярных редакторов можно выделить: Sublime, Atom, WebStorm, VisualStudio Code. Было решено остановить свой выбор на Атоме.

Atom – это текстовый редактор с открытым исходным кодом, который можно использовать в качестве интегрированной среды разработки (IDE) для

широкого спектра языков программирования. Это может открыть много возможностей благодаря постоянной поддержке этого сообщества.

Приложение имеет все функции, которые вы ожидаете от редактора кода, включая подсветку синтаксиса, автоопределение языков, автозаполнение текста, возможность использования нескольких панелей и сохранения проекта в нескольких папках, поддержку фрагментов кода и мощный инструмент поиска. , И основной особенностью является модульность среды разработки, в которой вы можете добавлять дополнительные функции, устанавливая дополнительные пакеты. GitHub также включает в себя систему управления Git, поэтому вы можете публиковать любой контент, используя платформу GitHub.

Atom – это мультиплатформенный (Windows, Linux и Mac) инструмент, позволяющий сделать ваши проекты кроссплатформенными. Настройки IDE можно назвать основным преимуществом приложения. На момент написания статьи было доступно более 2000 пакетов и 600 тем. Учитывая многочисленные настройки программы и количество уже включенных в нее функций, ее, безусловно, можно назвать одним из лучших инструментов веб-разработки, доступных сегодня. Помимо этого, приложение выделяется на фоне других аналогичных альтернатив тем, что оно не занимает много места на компьютере.

3 Создание приложения с помощью выбранных технологий

3.1 Описание создаваемого проекта

Наш сайт представляет собой небольшой сервис для публикации различных постов с возможностью прикреплять фотографии, данные посты пишутся внутри сообществ, которые в свою очередь сортируются по различным категориям. Помимо этого чтобы писать посты пользователь должен быть авторизован, соответственно будет создан сервис аутентификации.

Таким образом у нас будет 3 страницы – страница с регистрацией и входом, страница с выбором категории и отображением соответсвующих сообществ, а так же страница с самим сообществом, где будут написать или просмотреть посты.

3.2 Проектирование базы данных

MongoDB скорее хранилище документов, а не традиционная табличная БД со строками и столбцами. Это дает MongoDB колоссальную свободу и гибкость, но иногда нам желательна, то есть необходима, структура для данных. Для взаимодействия нашего приложения и Бд будем использовать специальный модуль – Mongoose. Он предоставляет функцию-конструктор для описания новых схем, которую обычно присваивают переменной, чтобы можно было позднее к ней обратиться. Это выглядит так (Рисунок 3.2.1).

```
const UserSchema = new mongoose.Schema({})
```

Рисунок 3.2.1 – Объявление схемы в MongoDB

Пустой объект внутри скобок mongoose.Schema({ }) – то место, где мы будем описывать схему.

Теперь приступим к реализации необходимых моделей. При создании схем будут использоваться только строковые значения, так же в случае необходимости (например - поле username) необходимо указать уникальность и обязательное заполнение - required. В папке Models создаем модели данных для пользователей(см. рисунок 3.2.2), сообществ (см. рисунок 3.2.3) и постов(см. рисунок 3.2.4).

Во время работы с данными приложение не взаимодействует непосредственно со схемой – взаимодействие происходит через модели.

Модель – это скомпилированная версия схемы. Именно благодаря этому взаимно однозначному соответствию модель может создавать, читать, сохранять и удалять данные.

Компиляция схемы в модель – чрезвычайно простая задача, на одну строчку. Нужно только убедиться, что схема закончена, прежде чем вызывать команду model.

```
const UserSchema = new mongoose.Schema({
  username: {
    type: String,
    unique: true,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  name:String,
  surname:String,
  hash: String,
  salt: String
});
```

Рисунок 3.2.2 – Модель для пользователей

```
const BranchScheme = new mongoose.Schema({
    name: {
      type: String,
      unique: true,
      required: true
    },
    tree:{
      type: String,
      unique: false
    },
    text:{
      type:String,
      require:true
    }
});
```

Рисунок 3.2.3 – Модель для сообществ

```
const PostScheme = new mongoose.Schema({
    name: {
      type: String,
      unique: true,
      required: true
    },

    owner:{
      type: String,
      unique: false
    },
    branch:{
      type: String,
      unique: false,
      required: true
    },
    text:{
      type: String,
      unique: false,
      required: true
    },
    img:{
      type: String,
      unique: false
    }
});
```

Рисунок 3.2.4 – Модель для постов

3.3 Проектирования архитектуры приложения

В нашей разработке используется подход MVC. MVC расшифровывается как Model – View – Controller (модель – представление – контроллер), и эта архитектура нацелена на отделение друг от друга данных (модели), отображения (представления) и логики приложения (контроллера). Цель такого разделения – уничтожить тесные связи между компонентами, что теоретически должно сделать код более удобным для сопровождения и повторного использования. В нашем случае модель – это данные, которые хранятся в БД, контроллеры – это функции, которые будут изменять и получать данные из модели, а представление – это то, как эти данные будут отображаться в браузере.

Детальная архитектура проекта представлена ниже (см. рисунок 3.3.1).

Упрощенно этот цикл работает примерно так:

- в приложение поступает запрос;
- запрос маршрутизируется к контроллеру;

- контроллер, если необходимо, выполняет запрос к модели;
- модель возвращает ответ контроллеру;
- контроллер отправляет ответ представлению;
- представление отправляет ответ инициатору первоначального запроса.

Проектирование функциональной структуры WEB-приложения стало следующей задачей после того как стало понятно какие функции будет выполнять приложение (рисунок 3.3.2).

В этой дипломной работе не будет описываться процесс создания макета или его верстки, поскольку это не так важно в рамках задач этой работы.

Описание же начнется с создания необходимой серверной части на Node, далее создадим клиентскую часть на Angular и затем добавим аутентификацию для приложения.

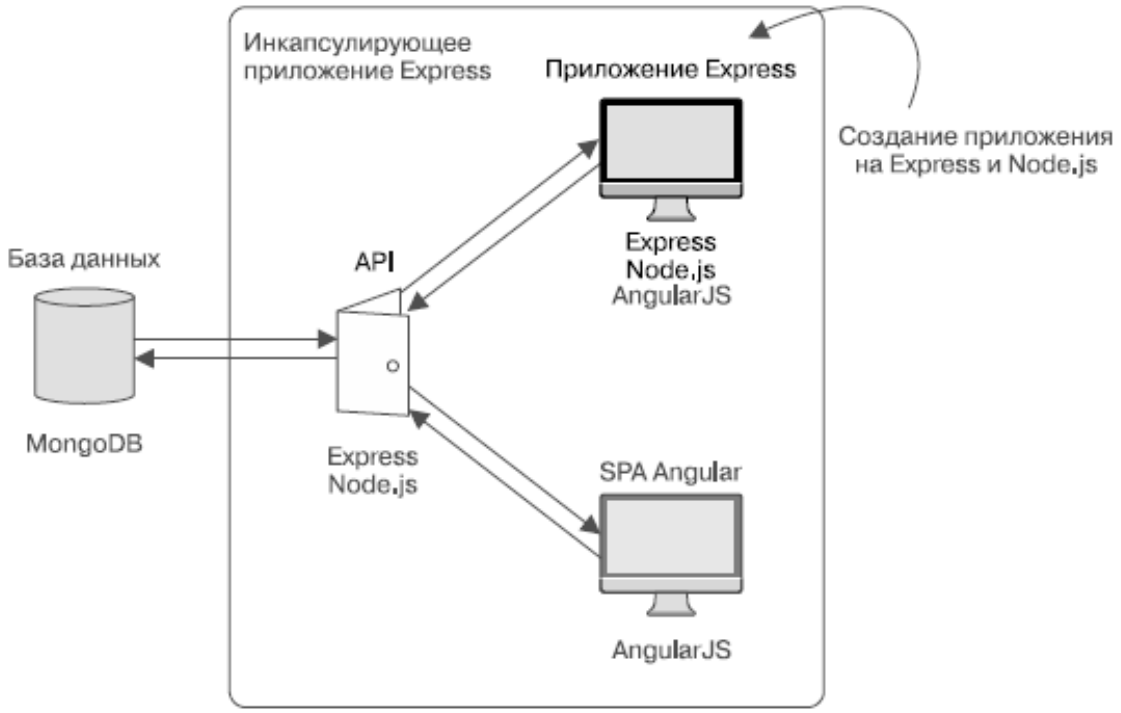

Рисунок 3.3.1 – Архитектура взаимодействия компонентов приложения

3.4 Создание серверной части с помощью на Node

Серверная часть заключается в создании системы, которая помимо того, что отображало наше приложение в браузере, выполняла следующие функции:

- принятие и обработка запросов от клиента, а так же формирование ответов;
- создание запросов к базе MongoDB;
- создание api для аутентификации;
- создание api для загрузки и хранении файлов.

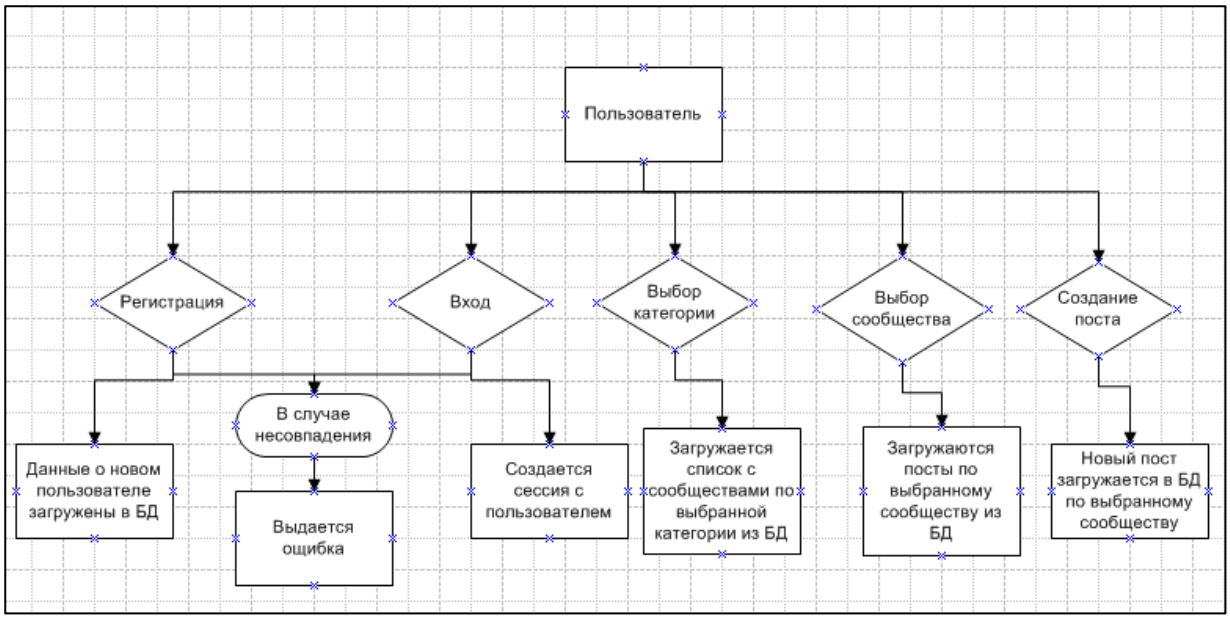

Рисунок 3.3.2 – Функциональная схема приложения

На данном этапе будем считать что у нас установленны все необходимые базовые элемента для работа стека, а именно – Node.js, MongoDB и пакетный менеджер npm.

Для начала создадим корневой каталог diplom, именно здесь будет хранится наш веб-проект. Исходя из указанной ранее схемы работы приложения – создаем в корневом каталоге следующие папки: config(для настройки сервера), DB(настройка базы данных), Controllers(для контроллеров), Models(для моделей базы), routes(для хранения маршрутов).

Затем нужноразобратьсяс модулями, которые нам понадобятся на серверной части исходя из предыдущей главы. Их условно можно разделить на:

- модули для функционирования сервера – expresss, http, path, helmet;
- модули для работы с базой данных и файлами – mongoose, multer;
- модули для работы сервиса аутентификации – passport, crypto.

Для установки модуля в node используется пакетный менеджер npm, команда выглядит так (рисунок 3.4.1).

```
B:\MEAN\diplom-sp>npm i express
```

Рисунок 3.4.1 – Установка модулей

После загрузки этих модулей у нас появится папка node-modules, где будут хранится загруженные модули, а так же файл package-json, где они будут объявляться. Вызывать загруженные и самописанные модули можно с помощью специальной команды – require.

Создание сервера начнем с исполняемого файла приложения, назовем его script.js. Именно при запуске этого файла приложение будет

устанавливаться на определенный порт и в следствии чего отображаться в браузере.

Так же создадим файл настройки сервера, где будет хранится номер порта. В папке config создаем файл– web-server.js, вкотором и указываем номер порта – 1337 и делаем его доступным для других файлов.

Далее перейдем к созданию архитектуры проекта по типу MVC.

Из подхода MVC на серверной части будут реализованы – модели и контроллеры. Модели будут хранить данные, а контроллеры обращаться к ним. Итак, следующий шаг в процессе – создание БД и модели данных.

Мы можем напрямую соединить наше приложение с MongoDB и заставить их работать вместе посредством нативного драйвера. Хотя нативный драйвер MongoDB обладает широкими возможностями, работать с ним нелегко. К тому же у него нет встроенного способа описания и сопровождения структур данных. Mongoose предоставляет большую часть функциональности нативного драйвера, но более удобным способом, спроектированным в расчете на включение в технологические процессы разработки приложений (рисунок 3.4.4).

Создание соединения Mongoose представляет собой просто объявление URI для вашей БД и передачу его методу connect Mongoose.

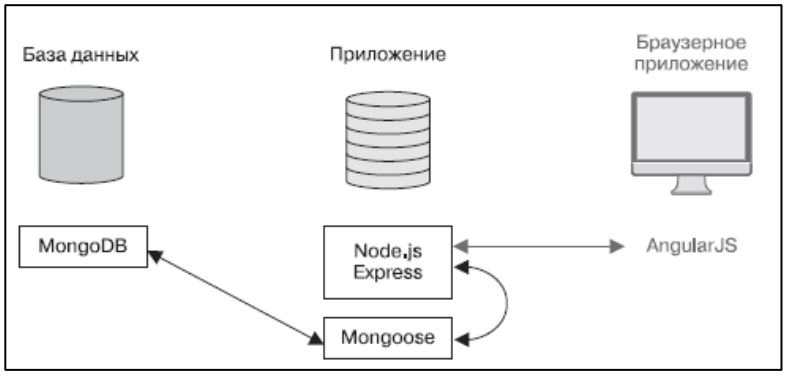

Рисунок 3.4.4 – Подключение модуля mongoose

В папки DB создаем конфигурационный файл database.js – здесь мы будум подключать к express базу данных через mongoose (рисунок 3.4.5).

Все взаимодействия данных с помощью Mongoose проходят через модель. Так как мы уже создали модели данных для нашего приложения, далее нам необходимо двинуться на шаг вперед – к Контроллерам, которые непосредственно будут обращаться к моделям. Контроллеры представляют собой API, которое будет создавать запросы к базе.

```
async function start() {
  mongoose.connect("mongodb://localhost:27017/Clients", { useNewUrlParser: true }
   if(err){
    console.log('Error');
  } else{
    console.log('Connected');
  }
});
```

Рисунок 3.4.5 – Подключение модуля mongoose

У нас уже есть база данных MongoDB и созданные модели, но взаимодействовать с ними мы можем только через командную оболочку MongoDB. Сейчас создадим API REST (рисунок 3.4.6), что даст выполнять типичные функции CRUD: создание, чтение, обновление и удаление. В основном мы будем иметь дело с Node и Express, применяя Mongoose в качестве вспомогательного средства при взаимодействии.

Начнем с напоминания о том, что такое API REST.

REST расшифровывается как «передача состояния представления» (REpresentational StateTransfer) и представляет собой скорее архитектурный стиль, чем жесткий протокол. В REST отсутствует сохранение состояния, он ничего не знает о состоянии или истории действий текущего пользователя;

API – аббревиатура для Application Program Interface – интерфейс программирования приложений, который дает приложениям возможность общаться друг с другом.

Ваш API должен возвращать данные в непротиворечивом формате. Обычные для API форматы – XML и JSON. Для нашего API будем использовать JSON, поскольку он естественным образом подходит для стека MEANи занимает меньше места, чем XML, а значит, поможет ускорить время ответа API.

Наш API будет возвращать для каждого запроса одно из трех:

- объект JSON, содержащий данные, отвечающие на исходный запрос;
- объект JSON, содержащий информацию об ошибке.

Контроллеры, которые и реализуют API REST получают запрос от соответствующих маршрутов и решают что делать дальше – какие данные отправлять, что делать с полученными данными, к каким моделям базы обращаться и т.д.

В нашем приложении присутствуют только функции ввода и получения данных из базы. Приступим к реализации контроллеров, которые в нашем приложении могут быть пользовательскими или для работы с постами.

Для нашего API будем использовать JSON, поскольку он естественным образом подходит для стека MEANи занимает меньше места, чем XML, а значит, поможет ускорить время ответа API.

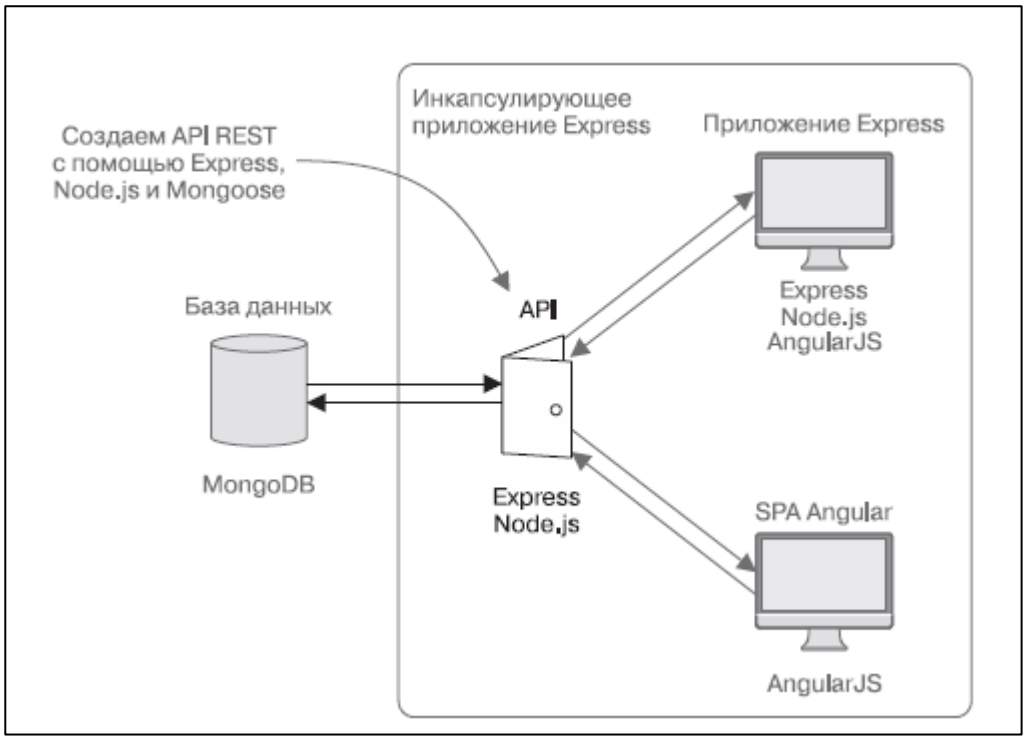

Рисунок 3.4.6 – Создание API REST

Для этого в папке Controllers создаем 2 файла: posts.js и profile.js. Контроллеры будут принимать req от клиентской стороны, которая содержит определенную информацию и затем генерировать res сообщение для ответа пользователю. В случае ошибки на сервере или в запросе должен будет отправить соответствующий статус или запрос.

К пользовательским контроллерам относятся:

- для регистрации нового пользователя, данный контроллер будет обращаться к созданной модели User и создавать в ней новый документ с пользователем. Как уже упоминалось ранее – в случае отправление пустого имени или пароля, а так же если пользователь с таким именем есть – генерируется определенный запрос с ошибкой (рисунок 3.4.8);
- для входа пользователя, данный контроллер ищет соответствия между полученными пользовательскими данными и данными в базе. Здесь так же ведется работа с моделью User. В случае нахождения соответствия возвращается успешный запрос, в противном случае – ошибка (рисунок 3.4.9).

Однако этот контроллер в дальнейшем будет модернизирован при создании сервиса аутентификации, с использованием Passport и JWT.

Для работы с постами:

- загрузка постов из сервера, данный контроллер получает название сообщества и возвращает список всех возможных постов, которые к нему относятся (рисунок 3.4.10);
- загрузка категорий из сервера, данный контроллер возвращает список всех категорий, которые хранятся в базе (рисунок 3.4.11);

- загрузка сообществ из сервера, данный контроллер получает название категории и возвращает список всех возможных сообществ, которые к нему относятся (рисунок 3.4.12).

```
module.exports.signup = function(req, res) {
      console.log(req.body);
      if (!req.body.username || !req.body.password) {
        res.json({success: false, msg: 'Please pass username and password.'});
      } else {
        var newUser = new User({
          username: req.body.username,
          password: "none",
          name: req.body.name,
          surname: req.body.surname
        });
        newUser.setPassword(req.body.password);
        // save the user
        newUser.save(function(err) {
          if (err) {
            return res.json({success: false, msg: 'Username already exists.'});
          }
          res.json({success: true, msg: 'Successful created new user.'});
        });
      }
}
```

Рисунок 3.4.8 – Контроллер для регистрации

```
module.exports.login = function(req, res, next) {
  passport.authenticate('local', function(err, user, info){
       User.findOne({ username: username }, function (err, user) {
       if (err) { return done(err); }
       if (!user) {
         res.json({success: true, msg: 'Successful '});
       }
       res.json({success: false, msg:: 'Error'});
     });
 }
```

Рисунок 3.4.9 – Контроллер для входа

```
module.exports.getPost = function(req, res) {
  var name ;
  Branch.find({_id:req.body.id}).exec(function(err, post) {
      name = post[0].name;
      Post
        .find({branch:name})
        .exec(function(err, post) {
          // console.log(post);
          res.status(200).json(post);
        });
    });
}
```

Рисунок 3.4.10 – Контроллер для получения списка постов

- добавление поста, данный контроллер получает данные от клиента, в которые входит – название поста, текст поста и сообщество к которому он относится и сохраняет его в базе и возвращает обновленный список постов. Данный контроллер работает с моделью Post (рисунок 3.4.13);

```
module.exports.getTree = function(req, res) {
    Tree
      .find({})
      .exec(function(err, post) {
        console.log(post);
        res.status(200).json(post);
      });
}
```

Рисунок 3.4.11 – Контроллер для получения списка категорий

```
module.exports.getBranch = function(req, res) {
  console.log(req.body);
  if(req.body.name){
    Branch
      .find({tree:req.body.name})
      .exec(function(err, post) {
        console.log(post);
        res.status(200).json(post);
      });
  }
}
```

Рисунок 3.4.12 – Контроллер для получения списка сообществ

Рисунок 3.4.13 – Контроллер для добавления поста

Этот метод так же будет модернизирован при создании сервиса для загрузки фотографий к посту.

Далее спускаемся еще ниже по MVC и переходим к созданию маршрутов.

У нас будет файл api.js в каталоге routes, в котором будут находиться все используемые в API маршруты. Маршруты – это своего рода порты, которые находятся в режиме ожидания и ждут когда к ним придет запрос. После чего они перенаправляют данные запроса в контроллер. Начнем с того, что сделаем ссылку на этот файл в основном файле приложения script.js.

В api.js нам понадобятся маршруты для входа, регистрации, для добавления поста, для получения категорий, для получения списка сообществ и для получения постов. Для каждого из этих маршрутов необходима ссылка на контроллеры, поскольку маршруты служат просто средством установления соответствия, получая URL входящего запроса и устанавливая его соответствие конкретному элементу функциональности приложения. Для начала подключим созданные контроллеры и библиотеку для реализации маршрутизации в express (рисунок 3.4.15).

Затем создадим соответствие между маршрутами и необходимыми контроллерами (рисунок 3.4.16).

Теперь осталось только объединить все созданные элементы в приложение.

У нас имеются различные детали готового приложение, в которые входят – файлы настроек, api для работы с базой данных, контроллеры, модели данных. Все что осталось сделать – объединить их в файле инициализации – script.js.

```
router.post('/login', ctrlProfile.login);
router.post('/signup', ctrlProfile.signup);
router.get('/tree', ctrlPosts.getTree);
router.post('/branch', ctrlPosts.getBranch);
router.post('/post', ctrlPosts.getPost);
router.post('/addpost', ctrlPosts.addPost);
```

Рисунок 3.4.16 – Объединение маршрутов в файле

Для начала в контроллерах добавляем созданные модели, чтобы была возможность обращаться к ним (рисунок 3.4.17).

```
var mongoose = require('mongoose');
var Tree = mongoose.model('Tree');
var Branch = mongoose.model('Branch');
var Post = mongoose.model('Post');
```

Рисунок 3.4.17 – Добавление моделей в контроллеры

В файле script.js подключаем необходимые модули и скрипты для работы приложения (рисунок 3.4.18).

```
const dbconnect = require('./DB/database.js');
var express = require('express');
var path = require('path');
var bodyParser = require('body-parser');
var route = require('./routes/api.js');
var app = express();
var http = require('http');
var server = http.createServer(app);
```

Рисунок 3.4.18 – Контроллер для получения списка сообществ

Затем добовляем функцию подключения к базе данных и созданные маршруты (рисунок 3.4.19).

Делаем возможным чтение передаваемых файлов и записей Json-формата (рисунок 3.4.20), а так же подключаем модуль защиты helmet (рисунок 3.4.19).И наконец, запускаем наш сервер на порт 1337 (рисунок 3.4.22). Все, серверная часть готова, для ее запуска заходим в командную строку и пишем команду.

```
var helmet = require('helmet')
app.use(helmet())
app.use('/api', route);
dbconnect.start();
```

Рисунок 3.4.19 – Подключение к базе данных

```
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended:false}));
app.use(function(req, res, next) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header('Access-Control-Allow-Methods', 'PUT, GET, POST');
    res.header("Origin, X-Requested-With, Content-Type, Accept");
    next();
});
```

Рисунок 3.4.21 – Добавление поддерживаемых форматов

```
server.listen(conf.port, function () {
  console.log('Listining port 1337...');
});
```

Рисунок 3.4.22 – Запуск сервера на порт 1337

Теперь наш сервер может получать, передавать и сохранять данные, однако для их отображения нам необходимо создать клиентскую часть. Этим и займемся.

3.5 Добавление клиентской части с помощью Angular

Пришло время посмотреть на клиентскую часть стека MEAN – Angular. В нашем приложении он выполняет функцию представления в модели MVC. Подобно Express, Angular позволяет разделять обязанности, работая с представлениями, данными и логикой в различных областях. Чтобы создать приложение на Angular нужно выполнить следующую команду в терминале, находясь в каталоге diplom: «ng new diplom».

Установка всех зависимостей для проекта занимает некоторое время, поскольку Angular CLI настраивает наше приложение. По завершении установки вы увидите папку с именем diplom в текущем каталоге, так же файлы angular.json. Можно скомпелировать приложение прямо сейчас, перейдя в каталог diplom, а затем запустив ng-build в консоли. Однако наше приложение будет запускаться с помощью node.js и отбражаться на указанном порте, для этого есть команда ng build, которая компилирует angular приложение в набор различных файлов в папку dist, к которым может получить доступ сервер. Для этого запускаем данную команду и возвращаемся к нашему файлу – script.js (рисунок 3.5.4) и в нем добавляем строку (рисунок 3.5.3).

```
B:\MEAN\diplom-sp>ng build
Your global Angular CLI version (7.3.8) is greater than your local
version (7.3.4). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".
 25% building 69/70 modules 1 active ...m-sp\node_modules\jquery\dist\jquery.js
```

Рисунок 3.5.2 – Компиляция приложения

```
app.use(express.static(path.join(__dirname,'dist/diplom')));
```

Рисунок 3.5.3 – Доступ к приложению через сервер

Эти строки необходимо добавить после подключения маршрутов. Теперь в случае получения роутов, которые не относятся к api.js сервер будет загружать файлы из папки dist и отбражать их в пользовательском окне.

Одним из ключевых элементов приложения являются компоненты. Компонент управляет отображением представления на экране. Для его создания используется команда: «ng g c my app».

Потом у нас появляется пустой компонент my-app, который имеет следующую структуру (рисунок 3.5.5).

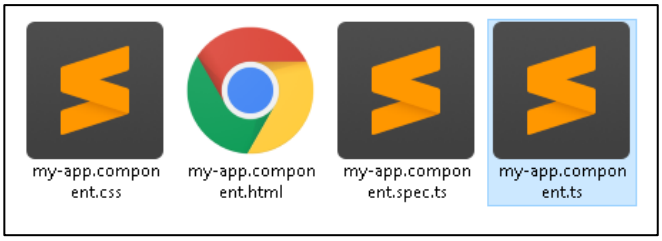

Рисунок 3.5.5 – Создание компонента в Angular

Файл с расширением html содержит HTML-код шаблона, css содержит список стилей для компонента, spec.ts отвечает за компиляцию компонента, а ts содержит логику нашего компонента и является аналогом JavaScript файла.

В ts файле находятся функции, переменные и т.д. Так как составляющие компонента тесно взаимодействуют, то переменную, созданную в ts можно отобразить в шаблоне HTML.

Итак, теперь сделав структуру Angular, приступим к созданию клиентской части нашего приложения.

На главной страницы будет отображаться: «шапка сайта», список категорий, при выборе которых будут загружаться блоки сообществ, которые относятся к выбранной категории, после нажатия на которые появляется страница этого сообщества. Так же нам потребуются формы регистрации и входа для пользователей. Поэтому нам нужно создать 4 компонента: header («шапка» сайта, которая будет везде повторяться), main(отображение категорий и сообществ), page(страница с выбранным сообществом), login(вход и регистрация).

Для начала создадим компонент для header-а сайта.Его HTML-код внесем в header.component.html (рисунок 3.5.7).

```
<nav class="menutopone">
<ul class="hospital">
<li class="menuitem1"><a href="#">Главная</a></li>
<li class="menuitem2"><a href="#">О нас</a></li>
<li class="menuitem3"><a href="#">Контакты</a></li>
<li class="menuitem4"><a href="#">Помощь</a></li>
</ul>
</nav>
```

Рисунок 3.5.7 – Создание компонента header

Добавим CSS-стили в header.component.css, а так же мы будем подставлять имя пользователя в случае удачного входа в правой части хедера, поэтому в ts файле создаем переменную username и вставляем ее в html-шаблон. Но так как этот блок с именем появляется только когда пользователь вошел – ставим флаги, по которым для не авторизованных пользователей высвечивается кнопка «Войти». И реализуем их в шаблоне (рисунок 3.5.9).

```
<button class="buttonnumone" *ngIf="!flag" >{{username}}</button>
```

Рисунок 3.5.9 – Создание компонента header

Создадим компонент login дляаутентификации пользователей. В html файле этого компонента создадим формы входа и регистрации (рисунок 3.5.10).

```
<form>
<label for="inputEmail" class="sr-only">Email address</label>
<input type="email" class="form-control" placeholder="Email address"
 [(ngModel)]="loginData.username" name="username" required/>
<label for="inputPassword" class="sr-only">Password</label>
<input type="password" class="form-control" placeholder="Password"
 [(ngModel)]="loginData.password" name="password" required/>
<button type="submit" class="btn btn-default">Sign in!</button>
</form>
```

Рисунок 3.5.10 – Создание компонента login

Далее переходим к созданию главной странице. Так же создаем для нее компонент. На этой странице будет отражаться категории, при выборе которых загружается список сообществ. Поэтому создаем две переменные с типом any: trees, barnches.

Так как значений может быть много – создаем в html-шаблоне цикл ng-for с помощью которого - можно создать неограниченное количество одинаковых блоков с разным содержанием (рисунок 3.5.12).

```
<div class="engel">
<div class="divindivin" *ngFor="let tree of trees" (click)="select($event)" [id]="tree.name">
<p class="imgdiv2"><img  width="75px" src="assets/img/8.png"></p>
<p>{{tree?.name}}</p>
</div>
</div>
 <div class="divofbutton"><button class="buttonpux" (click)="checks()">ВЫБРАТЬ</button></div>
```

Рисунок 3.5.12 – Создание компонента login

Для списка сообществ создаем отдельный блок, с использованием ng-for (рисунок 3.5.13).

```
<p class="text05">Доступные ветки</p>
<div class="boxinformation" *ngFor="let branch of branches" id="branch?.name">
<div class="box1information">
<div class="divdiv"><img src="assets/img/19.png"><div class="divdivdvi">
<span class="text06">Название:</span> <span class="text07">{{branch?.name}}
</span></div><p class="text08">Оценка: <div>
<img style="margin-top:8px;margin-left:5px" src="assets/img/20.png"></div></div>
<p class="text09"></p>
<button class="butttonlastone">Перейти →</button>
</div>
</div>
```

Рисунок 3.5.13 – Создание компонента login

Переходим к созданию страницы сообщества – снова создаем компонент page , в шаблоне которого так же используются переменные с названием, описанием сообщества (рисунок 3.5.14).

```
<p class="textautoehex">Автор:<span> {{branch?.owner}}</span></p>
<p class="textimglasdt"><img src="assets/img/27.png"></p>
<div class="informationblockopen">{{branch?.about}}</
<div><img src=" branch?.img"><br>
</div>
<div class=textinformationblok1>
<p>
{{branch?.text}}
</p>
```

Рисунок 3.5.14 – Создание компонента login

Так же на странице сообщества будет форма для добавления поста (см. рис 3.5.15).

Основа для сайта заложена и после подключения стилей и прочих второстепенных элементов – у нас получился готовый шаблон сайта. Дальше нам нужно создать взаимодействие с сервером для заполнения данного шаблона информацией из базы данных.

Таким образом, на данном этапе разработки клиентской части у нас есть – главная страница (рисунок 3.5.16), страница со списком сообществ (рисунок 3.5.17), форма входа (рисунок 3.5.18) и регистрации (рисунок 3.5.19). Но они не подключены, поэтому представляют из себя пустые макеты.

Теперь подключим различные библиотеки для организации клиент-серверного взаимодействия.

Так же у нас создались формы для входа и регистрации на отдельных страницах, которые пока тоже не работают, поскольку нет подключения к базе.

Чтобы взаимодействовать с созданным сервером будем использовать ajax-запросы. Для этого создадим 2 сервиса: для аутентификации и для работы с постами.

```
<p>Название:</p>
<textarea class="twxtareanum2" type="text" maxlength="20" placeholder="Введите название"
[(ngModel)]="newPost.name" [ngModelOptions]="{standalone: true}"></textarea>
</div>
<div class="dinintosecondsection">
<p class="text51231">Текст:</p><textarea class="twxtareanum1" [(ngModel)]="newPost.text"
[ngModelOptions]="{standalone: true}"></textarea>
</div>
<label class="btn btn-default">
<input type="file" id="filein" (change)="selectFile($event)">
</label>
<p class="butcenter"><button (click)="addPost(); modal.dismiss('Cross click')">опубликовать пост</button></p>
```

Рисунок 3.5.15 – Создание компонента login

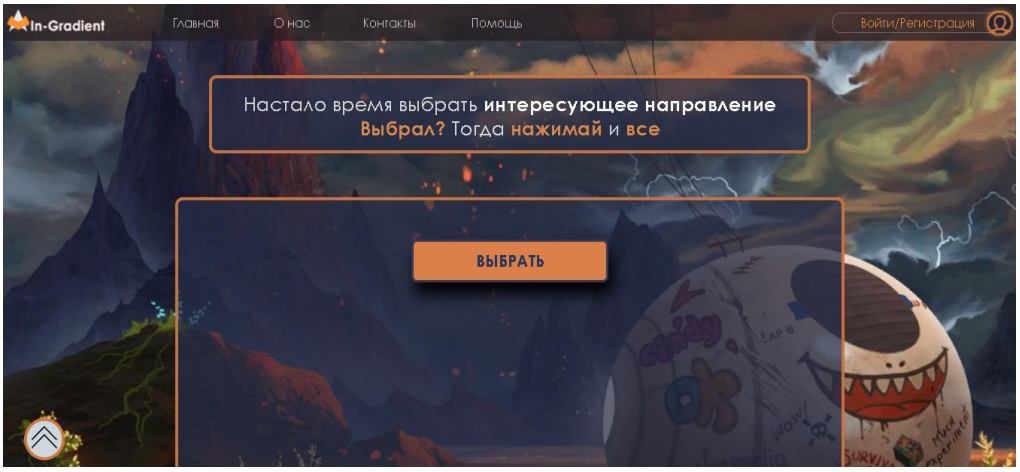

Рисунок 3.5.16 – Главная страница сайта

Рисунок 3.5.17 – Список доступных сообществ

Сервисы – автономные единицы функциональности, которые можно объединять для обеспечения полной функциональности приложения. Сервисы в Angular используются очень часто, поскольку выполнение большей части логики приложения желательно поручить им, чтобы обеспечить возможность его многократного применения несколькими контроллерами.

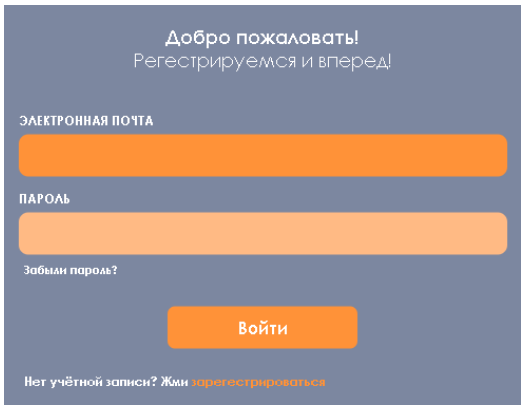

Рисунок 3.5.18 – Форма входа

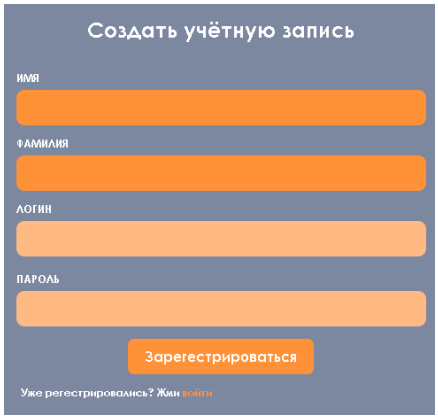

Рисунок 3.5.19 – Форма регистрации

В данном случае мы создадим информационный сервис, который сможем использовать где угодно.

Сервисы в Angular имеют расширение service.ts. Создаем сервис auth.service.ts и post.service.ts. В Angular для реализации технологии ajax имеется модуль http. Чтобы его подключить нужно его с помощью менеджера npm скачать, а затем объявить уже внутри сервиса.

В сервисе аутентификации должно быть 3 функции: для регистации (рисунок 3.5.21), которая передает данные из формы регистрации.

```
public register(user) {
  return this.http.post('/api/signup', user).subscribe(resp => {
    console.log(resp);
    if(resp['success']){
      this.router.navigate(['main']);
    }
  }, err => {
    // this.message = err.error.msg;
  });
}
```

Рисунок 3.5.21 – Создание сервиса аутентификации

Сервисы в Angular имеют расширение service.ts. Создаем сервис auth.service.ts и post.service.ts. В Angular для реализации технологии ajax имеется модуль http. Чтобы его подключить нужно его с помощью менеджера npm скачать, а затем объявить уже внутри сервиса (рисунок 3.5.20).

В данном случае мы создадим информационный сервис, который сможем использовать где угодно.

Для авторизации, которая передает данные из формы входа (рисунок 3.5.22).

```
public login(user){
  console.log(user);
  return this.http.post('/api/login',user).subscribe(resp => {
    console.log(resp);
    const url = this.router.url;
    if(url == '/main'){
      this.router.navigate([url]));
    }
  }
  }
```

Рисунок 3.5.22 – Создание сервиса аутентификации

В сервисе работы с постами должно быть 4 функции (загрузка постов, загрузка сообществ, загрузка категорий, а так же добавление поста) (рисунок 3.5.24).

Затем в главном компоненте создадим кнопки и функции для работы с сервисами. В первую очередь займемся регистрацией и входом. Для начала добавим в компоненты ссылку на сервис и инициализируем его в конструкторе (рисунок 3.5.25).

Теперь перейдем к обращению ко второму сервису. Для того чтобы сделать запрос к сервису и получить какие-то данные до того как загрузится

компонент в браузере в каждом созданом компаненте есть блок ngOnInit. Команды которого выполняются до загрузки html-шаблона.

```
public getTree(): Observable<any> {
    return this.http.get('/api/tree',{});
  }
  public getBranch(tree): Observable<any> {
    return this.http.post('/api/branch', tree);
  }
  public getPosts(tree): Observable<any> {
    return this.http.post('/api/post', tree);
  }
public addPost(post): Observable<any> {
    console.log(post);
    return this.http.post('/api/addpost', post);
  }
```

Рисунок 3.5.24 – Создание сервиса аутентификации

```
import { PostService } from '../post.service';
import { AuthenticationService } from '../Auth.service';
constructor(private post: PostService, private auth: AuthenticationService) { }
```

Рисунок 3.5.25 – Создание сервиса аутентификации

Категории как раз должны браться с базы данных и сразу отображаться при загрузке страницы. Поэтому помещаем в блок ngOnInit главного компонента обращение к функции получения категорий и присваиваем полученный результат переменной trees (рисунок 3.5.26).

В случае с загрузкой сообществ – они должны появляться при нажатии на кнопку, при чем если не выбрано ничего – загружаются сообщества, которые относятся к категории «Программирование».

Для того чтобы сделать запрос к сервису и получить какие-то данные до того как загрузится компонент в браузере в каждом созданом компаненте есть блок ngOnInit. Команды которого выполняются до загрузки html-шаблона.

После того как взаимодействие налажено можно занести в базу какие-либо значение, запустить сервер и посмотреть на результат, но перед этим нужно добавить маршрутизацию для перехода по разным страницам сайта (рисунок 3.5.29)

```
  getTree(){
   this.post.getTree().subscribe(tree => {
     console.log(tree);
     this.trees = tree;
     }, (err) => {
       console.error(err);
     });
 }
ngOnInit {
this.getTree();
}
```

Рисунок 3.5.26 – Создание сервиса аутентификации

Грубо говоря, в клиентской части у нас есть три разные страницы – с регистрацией /входом, страница сообщества и главная. Самое интересное в этом для нас в данный момент: откуда браузер знает, что именно отображать для различных URL. В Express мы настраивали набор маршрутов, указывающих на различные контроллеры. В Angular можем сделать нечто подобное, но сначала необходимо предотвратить перехват Express управления (рисунок 3.5.30).

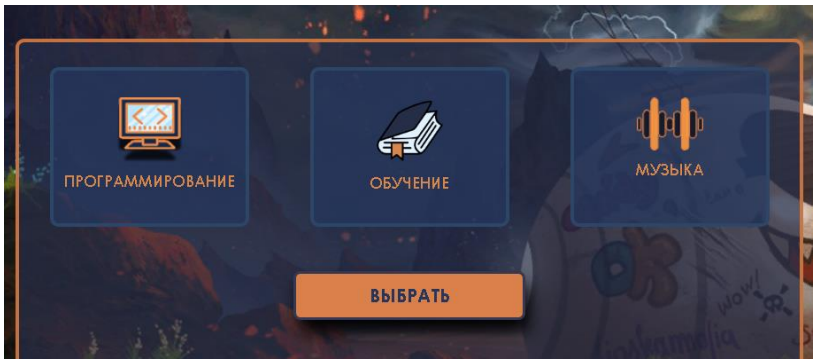

Рисунок 3.5.27 – Список доступных категорий

Теперь все приходящие маршруты, которые не относятся к api будут перенаправляться в angular. Ключевым для работы маршрутизации в Angular является модуль RouterModule, который располагается в пакете @angular/router. Маршрутизацию в этой среде представляет собой смену компонентов при получении определенных URL. Таким образом, что не происходит перезагрузка страницы – это и есть преимущество одностраничных приложений (SPA), создаваемых Angular. Чтобы реализовать это нужно в файле app.modile.ts создать массив Routes и добавить его в imports. Данный массив будет создавать соответствие между получаемыми URL и компонентами.

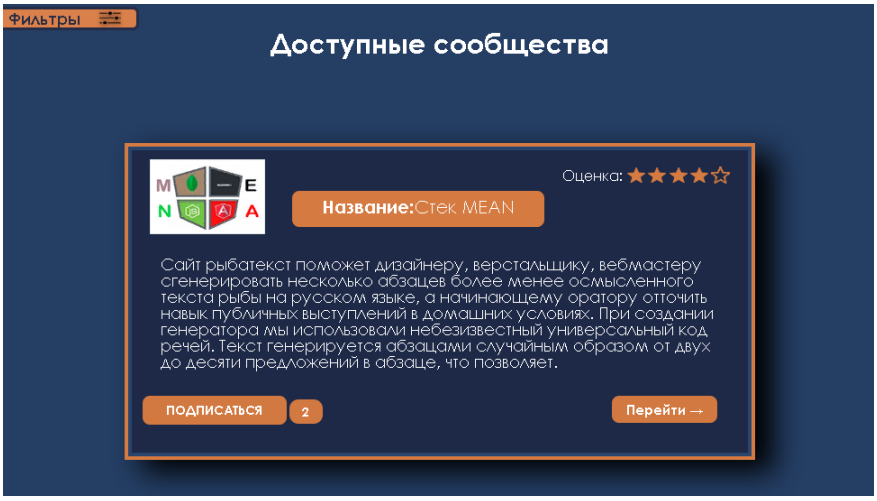

Рисунок 3.5.28 – Список доступных сообществ

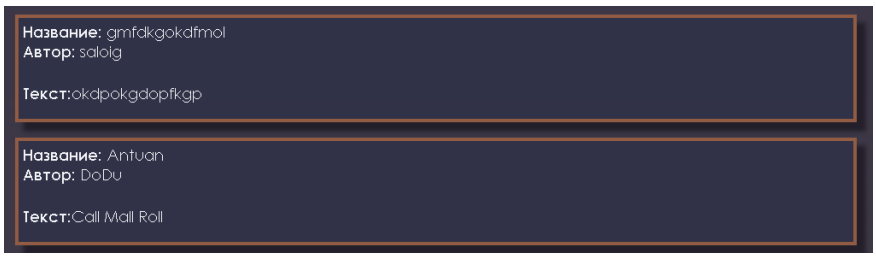

Рисунок 3.5.29 – Список постов из сообщества

```
app.get('*',function (req, res, next) {
  res.sendFile(path.join(__dirname,'dist/diplom/index.html'));
});
```

Рисунок 3.5.30 – Создание маршрутизации в Angular

Так как у нас будет много сообществ, которые используют один шаблон – делаем возможность загрузки данного компонента по id. При этом укажем, что если ничего не указано, то пользователя переносит на главную страницу.

Теперь у нас есть готовое приложение, которое позволяет пользователю получать данные(посты, категории, сообщества), а так же регистрироваться и добавлять посты. Можно заполнить таблицу пробными данными и посмотреть на результат. Осталось только реализовать функции клиент-серверной авторизации и загрузки файлов на сервер.

Аутентификацию в приложении можно так же разделить на клиентскую и серверную.

Реализуем процесс аутентификации на серверечерез passport и JWT. Mongoose позволяет добавить в схему методы, которые становятся доступными в качестве методов модели. Подобные методы дают коду прямой доступ к атрибутам модели. Создаем методы для хеширования пароля и его проверки, а так же метод для генерации JWT в модели пользователя:

В папке config создаем файл passport.js, который реализует нашу локальную стратегии авторизации пользователя (рисунок 3.5.32).

```
passport.use(new LocalStrategy({
    usernameField: 'username'
  },
  function(username, password, done) {
    User.findOne({ username: username }, function (err, user) {
      if (err) { return done(err); }
      if (!user) {
        return done(null, false, {
          message: 'User not found'
        });
      }
      return done(null, user);
    });
  }
));
```

Рисунок 3.5.32 – Локальная стратегия для пользователя

Традиционный подход к хранению данных пользователя в веб-приложении состоит в сохранении cookie-файла, и это, безусловно, заслуживающий рассмотрения вариант. Но cookie-файлы на самом деле предназначены для использования серверными приложениями: при каждом запросе к серверу cookie-файлы пересылаются в заголовке HTTP. В SPA это не требуется: конечные точки API не имеют состояния и не получают или не используют cookie-файлы. Поэтому будем использовать локальное хранилище данных в браузере. При использовании локального хранилища данные остаются в браузере и не передаются вместе с запросами.

Помимо этого, локальное хранилище очень удобно использовать с помощью JavaScript. Например следующий фрагмент кода, задающий и получающий определенные данные (рисунок 3.5.33).

```
localStorage.setItem('mean-token', token);
localStorage.getItem('mean-token', token);
```

Рисунок 3.5.33 – Обращение к локальному хранилищу браузера

В случае выхода нужно будет модифицировать нашу функцию выхода и входа (рисунок 3.5.34). Теперь у нас имеются методы для извлечения JWT с сервера, сохранения его в хранилище localStorage, чтения из localStorage, а также удаления.

```
public logout(){
    this.token = '';
    window.localStorage.removeItem('mean-token');
    this.router.navigate(['']);
  }
  public login(user){
    console.log(user);
    return this.http.post('/api/login',user).subscribe(resp => {
      console.log(resp);
      this.saveToken(resp['token']);
      const url = this.router.url;
      if(url == '/main'){
        this.router.navigateByUrl('/signup',
        {skipLocationChange: true}).then(() =>
        this.router.navigate([url]));
      }
    }
    }
```

Рисунок 3.5.34 – Функции входа и выхода с учетом JWT

Дата и время истечения срока действия JWT – часть содержимого, представляющего собой вторую часть данных. Как вы помните, эта часть –

просто закодированный объект JSON, он не зашифрован, так что можно его декодировать.

На этом изменении мы завершаем относящийся к аутентификации раздел. Пользователи должны быть аутентифицированы для получения определенных данных о пользователе из базы, а благодаря системе аутентификации опубликованным постам будет присвоено правильное имя пользователя.

3.6 Построение защиты для созданного приложения

Общедоступные веб-приложения интересны хакерам как ресурсы или инструменты заработка. Спектр применения полученной в результате взлома информации широкий: платное предоставление доступа к ресурсу, использование в бот-сетях и т. д [20]. Чтобы организовать безопасность приложения, нужно обезопасить уязвимые места системы. Для этого рассмотрим созданное приложение с точки зрения злоумышленника [21].

Первым делом определим, какие цели может преследовать злоумышленник – исходя из концепции информационной безопасности, атаки на веб-приложения могут быть направлены на нарушение конфиденциальности или доступности. Тогда в нашем случае цели, которые преследует хакер можно представить в виде рисунка 3.6.1.

Под конфиденциальными данными у нас в приложении будет выступать пароли пользователей, а нарушение работы сервера – ситуация при которой сервер «падает».

Теперь рассмотрим – через какие места может осуществить данные цели злоумышленник (рисунок 3.6.2).

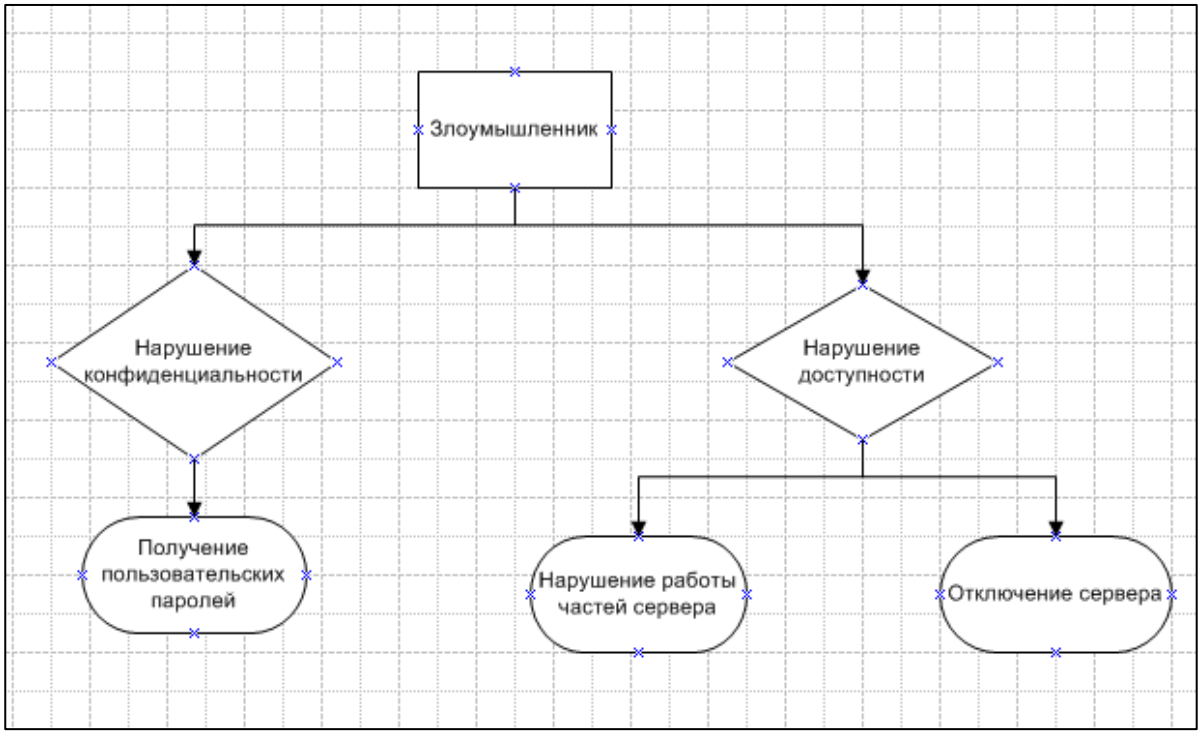

Рисунок 3.6.1 – Потенциальные цели злоумышленника

Каждый из этих компонентов представляет собой потенциальный источник утечки информации:

- перехват трафика до и после сервера веб-приложения;
- получение информации через уязвимости веб-приложения;
- выгрузка информации напрямую из базы данных.

К ним могут быть применены следующие атаки:

- атака на БД (или на сервис где находится БД);
- атака на HTTP-сервер (посредством уязвимостей передоваемых HTTP-заголовков);
- воздействие на сервер через клиентскую часть (основной тип атак, использующий уязвимости взаимодействия клиента и сервера).

Теперь нужно обезопасить уязвимые места в нашей системе. Мы уже внедрили некоторые средства защиты – подключили модуль hemlet, для защиты нашего http-сервера.

Теперь нужно обезопасить уязвимые места в нашей системе. Мы уже внедрили некоторые средства защиты – подключили модуль hemlet, для защиты нашего http-сервера.

Во-первых, обезопасим конфиденциальные данных пользователей (пароли), которые хранятся на сервере.

Во-вторых, попробуем обезопасить взаимодействие клиента и сервера – так как злоумышленник будет действовать со стороны клиента - он в первую очередь попытается внедрить определенный код на стороне клиента, так же он может подменить данные клиента для доступа к личной информации пользователя. Поэтому мы должны валидировать (проверять) все входящие данные.

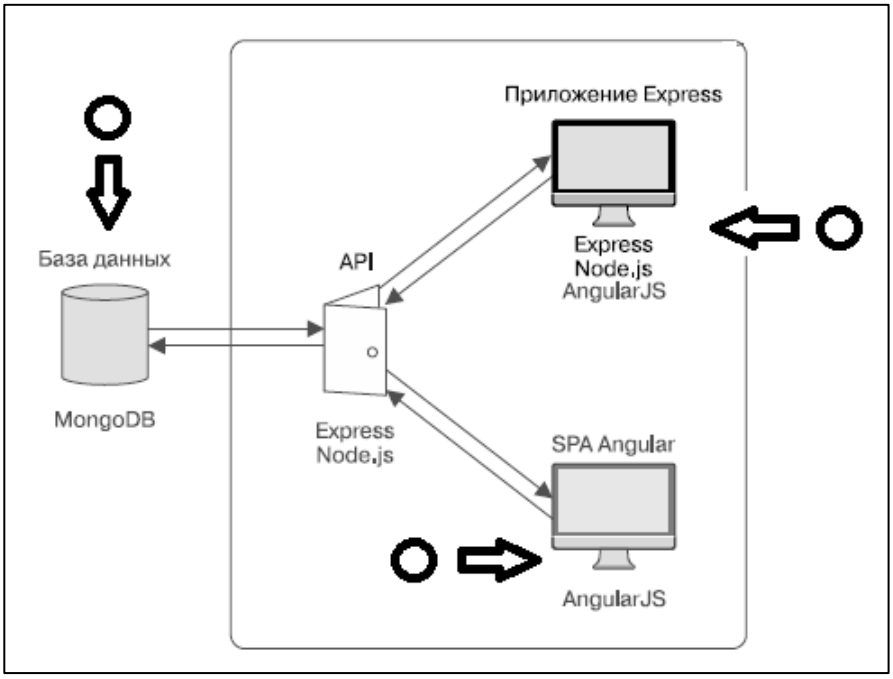

Рисунок 3.6.2 – Возможные уязвимые места системы

Так же, нужно учесть, что злоумышленник или случайный пользователь может вызвать ошибку на сервере (при неправильном вводе данных, получения данных другого пользователя, загрузки огромного файла и т.д.), что может привести к неправильной работе сервера или его полного отключения. Для того чтобы избежать это – сделаем обработчики ошибок для всех исполняемых функций на сервере. Так же ограничим доступ к некоторым ресурсам сайта через jwt-токен и функцию sign.

Одним из реализуемых элементов безопасности будет валидация данных на сервере(их проверка).

Валидация на стороне сервера должна осуществляться независимо от того, включена валидация на стороне клиента или нет. Пользователь может отключить JavaScript или совершить какие-нибудь другие непредвиденные действия, чтобы обойти валидацию на стороне клиента, и сервер останется последней линией защиты наших данных от некорректного ввода.

Наш сервер может получить данные при аутентификации или при отправке файла. Для первого случая будем использовать модуль - express-validator. Для этого в файле маршрутов подключим этот модуль и модернизируем наши маршруты и контроллеры специальными методами: isLength({ min: 3 }) –минимальная длина поля 3, escape() – запретить использование запрещенных символов (/?&">), которые могут реализовать различные инъекции, trim()- запрет на пробелы в полях (рисунок 3.6.3).

```
const {check, validationResult} = require('express-validator/check');
router.post('/login', [
    check('username', 'Not valid username').isLength({ min: 3 }).trim().escape(),
    check('password', 'Not valid password').trim().escape().isLength({min: 5}),
  ], ctrlProfile.login);
```

Рисунок 3.6.3 – Валидация входящих запросов

Далее в случае ошибки будет отправлено указанное сообщение с описание ошибки (рисунок 3.6.4).

Чтобы обеспечить фильтрацию файлов на сервере нет необходимости в подключении дополнительных библиотек, все что нужно – это данные req запросов пользователей (рисунок 3.6.5).

Укажем добустимый размер фотографий – 2 Мб, а так же допустимые форматы – ['image/jpg', 'image/jpeg', 'image/png']. В случае ошибки загруженный файл удаляется и пользователю приходит ошибка.

На этом этапе наше приложение готово, теперь созданный нами сайт можно выкладывать в сеть для дальнейшего тестирования. А в следущей главе будет произведен анализ эффективности созданного приложения при помощи выбранных технологий.

Для того чтобы избежать случайных или намеренных отключений сервера необходимо устанавливать слушатели для ошибок, которые в случае

критических ситуаций вместо того чтобы отключать сервер, будут просто выводить в консоль ошибку (рисунок 3.6.6).

```
const errors = validationResult(req);
   if (!errors.isEmpty()) {
     console.log(errors.array());
     return res.json({
       success : false,
       error : errors.array()
     });
```

Рисунок 3.6.4 – Валидация входящих запросов

Подобные блоки желательно добавлять при каждом выполнении функциональных блоков в приложении.

Такой подход для организации кода node.js хоть добовляет количество кода, несмотря на это закрывает многие критические уязвимости которыми могут воспользоваться злоумышленники.

```
var maxsize = 2 * 1024 * 1024;
  var supportMimeTypes = ['image/jpg', 'image/jpeg', 'image/png'];
       upload (req, res, function (err) {
       console.log(req.file);
       if (req.file.size > maxsize) {
         fs.unlinkSync(req.file.path);
         return res.json({success: false, msg: 'File is so more than 2 Mb'});
       }
       if(supportMimeTypes.indexOf(req.file.mimetype) == -1) {
         fs.unlinkSync(req.file.path);
         return res.json({success: false, msg: 'Unsupported mimetype'});
        }
```

Рисунок 3.6.5 – Валидация входящих запросов

```
newTree.save(function(err) {
  if (err) {
    console.log(err);
    }
```

Рисунок 3.6.6 – Обработка ошибок

У нас есть определенные функции, которые предоставляют пользователям их личные данные при поступлении определенных роутов. Необходимо их обезопасить. Для этого устанавливаем обязательную аутентификацию для пользователей с проверкой доступа через применение встроенной функции jwt (рисунок 3.6.7).

```
var auth = jwt({
  secret: 'MY_SECRET'
});
router.get('/profile', auth, ctrlProfile.profile);
```

Рисунок 3.6.7 – Ограничение доступа к ресурсам

При этом укажем секретный пароль «MY_SECRET». Теперь если пользователь будет обращаться к этому маршруту – будет произведена проверка его подписи, которая внедряется в созданный токен и является индивидуальной для каждого пользователя.

В нашем приложении будет выполняться одностороннее шифрование пароля. Одностороннее шифрование делает невозможной расшифровку пароля, сохраняя возможность с легкостью проверить его правильность. Когда пользователь предпринимает попытку входа, приложение может зашифровать предоставленный пароль и проверить, соответствует ли он сохраненному значению.

Однако простого шифрования недостаточно. Если несколько пользователей выбрали себе пароль «пароль» (да, такое случается!), то зашифрованный вариант для всех них будет одним и тем же. Любой хакер, получивший доступ к базе данных, сможет увидеть закономерность и вычислить потенциально слабые пароли.

Здесь нам пригодится понятие соли. Соль – это случайная строка, генерируемая приложением для каждого пользователя и объединяемая с паролем перед шифрованием. Получающееся в итоге зашифрованное значение называется хешем (рисунок 3.6.8).

Функция создания соли и хеша с помощью библиотеки crypto.

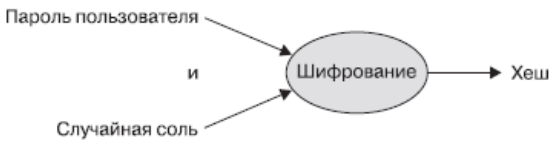

Рисунок 3.6.8 – Технология создания хешей

В нашем приложении будет выполняться одностороннее шифрование пароля. Обычно как соль, так и хеш хранятся в базе данных отдельно, а не в виде общего поля «пароль». Любой хакер, получивший доступ к базе данных, сможет увидеть закономерность и вычислить потенциально слабые пароли.

При таком подходе все хеши будут уникальны, а пароли – хорошо защищены (рисунок 3.6.11).

После чего добавим эти методы в контроллеры (рисунок 3.6.10).

```
UserSchema.methods.setPassword = function(password){
  this.salt = crypto.randomBytes(16).toString('hex');
  this.hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64, 'sha512').toString('hex');
};
Функция проверки хеша при обращении к серверу:
UserSchema.methods.validPassword = function(password) {
  var hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64, 'sha512').toString('hex');
  return this.hash === hash;
};
```

Рисунок 3.6.9 – Технология создания хешей

```
//При регистрации
newUser.setPassword(req.body.password);
//При входе
if (!user.validPassword(password)) {
  return done(null, false, {
    message: 'Password is wrong'
  });
}
```

Рисунок 3.6.10 – Проверка паролей через хеш

```
_id: ObjectId("5cd96ccbd5e838570411f922")
username: "salih"
name: "salih"
surname: "mamashev"
salt: "4405eb9478fe181723442468482359 5b"
hash: "4592a260ccdfdda18acf855ed06d24f186fd9a2d203ca314bab448b6defb529e032479..."
__v: 0
```

Рисунок 3.6.11 – Контроллер для получения списка сообществ

3.7 Преимущества выбранного стека технологий

Стек MEAN объединяет несколько лучших в своем классе современных веб-технологий в очень мощный и гибкий стек. Одна из лучших его черт: он не просто использует JavaScript в браузере – он использует JavaScript повсюду. С помощью стека MEAN можно программировать как клиентскую, так и серверную часть на одном и том же языке [22].

Основная технология, которая делает это возможным, – Node.js, внедряющая JavaScript в прикладную часть. Все это сильно упрощает взаимодействие фронтенд и бэкенд разработчиков, а вместе с тем и увеличивает скорость разработки [23].

Так же в таком приложении данные будут храниться, и передаваться в одном виде – JSON формате (рисунок 4.1.1), т.е. нет необходимости писать сложные SQL запросы для базы данных (рисунок 4.1.2) и затем их преобразовывать в JSON формат для передачи или обработки [24].

Серверная часть написана на плотформе Node.js. Одна из главных причин эффективности платформы Node.js такова: для нее программируют на знакомом большинству программистов языке – JavaScript. До сегодняшнего дня, если вы хотели быть , вам нужно было в совершенстве знать по крайней мере два языка программирования: JavaScript для клиентской части и что-то еще, например PHP или Ruby, – для серверной.

Рисунок 4.1.1 – Формат хранения данных

Еще одна причина эффективности платформы Node.js заключается в том, что при правильном программировании она работает очень быстро и чрезвычайно эффективно расходует системные ресурсы, что позволяет приложениям Node.js обслуживать больше пользователей при меньших системных ресурсах, чем большинству других широко распространенных серверных технологий. Так что владельцам компаний тоже нравится идея использования Node.js, ведь она может снизить текущие расходы даже при широкомасштабном применении.

Это происходит за счет того, что Node.js использует мало системных ресурсов в силу своей однопоточности, тогда как традиционные веб-серверы многопоточны. Вместо того чтобы предоставлять каждому посетителю отдельный поток и отдельный набор ресурсов, он всех посетителей соединяет с одним и тем же потоком. Посетитель и поток взаимодействуют только при необходимости – когда посетитель что-либо запрашивает или поток отвечает на запрос.

Если проводить аналогию с банковскими служащими, в данном случае будет только один клерк, работающий со всеми посетителями. Но вместо того, чтобы отходить и выполнять все запросы от начала до конца, он поручает все требующие много времени задачи персоналу резервного офиса и занимается следующим запросом.

При однопоточном подходе, показанном на рис. 4.1.2, Салли и Саймон подают свои запросы одному и тому же сотруднику банка. Но вместо того, чтобы целиком работать с одним из них, клерк принимает первый запрос и передает его лицу, которое лучше всего может его обработать, прежде чем принять следующий запрос и поступить аналогичным образом. Когда клерку говорят, что задание выполнено, он передает результат непосредственно запросившему его посетителю.

Несмотря на то, что банковский клерк всего один, никто из посетителей не догадывается о наличии других и ни на кого из них не влияют запросы другого. Такой подход означает, что банку не требуется большое количество постоянно работающих клерков. Конечно, эта модель масштабируется не безгранично, но она более эффективна, чем многопоточная. Вы можете сделать больше, расходуя меньше ресурсов.

Данный подход оказалось возможно реализовать в .js благодаря асинхронным возможностям JavaScript. Для примера можно сравнить скорость обработки пользователей в Node и Java [25] (рисунок 4.1.3).

Вы можете увидеть, что Node.js приложение имеет:

- в двое больше запросов в секунду в сравнении с Java приложением. Это ещё более интересно потому, что для получения наших внутренних результатов производительности было использовано одно ядро для Node.js по сравнению с пятью ядрами для Java. Мы ожидаем дальнейшего увеличения этой разницы;
- 35% снижение среднего времени ответа для той же страницы. Это привело к тому, что страницы стали обслуживаться на 200 миллисекунд быстрее – что пользователи точно заметят.

Как уже отмечалось, Node.js – это платформа, а не сервер. Это дает вам возможность проявить при настройке сервера свои творческие способности и заставить его выполнять то, что другие веб-серверы делать не способны. Но это же делает более трудным создание и запуск простого сайта.

Express делает эти сложности несущественными благодаря настройке веб-сервера для прослушивания входящих запросов и возврата соответствующих ответов.

Например, если для запуска сервера при использовании нам нужно скачивать специальные программные средства по типу XAMPP, и уже там настраивать и запускать http-сервер, в Node.js достаточно написать небольшой скрипт, платформа создаст сервер и запустит его на определенный порт сама (рисунок 4.1.5).

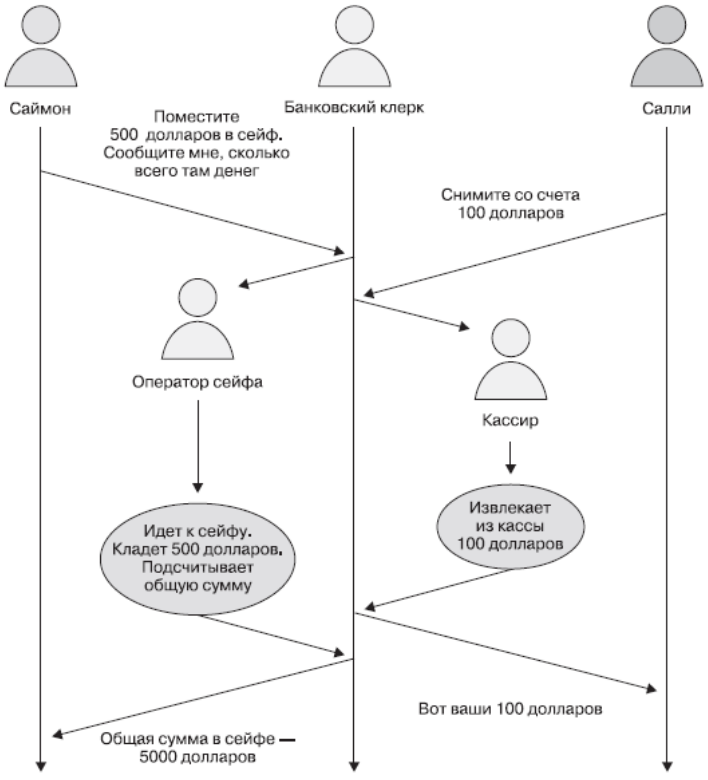

Рисунок 4.1.2 – Пример работы Node.js

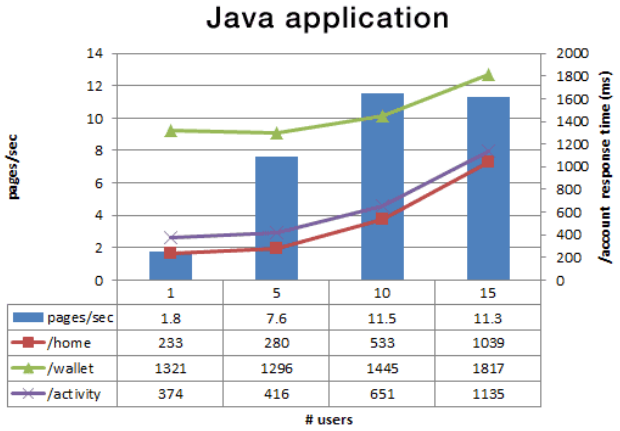

Рисунок 4.1.3 – Производительность приложения на Java

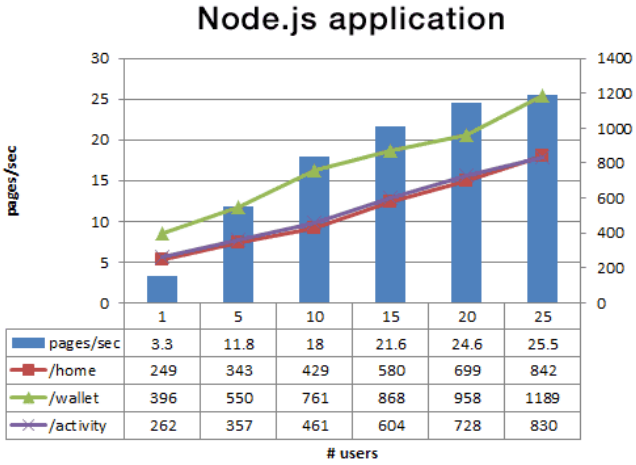

Рисунок 4.1.4 – Производительность приложения на Node.js

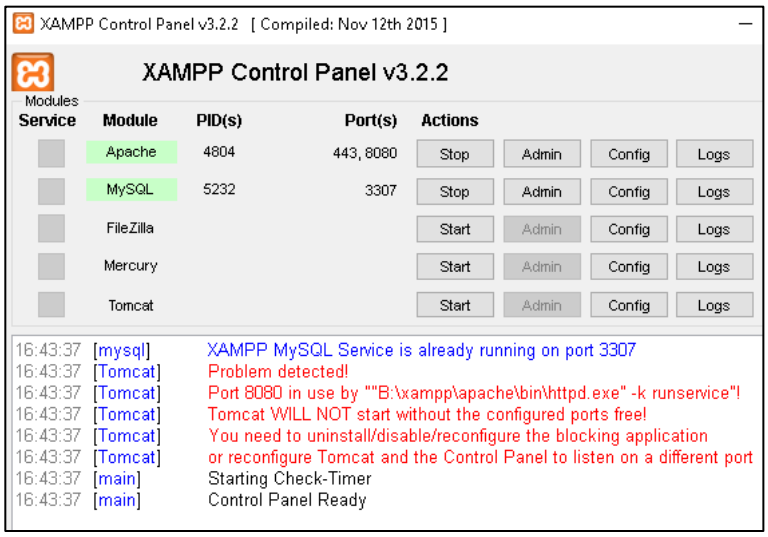

Рисунок 4.1.5 – Необходимость установки сервера при использовании LAMP

```
var express = require('express');
var http = require('http');
var server = http.createServer(app);
server.listen(conf.port, function () {
  console.log('Listining port 1337...');
});
```

Рисунок 4.1.6 – Компактный сервер при использовании Express

Так же это дает простоту при подключении различных модулей на сервер, например модулей, которые обеспечивают безопасность, путем прослушивания всех входящих запросов (рисунок 4.1.7).

```
const xssFilter = require('x-xss-protection')
app.use(xssFilter());
```

Рисунок 4.1.7 – Удобное внедрение средств защиты

Дополнительно он задает структуру каталогов. Один из этих каталогов настроен для обработки статистических файлов неблокирующим образом. Вы можете сконфигурировать эту возможность самостоятельно в .js, но Express делает это за вас. В нашем проекте мы подключили папку статик для хранения статистически загруженных файлов с помощью одной команды (рисунок 4.1.8).

```
app.use('/static', express.static('./static'));
```

Рисунок 4.1.8 – Большой диапазон настройки веб-сервера

Замечательная особенность Express – по-настоящему простой интерфейс для маршрутизации входящего URL к соответствующему фрагменту кода. Неважно, будет ли этот код выдавать статическую веб-страницу, выполнять чтение из БД или записывать в нее, – интерфейс прост и единообразен (рисунок 4.1.9).

```
app.get('*',function (req, res, next) {
  res.sendFile(path.join(__dirname,'dist/diplom/index.html'));
});
```

Рисунок 4.1.9 – Контроллер для получения списка сообществ

Здесь Express фактически уменьшил сложность выполнения всего этого в .js, ускоряя тем самым написание кода и облегчая его поддержку.

Будучи надстройкой над Node.js, Express предоставляет надежную отправную точку для создания веб-приложений. Он снимает множество сложностей и повторяющихся задач.

На клиентской части использовался фреймворк Angular.

Angular был специально спроектирован с расчетом на функциональность, называемую одностраничными приложениями (single-pageapplication (SPA)). Фактически SPA – это приложение, выполняющее все внутри браузера и никогда не производящее полной перезагрузки страницы [26].

Такой подход способен существенно снизить количество необходимых вам ресурсов сервера, причем фактически производится перераспределение требуемых приложению вычислительных мощностей. Браузер каждого пользователя выполняет тяжелую работу, а ваш сервер, по сути, просто выдает статические файлы и данные по запросу.

Подобный подход также улучшает механизм взаимодействия с пользователем. Как только приложение загружено, количество обращений к серверу уменьшается, что снижает вероятность задержек.

Так же Angular, как и большинство клиентских фреймворков облегчает создание и организацию кода. Это означает, что в итоге получается меньше кода за малое количество времени разработки. Это происходит за счет использования таких встроенных функций как ng-for, ng-if, ng-show и т.д. Для примера можно сравнить код, написанный на «чистом» JS(слева) и код, с использованием Angular(справа) (рисунок 4.1.10).

К недостаткам использования фреймворка можно отнести более медленную загрузку первой страницы, чем серверные приложения на основе шаблонизаторов. Это происходит потому, что при первой загрузке приходится загружать фреймворк и код приложения, прежде чем визуализировать в браузере нужное представление в виде HTML [27].

Такой подход способен существенно снизить количество необходимых вам ресурсов сервера, причем фактически производится перераспределение требуемых приложению вычислительных мощностей.

Браузер каждого пользователя выполняет тяжелую работу, а ваш сервер, по сути, просто выдает статические файлы и данные по запросу.

На момент написания дипломной работы через npm доступно более 46 000 пакетов, которые представляют разработчикам глубокие знания и огромный, которые возможно сделать частью приложения. Для получения доступа, к которым не надо заходить на сайт разработчиков, скачивать и распаковывать их, а достаточно написать название необходимого модуля в командной строке – все сохранится и инициализируется автоматически благодаря файлу конфигурирования – package.json (рисунок 4.1.11).

При чем, npm автоматически определит, для какой платформы был скачан пакет.

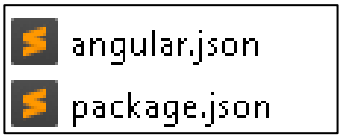

Рисунок 4.1.11 – Единый пакетный менеджер

```
$.ajax({
  method:"POST",
  url:'/getdoctors',
  cache: false,
  success:function (datas) {
    if(datas){
      for(let i = 0; i<datas.length; i++)
      {
        var div1 = document.createElement('div');
        var div2 = document.createElement('div');
        var div3 = document.createElement('div');
        var img = document.createElement('img');
        var p_name = document.createElement('p');
        var p_work = document.createElement('p');
        var p_disc = document.createElement('p');
        var p_recen = document.createElement('p');
        var button = document.createElement('button');
        $(div1).addClass("div_for_vrach");
        $(div2).addClass("div_for_img");
        $(div3).addClass("div_for_discr");
        $(p_name).addClass("text_for_name");
        $(p_work).addClass("text_for_work");
        $(p_disc).addClass("text_for_disc");
        $(p_recen).addClass("text_for_disc1");
        $(button).addClass("button_zapix");
        $(img).attr('src', datas[i].img);
        $(img).css('width','330px').css('height','380px');
        $(p_name).text(datas[i].name+' '+datas[i].surname+'
        $(p_work).text("Должность: "+datas[i].position);
        $(p_disc).text("Описание: "+datas[i].description);
        $(p_recen).text("Рецензия: "+datas[i].recenzia);
        $(button).text("Записаться на прием");
        div2.appendChild(img);
        div3.appendChild(p_name);
        div3.appendChild(p_work);
        div3.appendChild(p_disc);
        var vrachi = document.getElementById('vrachi');
        vrachi.appendChild(div1);
      }
    }else{
      console.log("error");
    }
  }
});
```

```
getPosts(){
  this.post.getPosts({id: this.id_branch}).subscribe(posts => {
    console.log(posts);
    this.posts = posts;
    }, (err) => {
      console.error(err);
    });
}
```

```
<div class="informationblockopen">
  <div><img [src]="branch.img"  style="width: 184px;height: auto;"><br
  </div>
  <div class=textinformationblok1>
    <p>{{branch?.text}}</p>
  </div>
</div>
<div class="dopolnitelni1">
  <p>Дополнительно:</p>
  <p>{{branch?.text}}</p>
</div>
</div>
```

Рисунок 4.1.10 – Удобство при написании кода на Angular

3.8 Анализ безопасности веб-приложения

Чтобы проверить надежность системы, нужно протестировать ее на наличие критических уязвимостей. Для сужения списка уязвимостей рассмотрим статистику наиболее известных уязвимостей за предыдущий год, выберем те, которые могут быть применены к нашей системе.

После двухгодичного курса на снижение доли веб-приложений, содержащих уязвимости высокого уровня риска, она вновь выросла и достигла 67%. Наиболее распространены уязвимости, связанные с недостаточной авторизацией, возможностью загрузки или чтения произвольных файлов, а также с возможностью внедрения SQL-кода.

В этом году специалисты Positive Technologie находили около 70 различных видов недостатков в веб-приложениях. Как всегда, высока доля веб-приложений с уязвимостями «Межсайтовое выполнение сценариев» (Cross-Site Scripting, XSS), неправильная конфигурация, использование инъекций, ошибки или недочеты в системе авторизации (рисунок 4.2.1).

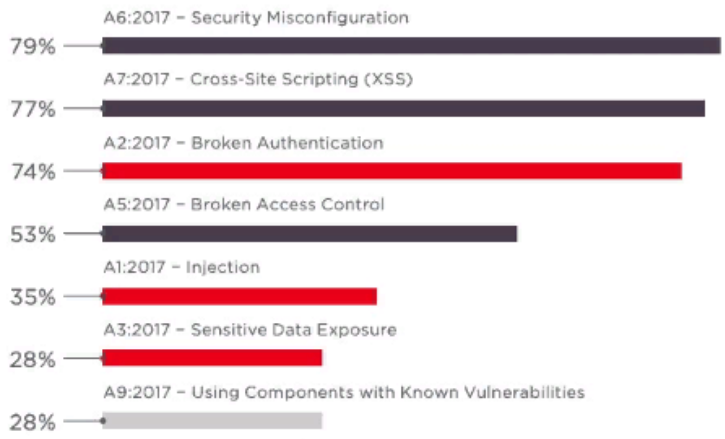

Рисунок 4.2.1 – Статистические данные веб-уязвимостей

Если соотнести статистические данные, полученные от Positive Technologie и нашу систему, то можновыделить следущие уязвисости, которыми злоумышленники попробуют воспользоваться в первую очередь:

- ошибки конфигурации или настройки сервера;
- NoSQL-инъекции;
- межсайтовый скриптинг (XSS);
- незашифрованная сессия авторизации;
- уязвимости механизмов аутентификации(Broken Authentication);
- инъекция удаленных/локальных файлов;
- атаки на http-сервер.

Межсайтовые сценарии (XSS) позволяют злоумышленникам для внедрения вредоносного кода на веб-страницы. Такой код может затем, например, украсть данные пользователя (в в частности, данные для входа) или выполнять действия по олицетворению пользователя. Это один из самых распространенные атаки в интернете.

Чтобы заблокировать атаки XSS, необходимо запретить ввод вредоносного кода в DOM (модель объекта документа). Например, злоумышленники могут обмануть вас, вставив <script> тег в DOM, тем самым запустив произвольный код на вашем сайте.

Чтобы систематически блокировать ошибки XSS, Angular по умолчанию обрабатывает все значения как ненадежные. Когда значение вставляется в DOM из шаблона с помощью свойства, атрибута, стиля, привязки класса или интерполяции, Angular очищает и экранирует ненадежные значения.

Но так как мы используем сервер на Node, существует возможность внедрить XSS инъекцию в серверный код. Для этого достаточно внедрить тег <script> в форму отправки данных. Давайте так и сделаем (см. рис 4.2.2).

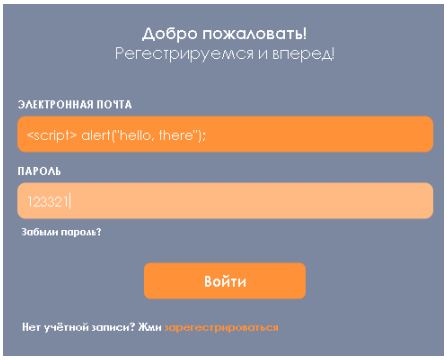

Рисунок 4.2.2 – Проверка на XSS

Теперь посмотрим как на такую уязвимость реагирует наша система (рисунок 4.2.3).

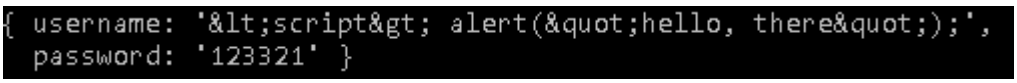

Рисунок 4.2.3 – Защита от XSS

Как мы видим XSS не удался, все благодаря внедренных методов валидации входящих данных (рисунок 4.2.4).

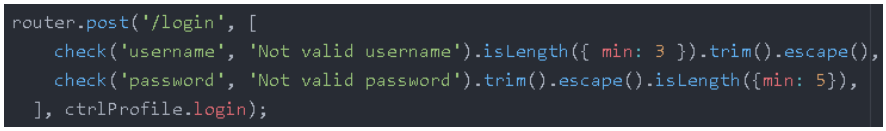

Рисунок 4.2.4 – Защита от XSS

Инъекция – это один из самых доступных способов взлома сайта. Суть инъекций – внедрение в данные (передаваемые через GET, POST запросы или значения Cookie) произвольного SQL или JavaScript кода. Если сайт уязвим и выполняет такие инъекции, то по сути есть возможность серьезно навредить серверу [28].

Однако стоит учесть, что в нашем приложении используется NoSQL база – MongoDB, поэтому злоумышленник не может провести на них атаки типа SQL-injection.

Но, если в систему невозможно внедрить SQL-код, это еще не значит, что она безопасна.

NoSQL закрывает одну потенциальную уязвимость, при этом открывая с десяток других, которые позволяют совершать инъекции с использованием регулярных выражениях MongoDB [29].

MongoDB, как и многие другие NoSQL-СУБД, позволяет осуществлять поиск с помощью регулярных выражений. Это очень мощное, но в то же время опасное средство, которое может нанести существенный вред при неправильном использовании.

Аутентификация пользователя происходит посредством запроса, который указывает регулярное выражение в качестве пароля. При этом если переменные никак не фильтруется, это открывает полную свободу действий для злоумышленника.

Здесь можно указать имя пользователя root и регулярное выражение вместо пароля, например {$ne:1}. В результате MongoDB выполнит следующий запрос: «db.users.findOne({login: 'root'})», и будет совершен успешный вход на сайт под логином root (этот прием напоминает классическую SQL-инъекцию «1' or 1=1 --»). Давайте попробуем это реализовать (рисунок 4.2.5).

Однако вместо успешного входа, приходит ошибка о том, что такого пользователя не существует (рисунок 4.2.6).

Это происходит из-за того, что для защиты от подобных типов атак в нашем приложении использовался встроенный механизм в mongoose –

sanitize, который позволяет избежать большинство уязвимотсей с использованием регулярных выражений (рисунок 4.2.7).

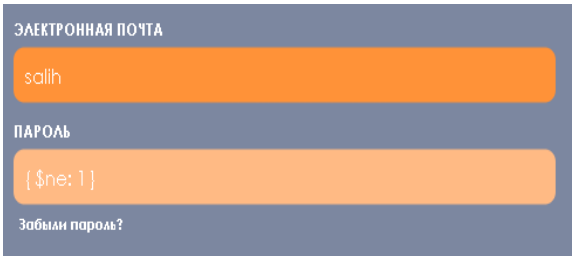

Рисунок 4.2.5 – Реализация NoSQL инъекций

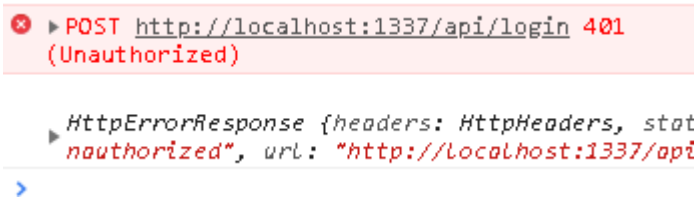

Рисунок 4.2.6 – Защита от NoSQL инъекций

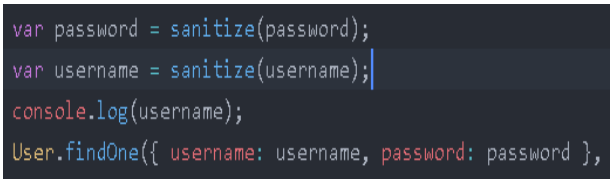

Рисунок 4.2.7 – Защита от NoSQL инъекций

Это происходит из-за того, что для защиты от подобных типов атак в нашем приложении использовался встроенный механизм в mongoose – sanitize, который позволяет избежать использования большинство уязвимотсей с использованием регулярных выражений. Для более мощной проверки будет использовать специальный сканнер уязвимостей для инъекций – sqlmap. Для этого вводим ссылку на формы регистрации и входа в необходимое поле, после чего будет запущена проверка (рисунок 4.2.8).

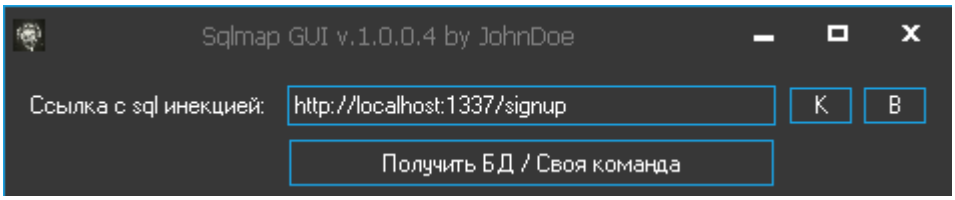

Рисунок 4.2.8 – Использование sqlmap для поиска уязвимостей перед инъекциями

Который так же показывает, что уязвимости не были найдены (рисунок 4.2.9).

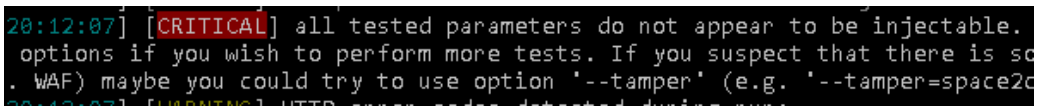

Рисунок 4.2.9 – Сообщение о том, что уязвимостей нет

Все запросы от браузера к серверу ведутся по открытым каналам. Существует возможность перехвата сессии пользователя злоумышленником, с последующей ее расшифровкой или подменой. В нашем приложении для хранении сессии используется технология JWT. Данный объект хранит в себе значения 3 полей – заголовок, содержимое и подпись.

Допустим, злоумышленник перехватил данный токен. Для того чтобы расшифровать он применит метод atob() – для декодирования данных (см. рис 4.2.10).

```
payload = window.atob(payload);
```

Рисунок 4.2.10 – Декодирование JWT-токена

Теперь злоумышленник получил содержимое пользовательского токена, в текстовом формате он выглядит так (рисунок 4.2.11).

```
exp: 1558785442
iat: 1558180642
username: "salih"
```

Рисунок 4.2.11 – Декодирование JWT-токена

Теперь злоумышленник может подменить никнейм пользователя и обратится к серверу. Однако у него ничего не выйдет, поскольку причина, почему JWT используются – это проверка, что отправленные данные были действительно отправлены авторизованным источником. Как было продемонстрировано выше, данные внутри JWT закодированы и подписаны, а это значит, что декодировать данные очень легко, но подделать подпись невозможно, поскольку для подписи применяется специальный пароль, который создается разработчиком и «вшит» в сервер (рисунок 4.2.12).

```
var auth = jwt({
  secret: 'MY_SECRET'
});
```

Рисунок 4.2.12 – Невозможность подделать JWT-токена

Таким образом, данную уязвимость реализовать в нашем приложении практически невозможно.

Загрузка пользователем файлов на веб-сайт, даже если это просто смена аватара, несёт в себе угрозу информационной безопасности. Загруженный

файл, который, на первый взгляд, выглядит безобидно, может содержать скрипт и при выполнении на сервере откроет злоумышленнику доступ к сайту. Для проверки наличия подобной уязвимости попробуем загрузить файл формата .bat, который в определенное время сканирует папку и отсылает ответ на удаленный сервер (рисунок 4.2.13).

Приходит ошибка о том, что произошла ошибка при загрузке файла, по причине – «запрещенный формат» (рисунок 4.2.14).

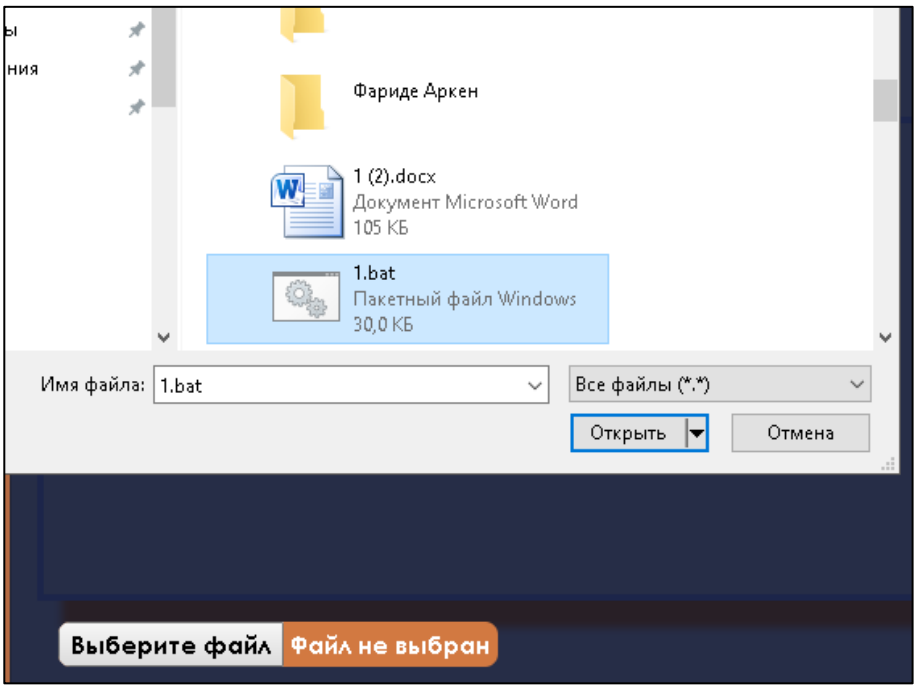

Рисунок 4.2.13 – Реализация инъекции удаленных файлов

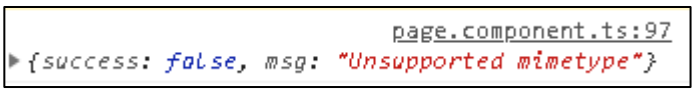

Рисунок 4.2.14 – Срабатывание защиты от инъекции удаленных файлов

Даже если установлено ограничение на тип (например, только изображения), есть возможность загрузить файлы огромного размера, чтобы засорить сервер. Попробуем загрузить файл размером 90 МБ (рисунок 4.2.15). Приходит ошибка о том, что размер загружаемого файла, больше допустимой нормы – 2 МБ (см. рис 4.2.16).

Для реализации защиты от данных уязвимости была применена валидация загружаемых на сервер данных, таким образом, что файлы определенного формата и размера не могут быть загруженными на сервер (см. рис 4.2.17).

Этот механизм защиты предотвращает большинство уязвимостей подобного типа.

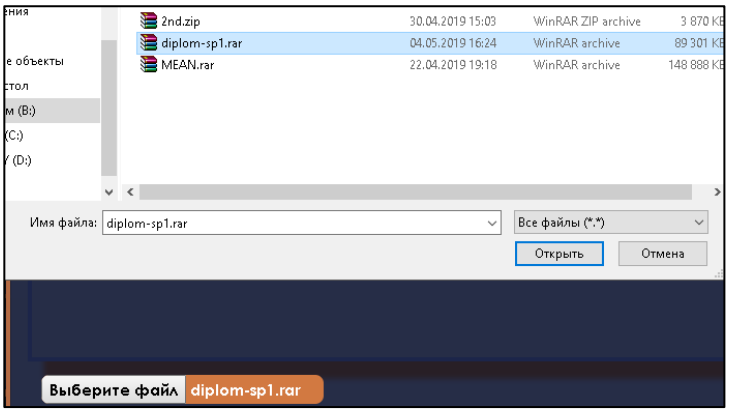

Рисунок 4.2.15 – Реализация инъекции удаленных файлов

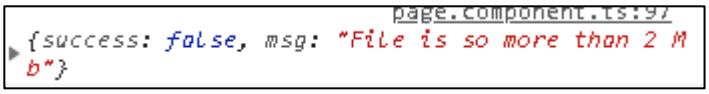

Рисунок 4.2.16 – Срабатывание защиты от инъекции удаленных файлов

```
var maxsize = 2 * 1024 * 1024;
var supportMimeTypes = ['image/jpg', 'image/jpeg', 'image/png'];
    upload (req, res, function (err) {
    console.log(req.file);
    console.log(req.body);
    if(req.file){
      if (req.file.size > maxsize) {
      fs.unlinkSync(req.file.path);
      return res.json({success: false, msg: 'File is so more than 2 Mb'});
    }
    if(supportMimeTypes.indexOf(req.file.mimetype) == -1) {
      fs.unlinkSync(req.file.path);
      return res.json({success: false, msg: 'Unsupported mimetype'});
     }
   }
```

Рисунок 4.2.17 – Реализация защиты от инъекции удаленных файлов

Так же будет рассмотрен вариант – при котором злоумышленник получил прямой доступ к базе данных (был получен пароль от хостинга или физический доступ к серверам базы данных) для получения паролей пользователя. После получения допуска, злоумышленник находит все данные по пользователю и обращается к ним (рисунок 4.2.18). Но как мы вы видим – вместо полей с паролем, есть 2 поля, которые хранят хеш пароля и его соль.

Рисунок 4.2.18 – Защита от прямого доступа к БД

Для избегания этой уязвимости в нашем приложении был применен подход проектирования приложения, при котором пароли хранятся в виде хэша с использованием алгоритмов одностороннего хэширования SHA. В этом случае для авторизации пользователей сравниваются хэшированные значения. Если злоумышленник взломает ресурс и получит хэшированные пароли, ущерб будет снижен за счёт того, что хэш имеет необратимое действие и получить из него исходные данные практически невозможно. Но хэши на популярные пароли легко перебираются по словарю, поэтому также применилась «соль», уникальная для каждого пароля, поэтому взлом большого количества паролей становится ещё медленнее и требует больших вычислительных затрат.

Так как наше приложение является в то же время веб-сервером, на него можно совершить различные виды атак с использованием протокола HTTP по типу клиджекинга, сниффинга и т.д. Поэтому существуют несколько относящихся к безопасности HTTP-заголовков, которые стоит установить:

- Strict-Transport-Security делает обязательным безопасные (HTTP через SSL/TLS) соединения с сервером;
- X-Frame-Options предоставляет защиту от кликджекинга;
- X-XSS-Protection включает фильтр межсайтового скриптинга (XSS), встроенный в большинство современных браузеров;
- X-Content-Type-Options предотвращает MIME-сниффинг ответа без декларированного Content-Type;
- Content-Security-Policy предотвращает широкий спектр атак, включая XSS и другие межсайтовые инъекции.

В нашем приложении все эти элементы реализованы благодаря использованию специальной технологии - Helmet (рисунок 4.2.19).

```
var helmet = require('helmet')
app.use(helmet())
```

Рисунок 4.2.19 – Защита от http-уязвимостей

Для проверки того, что данные механизмы были включены, можно зайти на сайт, открыть вкладку NetWork и посмотреть использованные заголовки (см. рис 4.2.20).

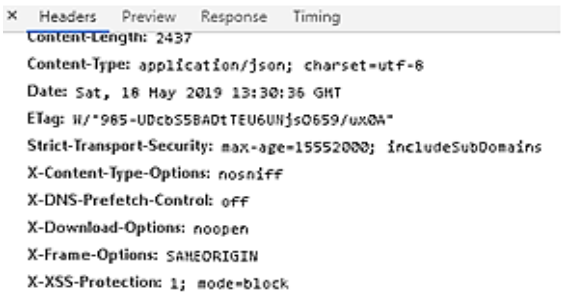

Рисунок 4.2.20 – Защита от http-уязвимостей

Для проверки эффективности данных методов, запустим сканирования нашего приложения через специальную утилиту – nikto. Это специальная программа для проверки сайта на уязвимости, которая также имеется в дистрибутиве Kali Linux. Обладает богатым функционалом, при всей своей простоте она дает подробный отчет о системе.

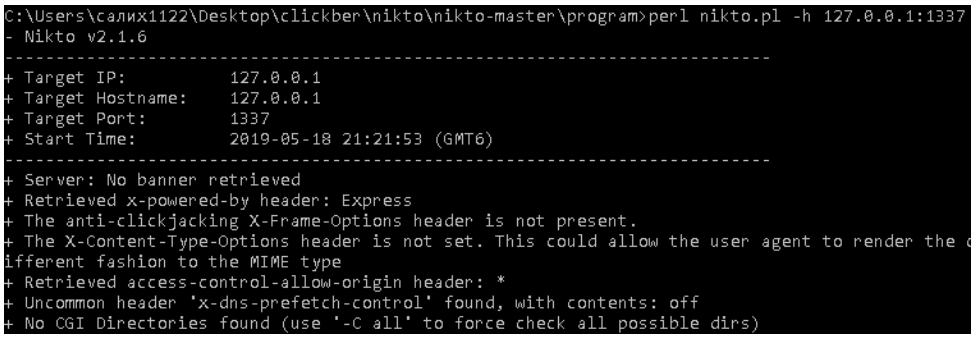

Рисунок 4.2.21 – Проверка на http-уязвимости

Как видим из вывода программы (рисунок 4.2.21) – все примененные средства отображены, а значит и будут выполнять свои функции в случае реальной атаки.

4 Безопасность жизнедеятельности

Содержание и цели БЖД:

– изучение существующих факторов окружающей среды, оказывающих вредоносное влияние на человека;

– ослабление данных факторов до безопасных значений или их полная ликвидация.

На рабочем месте обязаны быть предусмотрены мероприятия по защите от влияния опасных факторов производства. Для обеспечения нормального трудового процесса они не должны превышать допустимые санитарно-технические нормы.

4.1 Исходные данные

Т а б л и ц а 4.1.1 – Исходные данные

Город	Алматы
Параметры помещения (Д*Ш*В), м	10 * 7 * 4
Данные по оборудованию	
Количество компьютеров, шт.	6
Мощность $P_{об}$, кВт/ч	0.95
КПД η	0.9
Данные по источнику света	
Мощность $N_{осв}$, Вт/м^2	73
Вид источника освещения	люминесцентные лампы
Количество сотрудников	3 (мужчины), 2 (женщины)
Окна	
Количество, шт.	2
Площадь 1 окна, м^2	3.4
Расположение	СВ
Вид жалюзи	вертикальные металлические
Расчетное время суток, ч.	11-13
Температура в помещении	
Летом, ^oC	24
Зимой, ^oC	18
Вид положения работы	сидячая

4.2 Расчёт тепловых нагрузок в помещении: внутренних и наружных

В помещениях различного назначения в основном действуют тепловые нагрузки, возникающие вне помещения (наружные); а также тепловые нагрузки, возникающие внутри зданий (внутренние). Данные нагрузки представлены следующими составляющими:

- прирост или потеря тепла в результате разницы температур внутри и снаружи здания через стены, потолки, полы, окна и двери;разность температур снаружи здания и внутри него летом является положительной, в результате чего имеет место приток тепла снаружи во внутрь помещения; и

наоборот – зимой эта разность отрицательна и направление потока тепла меняется;

- приток тепла от солнечного излучения через остекленные участки; эта нагрузка проявляется в форме ощущаемого тепла;

- теплопоступления от инфильтрации.

Тепловые нагрузки могут быть положительными. Подвод тепла и потери тепла в результате разности температур по формуле 5.1:

$$Q_{\text{огр}} = V_{\text{пом}} * X_o * \left(t_{Н\text{расч}} - t_{\text{Врасч}}\right) \quad (5.1)$$

где $V_{\text{пом.}}$ – объем помещения, м^3

$$V_{\text{пом}} = 10 * 7 * 4 = 283 \text{ м}^3;$$

X_o – удельная тепловая характеристика, Вт/м^{3o}С, X_o = 0.42;

$t_{Н\text{расч}}$ – температура наружного воздуха (параметр А). Для холодного периода - средняя температура самого холодного месяца – 13 часов, для теплого периода – средняя температура самого жаркого месяца – 13 часов.

$t_{\text{Врасч}}$ – внутренняя температура подбирается с учетом комфортных условий или технологических требований к производственным процессам.

Для теплого времени года:

$t_{Н\text{расч}} = 29{,}4\ ^0$С

$t_{\text{Врасч}} = 26\ ^0$С

$$Q_{\text{огр}} = 280 \cdot 0{,}42 \cdot 3{,}4 = 399{,}84 \approx 400 \text{ Вт}$$

Для холодного времени года

$t_{Н\text{расч}} = -9\ ^0$С

$t_{\text{Врасч}} = 19\ ^0$С

$$Q_{\text{огр}} = 280 * 0.42 * |-28| = 3292{,}84 \approx 3292 \text{ Вт}$$

Избыточное тепло солнечного излучения, в зависимости от типа стекла, поглощается окружающей средой помещения почти до 90%, остальное отражается. Максимальная тепловая нагрузка достигается при максимальном уровне излучения, который имеет прямые и рассеянные компоненты. Интенсивность радиации зависит от ширины местности, времени года и времени суток.

Подвод тепла от солнечного излучения через остекление определяется по формуле 5.2:

$$Q_P = (q^I F_O^I + q^{II} F_O^{II}) * \beta_{\text{с.з}} \quad (5.2)$$

где q^I, q^{II} - тепловые потоки от прямой и рассеянной солнечной радиации, Вт/м^2;

F_O^I, F_O^{II} - площади светового проема, облучаемые и необлучаемые прямой солнечной радиации, м^3;

$\beta_{с.з}$ - коэффициент тепло-пропускания. $\beta_{с.з} = 0.15$;

Без внешних солнцезащитных козырьков, ребер и т. Д. На время солнечного воздействия остекления, когда его лучи проникают через окно в помещение:

$$Q_P = q^I F_O * \beta_{с.з} = \left(q_{вп} + q_{вр}\right) * K_1^C * K_2 * n * S_O \qquad (5.3)$$

$q_{вп}$, $q_{вр}$ - тепловые потоки от прямой рассеянной радиации, Вт/м2. Для широты в 44оСШ до полудня в 11-13 ч. При расположении С: $q_{вп}$, $q_{вр}$

$q_{вп} = 214$ Вт/м2

$q_{вр} = 79$ Вт/м2

F_O – площадь светового проема (5.7) (n – число окон, S_O – площадь одного окна);

$$F_O = n * S_o \qquad (5.4)$$

$$F_O = 2 \cdot 3{,}4 = 6{,}8 \text{ м}^2$$

K_1 – коэффициент затемнения остекления переплетами (K^C_1 – для облученных проемов). По таблице 4.1.1:

$K^C_1 = 0.72$;

K_2 – коэффициент загрязнения остекления. По таблице 4.1.1:

$K_2 = 0.9$.

Тогда:

$$Q_P = (214 + 79) * 0.72 * 0.9 * 0.15 * 6{,}8 = 193{,}81 \text{ Вт}$$

Поступление солнечного излучения равно:

$$Q_P = 193{,}81 \text{ Вт}$$

Внутренние нагрузки в жилых, офисных или относящихся к сфере обслуживания помещениях слагаются в основном из тепла:

- выделяемого людьми;
- выделяемого лампами и осветительными, электробытовыми приборами;
- выделяемого компьютерами, печатающими устройствами фотокопировальными машинами пр.;

В производственных и технологических помещениях различного назначения дополнительными источниками тепловыделений могут быть: нагретое производственное оборудование, горячие материалы, в том числе

жидкости и различного рода полуфабрикаты, продукты сгорания и химических реакций.

Теплопоступления от людей зависит от интенсивности выполняемой работы и параметров окружающего воздуха. Тепло, выделяемое человеком, складывается из ощутимого (явного), то есть передаваемого в воздух помещения путем конвекции и лучеиспусканий, и скрытого тепла, затрачиваемого на испарение влаги с поверхности кожи и из легких.

Летом при 24 ^{0}С один мужчина выделяет явного тепла 61 Вт, а общего – 102 Вт. Женщина выделяет 85% от нормы тепловыделений взрослого мужчины. Тогда выделение явного тепла в помещении составит:

$$Q_{л}^{я} = 61 * 3 + 61 * 2 * 0{,}85 = 286{,}7 \text{ Вт}$$

А выделение общего тепла:

$$Q_{л}^{о} = 102 * 3 + 102 * 2 * 0{,}85 = 479{,}4 \text{ Вт}$$

Зимой при 20 ^{0} С один мужчина выделяет явного тепла 82 Вт, а общего – 103 Вт. Женщина выделяет 85% от нормы тепловыделений взрослого мужчины. Тогда выделение явного тепла в помещении составит:

$$Q_{л}^{я} = 82 * 3 + 82 * 2 * 0{,}85 = 385{,}4 \text{ Вт}$$

А выделение общего тепла:

$$Q_{л}^{о} = 103 * 3 + 103 * 2 * 0{,}85 = 484{,}1 \text{ Вт}$$

Теплопоступление от осветительных приборов, оргтехники и оборудования рассчитывается следующим образом. Теплопоступление от ламп определяется по формуле(5.5):

$$Q_{\text{осв}} = \eta * N_{\text{осв}} * F_{\text{пол}} \qquad (5.5)$$

где η – коэффициент перехода электрической энергии в тепловую (для люминесцентных ламп η=0.5-0.6);

$N_{осв}$ – установленная мощность ламп (N=60 Вт/м^2);

$F_{пол}$ – площадь пола:

$F_{пол}$=10·7=70

Тогда:

$$Q_{\text{осв}} = 0{,}5 \cdot 60 \cdot 70 = 2100 \text{ Вт}$$

Тепло, выделяемое производственным оборудованием, определяется по формуле (1.5):

$$Q_{\text{об}} = N_{\text{уст}} * K \quad (5.6)$$

$$Q_{\text{об}} = 1{,}8 * 103 * 6 * 0{,}9 = 9720 \text{ Вт}$$

Теплопритоки, возникающие за счет находящейся оргтехники, – это 30% мощности оборудования:

$$Q_{\text{орг}} = 1{,}8 * 103 * 6 * 0{,}3 = 2916 \text{ Вт}$$

4.3 Расчет количества воздуха, необходимого для подачи в помещение

На основании выполненных расчетов составим баланс теплопоступлений в помещении:

Лето:

$$Q_{\text{изб}} = 193{,}8 + 286{,}7 + 2100 + 9720 + 2916 + 400 = 15616 \text{ Вт}$$

Зима:

$$Q_{\text{изб}} = 193{,}8 + 286{,}7 + 2100 + 9720 + 2916 + 3292{,}8 = 18908 \text{ Вт}$$

Так как тепловой баланс для лета больше зимнего теплового баланса, то рассчитаем теплонапряженность воздуха по формуле:

$$Q_H = \frac{Q_{\text{изб}} * 860}{V_{\text{пом}}} \quad (5.7)$$

$$Q_H = \frac{18908 \cdot 860}{270} = 60{,}325 \frac{\text{ккал}}{\text{м3}}$$

При $Q_H > 20$ ккал/м^3, $\Delta t = 8$ °С

Определение количества воздуха, необходимое для поступления в помещение:

$$L = \frac{Q_{\text{изб}} * 860}{C * \Delta t * \gamma} \quad (5.8)$$

$$L = \frac{18908 \cdot 860}{0{,}24 \cdot 8 \cdot 1{,}206} = 702{,}25 \text{ м3/час}$$

где С=0,24 ккал/(кг°С) – теплоемкость воздуха,

γ=1,206 кг/м^3 – удельная масса приточного воздуха.

Определение кратности воздухообмена:

$$n = \frac{L}{V_{\text{пом}}} \tag{5.9}$$

$$N = \frac{5350}{270} = 2.6 \text{ час} - 1$$

4.4 Выбор кондиционера для помещения

Исходя из полученных данных, выберем кондиционер сплит-системы настенного типа LG UU18W ULDR0.

Таблица 4.4.1 – Основные технические характеристики настенного кондиционера серии LG UU18W ULDR0

Эл. питание В/Гн	**Произв. по холоду, кВт**	**Потр. эл мощн, кВт**	**Потребл ток, А**	**Произв. по теплу, кВт**	**Размер (внешн. блок) мм**	**Расход воздуха, м3 /ч**	**Размер (внутр. блок) мм**
240/1/50	11	3,50	6,9	12	L 950 H 870	4000	L 570 H 570

4.5 Схема расположения кондиционера в помещении

Во внешнем блоке находятся компрессор, конденсатор и вентилятор. Внешний блок можно установить на стене здания, на крыше или на чердаке, в подсобном помещении или на балконе, то есть в таком месте, где горячий конденсатор может продуваться атмосферным воздухом более низкой температуры.

Внутренний блок устанавливается непосредственно в кондиционируемом помещении и предназначен для охлаждения или нагревания воздуха, фильтрации его и создания необходимой подвижности воздуха в помещении. Внутренние блоки поддерживают заданную температуру, обеспечивают равномерное распределение воздуха в помещении и работают практически бесшумно (уровень шума 35-38 дБ).

Управление работой настенного кондиционера производится с дистанционного пульта, который позволяет задать режим работы кондиционера: обогрев, охлаждение, осушку, вентиляцию, ночной режим; задать требуемую температуру, которую должен поддерживать автоматически; выбрать режим работы вентилятора: настроить таймер, который включит или выключит кондиционер в заданное время; автоматически регулировать положение направляющих шторок и изменить таким образом направление воздушного потока.

Так как количества воздуха, необходимое для поступления в помещение равно 702,25 $м^3$/час, то будет использован один кондиционер.

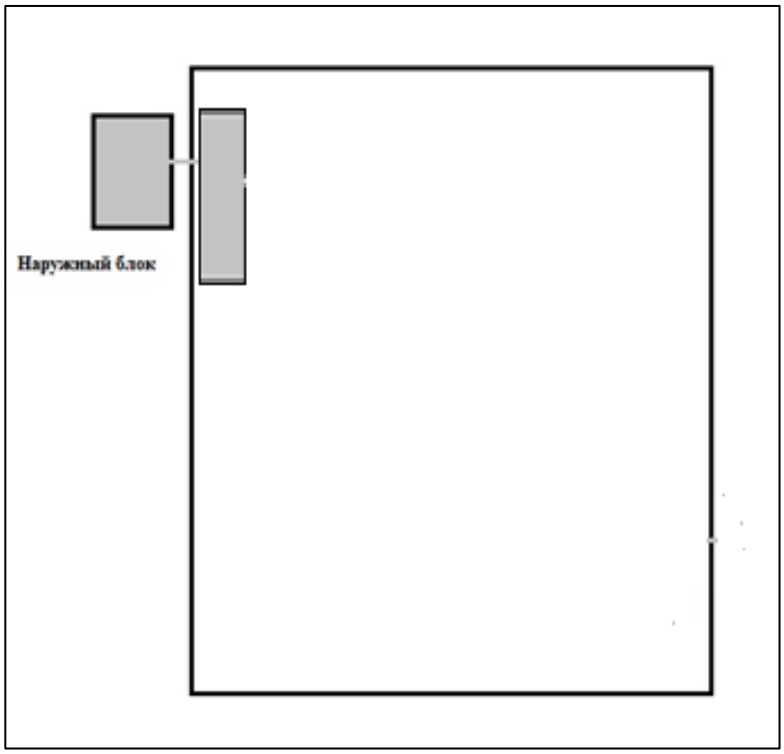

Рисунок 5.5.1 – Схема расположения кондиционеров в производственном помещении

Вывод

Задача кондиционирования воздуха состоит в выполнении вентиляции и отопления, а также в поддержании таких параметров воздушной среды, при которых каждый человек благодаря своей индивидуальной системе автоматической терморегуляции организма чувствовал бы себя комфортно, не замечая влияния этой среды.

В данной главе дипломного проекта были рассчитаны такие величины, как – внутренние и внешние нагрузки в помещении, необходимое количество воздуха для данных нагрузок, значение которой равно 702,25 $м^3$/час. Для данного количества был взят кондиционер типа LG UU18W ULDR0 с общей ценой 200110 тг. Далее была построена система с размещением этих кондиционеров.

Если пренебречь системой кондиционирования то у всей команды разработчиков состоящей из 5 человек в период разработки могли возникнуть такие негативные последствия, как – головные боли, быстрая утомляемость, увеличился бы риск возникновения различных инфекционных заболеваний и т.д. Поэтому при организации помещения для команды разработчиков, которая находится в нем в течении нескольких часов очень важно организовать систему вителяции.

5 Технико-экономическое обоснование

Цель данной дипломной работы заключается в разработке Web-приложения с использованием современных технологий с последующим анализом их эффективности и дальнейшим расположением в сети для общего пользования.

В данном дипломном проекте по разработке Web-приложения будет участвовать группа специалистов, которая включает в себя: руководитель проекта, дизайнер, бэкенд-программист, фронтенд-программист и тестировщик.

В обязанности руководителя проекта входит соблюдение и разработка рабочих графиков, их контроль и оптимизация. В обязанности дизайнера входит создание готового макета сайта. В обязанности бэкенд-программиста входит создание серверной части сайта.

Технико-экономическое обоснование содержит следующие пункты:

- определение сложности разработки программного обеспечения;
- расчет затрат на разработку ПО;
- определение ценности готового продукта;
- оценка результатов работы программного обеспечения.

5.1 Определение трудоемкости разработки ПО

Последовательность создания сайта и четкая проработка этапов – залог успеха всего проекта. Каждый этап создания сайта – очень важный шаг, за который ответственен каждый разработчик из команды.

Создание веб-сайта заключает в себе не только разработку дизайна и программинг, а и детальный анализ проекта, сотрудничество с заказчиком и поиск решений для достижения поставленных целей проекта.

В таблице 5.1.1 показана возможная поэтапная разработка сайта.

Таблица 5.1.1 - Этапы разработки ПО

Этапы разработки ПО	Вид работы	Трудоемкость, чел. час.
Этап 1	Предпроектные исследования	15
Этап 2	Разработка технического задания	15
Этап 3	Дизайн-концепция сайта (креативный дизайн)	30
Этап 4	Технический дизайн	30
Этап 5	Верстка html-страниц сайта на основе утвержденного дизайна типовых страниц.	25
Этап 6	Интеграция сайта с системой управления	35
Этап 7	Программирование, запуск проекта	25
Этап 8	Информационное наполнение сайта	25

Продолжение таблицы 5.1.1

Этапы разработки ПО	Вид работы	Трудоемкость, чел. час.
Этап 9	Тестирование сайта в Интернете	20
Этап 10	Сдача сайта в эксплуатацию	30
Итого: трудоемкость выполнения дипломного проекта		250

Продолжительность рабочего дня равна 8 часам. В результате для реализации программного обеспечения необходимо 30 рабочих дней.

5.2 Расчет затрат на разработку ПО

Для определения затрат, которые необходимые при создании веб-приложения необходимо учитывать следующие элементы имеющейся сметы:

- материальные затраты;
- затраты на оплату труда;
- социальный налог;
- амортизация основных фондов.

К материальным затратам относятся затраты на материалы, энергию и другие затраты необходимые для разработки ПО. Расчет материальных затрат происходит по форме, предоставленной в таблице 4.2.

Таблица 5.2.1 – Затраты на материальные ресурсы

Наименование материала	Ед. измерения	Количество	Цена за ед. в тенге	Сумма в тенге
Бумага	Упаковка	2	1 200	2 400
Чистящий набор	Штук	1	1200	1200
Тонер	Штук	2	2 000	4 500
Маркерная доска	Штук	1	6 000	6 000
Итого				14 100

Общую сумму, необходимую на материальные средства ($З_м$) можно рассчитать по следующей формуле:

$$З_м = \sum P_i * Ц_i, \quad (5.1)$$

где P_i - расход i-го вида материального ресурса, натуральные единицы;

$Ц_i$ - цена за единицу i-го вида материального ресурса, тг;

i - вид материального ресурса;

n - количество видов материальных ресурсов.

Для реализации программного обеспечения необходимы материалы на сумму 14 100 тенге.

Для разработки программного обеспечения будет использоваться ноутбук HP 250 G6, мощности ноутбука достаточно для выполнению поставленных задач.

Общую сумму, необходимую на материальные средства ($З_м$) можно рассчитать по следующей формуле:

$$З_м = \sum P_i * Ц_i, \quad (5.2)$$

где P_i - расход i-го вида материального ресурса, натуральные единицы;

$Ц_i$ - цена за единицу i-го вида материального ресурса, тг;

i - вид материального ресурса;

n - количество видов материальных ресурсов.

Расчет затрат на необходимое оборудование и программное обеспечение производится по форме, приведенной в таблице 4.3.

Таблица 5.3.1 – Расчет затрат на оборудование и ПО, необходимое для проекта

Наименование материала	Марка	Ед. измерения	Количество	Цена за ед. в тенге	Сумма в тенге
Ноутбук	HP 250 G6	Штук	1	190 000	190 000
Принтер	Samsung SL-M2020 A4	Штук	1	38 670	38 670
Модем	TP-Link TD-W8901N	Штук	1	8 000	8 000
Итого:					236 670

Зм = 7 360 + 464 912 = 236 670 (тг)

Для реализации программного обеспечения необходимы материалы на сумму 236 670тенге.

5.3 Расчет затрат на электроэнергию

Так как длч разработки ПП используется электрооборудование, то необходимо рассчитать затраты на электроэнергию.

Расчет электроэнергии, которая необходима для оборудования определяется по следующей формуле:

$$З_{эл.эн.обор.} = \sum M * K * S * T, \quad (5.3)$$

где M – потребляемая мощность, Вт;

K – коэффициент использования ($К_{исц}$ = 0,7..0,9);

T – время работы;

S – тариф (1кВт/ч = 17,81 тг).

Итоги по расчетам стоимости затрачиваемой электроэнергии представлены в таблице 5.3.2.

Таблица 5.3.2 – Затраты на электроэнергию

Наименование приборов	Паспортная мощность, кВт	Коэффициент мощности	Время работы оборудования, ч	Цена ЭЭ тг/кВтч	Сумма, тг.
Ноутбук	0,54	0,8	160	17,81	1 231
Модем	0,1	0,9	160	17,81	256,4
Принтер	0,4	0,9	16	17,81	102,6
Кондиционер	0,76	0,9	160	17,81	1 949,1
Освещение	0,3	0,7	160	17,81	598,4
Итого:					4137,5

$$З_{эл.эн.обор.} = 7\ 117\ (\text{тенге})$$

5.4 Расчет затрат на оплату труда

Для разработки программного обеспечения, как указывалось ранее, необходимо пять работников:

- руководитель;
- разработчик;
- бэкенд-программист;
- фронтенд-программист;
- тестировщик.

Сумму расходов на оплату труда можно рассчитать по следующей формуле:

$$З_{тр} = \sum ЧС_i * T_i \quad (5.4)$$

где $ЧС_i$ - часовая ставка i-го работника, тг;
T_i - трудоемкость разработки модели, чел.×ч; i - категория работника;
n - количество работников, занятых разработкой ПП.
Часовую ставку сотрудника рассчитать по следующей формуле:

$$ЧС_i = \frac{ЗП_i}{ФРВ_i} \quad (5.5)$$

где $ЗП_i$ - месячная заработная плата i-го работника, тг;
$ФРВ_i$ - месячный фонд рабочего времени i-го работника, час.

Рассчитаем часовую ставку каждого работника согласно формуле (5.5):

$$ЧС_{руководитель} = \frac{200\ 000}{30 * 8} = 833{,}33\ тг/ч$$

$$ЧС_{дизайнер} = \frac{100\,000}{30 * 8} = 416{,}66 \text{ тг/ч}$$

$$ЧС_{бэкенд-прог.} = \frac{150\,000}{30 * 8} = 625 \text{ тг/ч}$$

$$ЧС_{фронтенд-прог.} = \frac{150\,000}{30 * 8} = 625 \text{ тг/ч}$$

$$ЧС_{тестировщик} = \frac{80\,000}{30 * 8} = 333{,}33 \text{ тг/ч}$$

Расчеты затрат по оплате труда показаны в таблице (6.5).

Таблица 5.4.1 – Расчет заработной платы

Категория работника	Квалификация	Трудоемкость разработки ПП, час.	Часовая ставка, тг/ч	Сумма, тг.
Руководитель	Руководитель	120	833,33	100 000
Дизайнер	Дизайнер	60	416,66	25 000
Фронтенд-программист	Программист	90	625	56 250
Бэкенд-программист	Программист	90	625	56 250
Тестировщик	Программист	100	333,33	33 333
Итого:				270 833

Данные расчеты были сделаны с учетом того, что месячная заработная плата руководителя равняется 200 000 тенге, месячная заработная плата дизайнера равняется 100 000 тенге, месячная заработная плата бэкенд-программиста равняется 150 000 тенге, месячная заработная плата фронтенд-программиста равняется 150 000 тенге, месячная заработная плата тестировщика равняется 80 000 тенге.

5.5 Расчет затрат по социальному налогу

Согласно Налоговому кодексу Республики Казахстан социальный налог составляет 9,5% от фонта оплаты труда, поскольку в данном случае нет необходимости добавлять 1,5% на медицинскую страховку. Социальный налог можно рассчитать по следующей формуле:

$$С_н = (ФОТ - ПО) * 0{,}095 \quad (5.6)$$

где ПО - отчисления в пенсионный фонд, они составляют 10% от ФОТ.

$$ПО = 270\,833 * 0{,}1 = 27\,083{,}3 \text{ тенге}$$
$$С_н = (270\,833 - 27\,083{,}3) * 0{,}095 = 23\,156{,}22 \text{ тенге}$$

5.6 Амортизация основных фондов и прочие затраты

Нормы амортизации ОФ необходимо определить в соответствии с налоговым кодексом РК. Амортизацию ОФ можно определить по следующей формуле:

$$A_{\text{г}} = \frac{\text{С}_{\text{об}} * \text{Н}_{\text{а}}}{100} \quad (5.7)$$

где, $\text{С}_{\text{об}}$ – стоимость оборудования;

$\text{Н}_{\text{а}}$ – норма амортизации (норма амортизация = 25);

Формула (5.7) позволяет рассчитать нужную сумму для амортизационных отчислений за год для ноутбука:

$$A_{\text{г}} = \frac{190\,000 * 25}{100} = 47\,500 \text{ тенге}$$

Теперь необходимо рассчитать норму амортизации за период разработки:

$$A_{\text{г}} = \frac{47\,500 * 30}{365} = 3904{,}1 \text{ тенге}$$

Где 30 – это общее количество дней разработки. Подобным образом необходимо рассчитать норму амортизации для всего оборудования. Результаты расчетов приведены в таблице (5.6.1).

Таблица 5.6.1 – Амортизация ОФ

Наименование оборудования и ПО	Стоимость оборудования и ПО, тг	Годовая норма амортизации, %	Сумма амортизации за год, тг	Сумма амортизации за время разработки, тг
Ноутбук	190 000	25	47 500	3904,1
Принтер	38 670	25	9 667	794,58
Модем	8 000	25	2 000	164,38
Итого:			59 167	4 863,06

Смета расходов на разработку ПО.

На основе всех представленных расчетов необходимо оформить смету расходов на разработку ПО согласно форме, которая приведена в таблице (5.6.2).

При этом нужно добавить прочие расходы, к которым относится – материальные расходы и оплата за интернет.

$$З_{пр} = 14\,100 + 4000 = 18\,100$$

Таблица 5.6.2 – Смета затрат на разработку ПО

Статьи затрат	Сумма, тг
Затраты на оборудование	236 670
Затраты на программное обеспечение	0
Затраты на оплату труда	270 833
Социальные налоги	23 156,22
Затраты на электроэнергию	4137,5
Амортизация основных фондов	4 863,06
Прочие расходы	18 100
Итого по смете:	557 759,78

5.7 Определение возможной (договорной) цены ПО

Прибыль программного продукта равна 25% от общей стоимости разработки приложения и может быть вычислен как:

$$Ц_{п} = З_{нир} * 0{,}25 = \ 557\,759{,}78 * 0{,}25 = 139439{,}94 \text{ тенге}$$

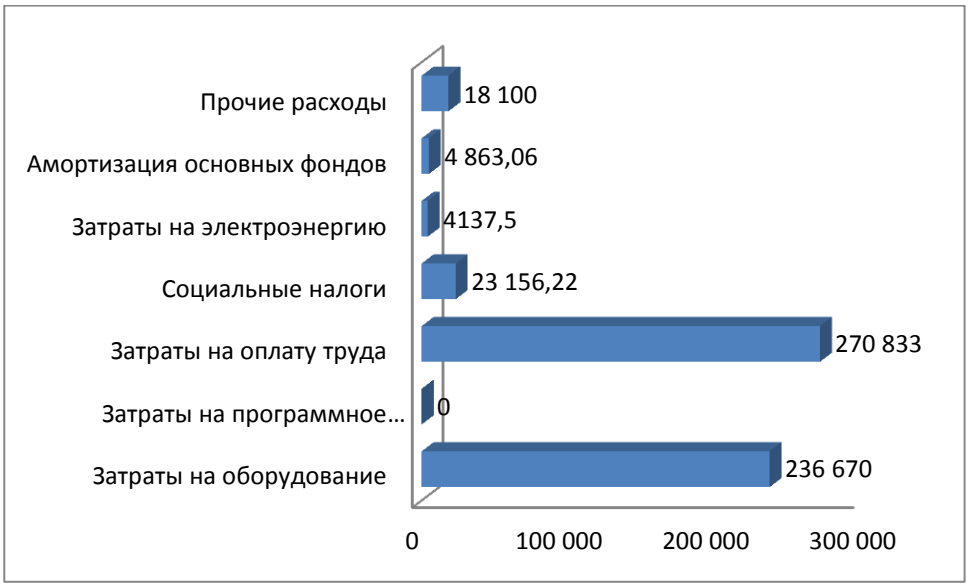

Рисунок 5.7.1 – Диаграмма затрат

Стоимость программного обеспечения определяется на основе качества разработанного продукта, сроков его разработки и производительности продукта. Стоимость $Ц_{Д}$ для программного обеспечения можно рассчитать по следующей формуле:

$$Ц_{Д} = З_{нир}\left(1 + \frac{P}{100}\right), \qquad (5.8)$$

где $З_{нир}$ – затраты на разработку программного обеспечения, тг;

Р – средний уровень рентабельности ПО, (%). Данный параметр принят равным 25%.

$$Ц_Д = 557\,759{,}78 + 557\,759{,}78 * 0{,}25 = 697199{,}72 \text{ тенге}$$

Далее необходимо определить стоимость реализации с учетом НДС, ставка НДС устанавливается законодательством РК. На 2019 года ставка НДС составляет 12%. Стоимость реализации, учитывая НДС можно рассчитать по следующей формуле:

$$Ц_р = Ц_д + Ц_д * НДС, \quad (5.9)$$

$$Ц_р = 697199{,}72 + 697199{,}72 * 0{,}12 = 780863{,}69 \text{ тг.}$$

Вывод

В данной части дипломного проекта содержится экономические расчеты, которые показывают затраты необходимые при разработке выбранного Web-приложения. Расчеты включают в себя:

- определение трудоемкости разработки программного обеспечения;
- расчет затрат на разработку ПО;
- определение ценности готового продукта;
- оценка результатов работы программного обеспечения.

По выполненным расчетам можно сказать, что полная стоимость выполненного проекта равна 780863,692 тг с учетом НДС, расчеты так же показали, что возможная прибыль равна 139439,94 тг .

Заключение

В заключение данной дипломной работы хотелось бы сказать, что разработка веб-приложений сложный технический процесс, результат которого даже если соответствуют рамкам поставленной задачи, не всегда является «идеальной» системой. Однако при правильном подходе с четко определенными требованиями и технологиями реализации можно спроектировать и реализовать надежное и современное приложение, способное предоставить заказчику, а так же его дальнейшим пользователям оптимальное решение на несколько лет вперед.

В ходе решения поставленной цели дипломной работы были выполнены следующие задачи:

- было предоставлено описание и структура веб-приложения;
- исходя из тенденций и существующего опыта, был выбран определенный набор технологий с их детальным описанием;
- с помощью выбранных технологий был создан сервис с заранее определенными возможностями;
- для созданного приложения была создана система защиты;
- был проведен анализ уровня безопасности приложения;
- был проведен анализ эффективности выбранных технологий.

Так же на основании проведенного анализа эффективности были сделаны следующие выводы:

- для создания приложения на выбранном стеке нет необходимости изучать другие языки кроме JavaScript;
- данные хранятся и передаются в JSON-формате, что убирает время на форматирование данных;
- node.js позволяет создавать приложения с неблокирующим вводом/выводом, которые способны обрабатывать несколько запросов одновременно, что увеличивает скорость работы приложения по сравнению с многопоточными языками программирования;
- node.js не имеет строгих правил или жестких зависимостей, что оставляет простор для созидания и креативности при разработке приложений. разработчики сами выбирают архитектуру, зависимости;
- время необходимое на разработку, а так же общее количество кода клиентской части, благодаря возможностям angular было уменьшено по сравнению с базовой разработкой при помощи javascript;
- для тестирования и запуска приложения нет необходимости в установке дополнительных веб-серверов по типу apache;
- роутинг в angular позволил создать приложение, которое после первой загрузке не перезагружается, что обеспечивает удобство и быстроту при взаимодействие с сайтом;
- при создании системы защиты для каждого приложения нужно использовать индивидуальный подход;

- для того чтобы обеспечить максимальную безопасность необходимо применить комплексный подход, принимая во внимание все уязвимые места.

Несмотря на то, что выбранный стек MEAN является прекрасной альтернативой традиционному стеку LAMP, он все еще находится на ранних стадиях. Он принят некоторыми амбициозными организациями, поскольку большинство из них все еще придерживаются «базовым» технологиям. Несмотря на это, как видно из построенного приложения и его дальнейшим анализом выбранный стек во многом превосходит базовые аналоги и имеет прекрасные перспективы в будущем.

Список литературы

1 «Технология AJAX» / URL: https://wiki.rookee.ru/ajax/(дата обращения: 02.02.2019)

2 Александр Суханов: Защита данных веб-приложений от внутренних угроз / URL: https://www.anti-malware.ru/practice/solutions/web-applications-internal-threats-security/(дата обращения: 09.03.2019)

3 Nick Nauert. Концепция MVC для чайников/ URL: https://ruseller.com/lessons.php?id=666/(дата обращения: 05.02.2019)

4 «Выбор технологий для большого и не очень большого веб-проекта»/ URL: https://habr.com/ru/company/SECL_GROUP/blog/315734/ (дата обращения: 05.02.2019)

5 Кириченко А. В., Хрусталев А. А. HTML5 + CSS3. Основы современного веб-дизайна – СПб: «Наука и Техника», 2018. – 352 с., ил.

6 Джон Дакетт; [пер. с англ. М. А. Райтмана]. HTML и CSS. Разработка и дизайн веб-сайтов – Мс: Эксмо, 2019. -480 с.: ил. - (Мировой компьютерный бестселлер)

7 А. Н. Васильев. JavaScript в примерах и задачах – Москва: Издательство «Э», 2017. – 720 с. – (Российский компьютерный бестселлер)/ (дата обращения: 08.02.2019)

8 Джон Дакетт; [пер. с англ. М. А. Райтмана]. JavaScript и jQuery. Интерактивная веб-разработка – Мс: Эксмо, 2017. – 640 с.: ил. – (Мировой компьютерный бестселлер)

9 Кэйл Банкер. MongoDB в действии, 4-е издание /– 2017. – 539 с.

10 Д. Майк Кантелон, Марка Хартер Node.js в действии, 6-е издание. – 2018. – 432 с., ил./ (дата обращения: 08.02.2019)

11 «Введение в модель данных MongoDB» / С.Д. Кузнецов – М.: Национальный Открытый Университет «ИНТУИТ», 2016./ (дата обращения: 08.02.2019)

12 «Руководство по TypeScript» https://metanit.com/web/typescript/(дата обращения: 08.02.2019);

13 А. П. Никольский. JavaScript на примерах – СПб.: «Наука и техника», 2017. – 272 с.

14 «Чем на самом деле является Node.js / URL: https://habr.com/ru/post/420123/(дата обращения: 08.02.2019)

15 Я. Файн, А. Моисеев. Angular и TypeScript. Сайтостроение для профессионалов – СПб.: Питер, 2018. – 464 с.

16 «SQL или NoSQL – вот в чём вопрос» / Блог компании «RUVDS.com». – 2017. URL: https://m.habr.com/ru/company/ruvds/blog/324936/

17 George Berezhnoy. Аутентификация с помощью JSON Web Token/ URL: https://codex.so/jwt/(дата обращения: 28.02.2019)

18 Nick Nauert. A Tutorial We Newbs Can Understand/ URL: https://medium.com/@nicknauert/mongooses-model-populate-b844ae6d1ee7 /(дата обращения: 12.02.2019)

19 «Node.js в PayPal» / URL: https://habr.com/ru/post/324912/ (дата обращения: 12.02.2019)

20 «Как защитить веб-приложение: основные советы, инструменты, полезные ссылки»/ URL: https://tproger.ru/translations/webapp-security/

21 «Как выявить слабые места на сайтах конкурентов и обогнать их в поисковых системах»/ URL: https://worldwideshop.ru/products/kak-vy-yavit-slaby-e-mesta-na-sajtah-konkurentov-i-obognat-ih-v-poiskovy-h-sistemah/ (дата обращения: 12.02.2019)

22 Саймон Холмс. Стек MEAN, 2-е издание. – СПб.: Символ-Плюс, 2018. – 622 с.

23 «Чем хорош Node.js: практика современного веб-программирования»/URL: https://skillbox.ru/media/code/chem_khorosh_node_js/(дата обращения: 12.02.2019)

24 Stas Bagretsov. Введение в JSON/ URL: https://medium.com/stasonmars/(дата обращения: 12.02.2019)

25 «Express Validator Tutorial» / URL: https://auth0.com/blog/express-validator-tutorial/(дата обращения: 12.02.2019)

26 Курс «Angular7» / «WebForMySelf.com Team»; – 2019. URL: udemy.com

27 Адам Фримен. Angular для проффессионалов – СПб.: Символ-Плюс, 2017. – 980 с.

28 «Волшебство Git» / Бен Лин. – 2016. / URL: http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/ru/ch01.html/(дата обращения: 12.02.2019)

29 «Бывают SQL-инъекции! А возможны ли NoSQL-инъекции?»/ URL: https://habr.com/ru/company/xakep/blog/143909/ (дата обращения: 12.02.2019)

Приложение А

Техническое задание

Общие требования

Сайт должен быть разработан с использованием новейшего стека технологий MEAN. Так же должна быть построена надежная система для защиты сайта.

По окончании работ Исполнитель обязан предоставить полностью функционирующий сайт, исходные графические материалы по дизайну.

Требования к дизайну сайта

Дизайн должен быть выдержан в строгих и мягких тонах. Использовать преимущественно сине-оранжевые оттенки. Дизайн сайта должен быть выполнен с использованием языка HTML и CSS, при необходимости для создания отдельных графических элементов допустимо использование технологии Angular.

Сайт должен корректно отображаться в браузерах Microsoft Internet Explorer 6.0, 7.0; Mozilla FireFox 2.0, 3.0; Opera 9.0; Chrome.

Необходимо создать структуру (шаблон) сайта, состоящую из следующих элементов:

- “шапка” (хедер). в данном блоке необходимо расположить логотип;
- блок отображения пользователя, который вошел в систему, в том случае если вход не был выполнен, то должна отображаться кнопка входа, которая переносит пользователя на страницу входа/регистрации;
- блок отображения категорий;
- блок для входа зарегистрированных пользователей на сайт;
- блок с отображением сообщества;
- блок с отображением информации о сообществе с фотографией;
- блок с отображением постов выбранного сообщества;
- “подвал” (футер) сайта. в данном блоке необходимо разместить краткую контактную информацию о предприятии;
- блок отображения сообществ, которые загружаются с сервера в соответствии с выбранной категорией;
- блок с добавлением поста в сообщество.

В общем, должно получиться 3 страницы: главная, регистрация или вход и страница сообщества.

Помимо всего прочего, у пользователя должна быть возможность загружать фотографии в посте.

На рисунке 1, 2, 3 представлена графическая схема шаблона сайта.

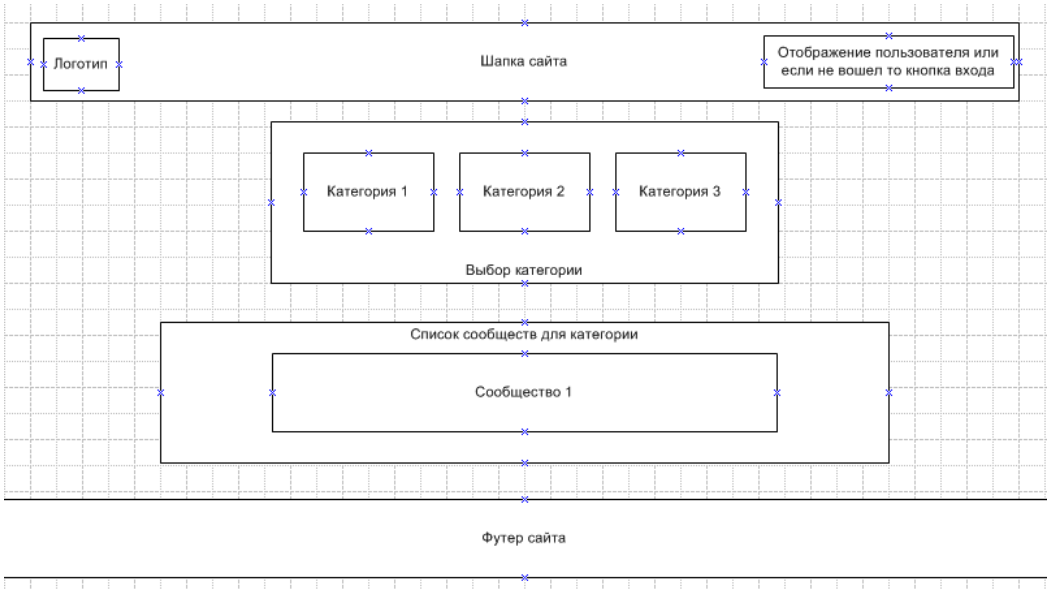

Рисунок 1 – Графическая схема шаблона сайта

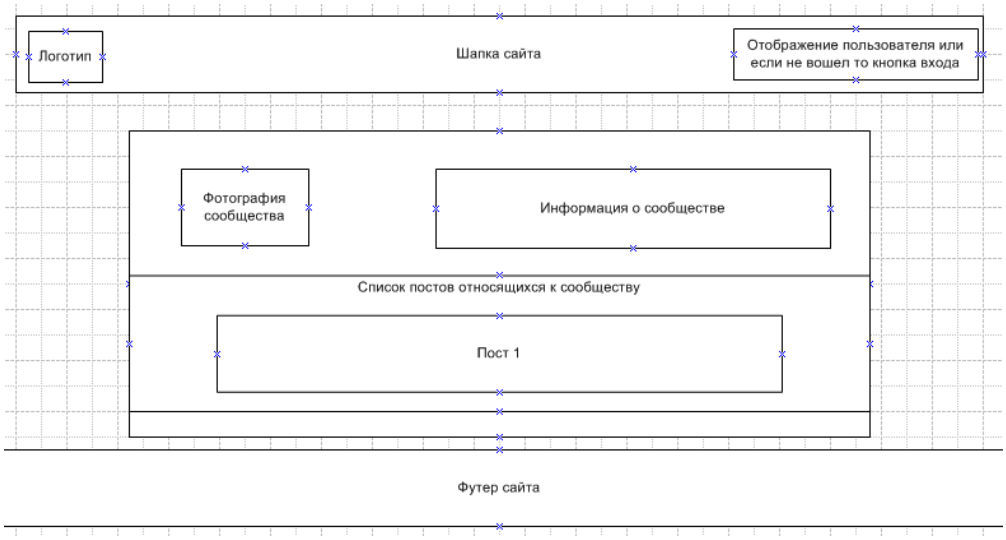

Рисунок 2 – Графическая схема шаблона сайта

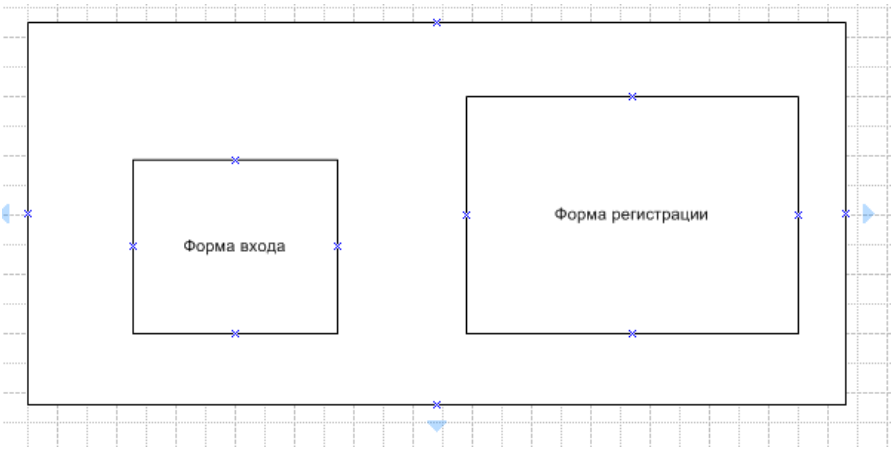

Рисунок 3 – Графическая схема шаблона сайта

Необходимо также обеспечить безопасность и надежность веб-приложения. Он должен противостоять большинству типов атак, распространённых в сети Интернет.

Требования к функциональности сайта

Необходимо обеспечить возможность предоставлять пользователям удобный и быстрый интерфейс взаимодействия без перезагрузки страниц.

Сайт должен позволять пользователям:

- осуществлять навигацию по сайту (переход между страницами);
- загружать фотографии на сервер;
- загружать текстовые данные на сервер;
- выполнять вход на сайт как зарегистрированный пользователь для возможности просмотра информации или ее добавления.

Требования к содержимому сайта

Необходимо создать следующие страницы сайта:

- главная страница сайта;
- страница сообщества;
- страница входа/регистрации.

Детальное описание страниц сайта

Главная страница - вначале данной страницы необходимо вставить изображение (логотип) сайта, так же там должна отображаться информация о пользователе, если он вошел в систему, список доступных категорий, кнопка выбора определенной категории, блок с доступными сообществами, футер сайта с информацией о компании.

Страница сообщества - на данной странице необходимо разместить краткую информацию о выбранном сообществе, кнопка, которая открывает форму для добавления поста с возможностью добавить фотографию и блок с доступными постами сообщества.

Страница входа/регистрации - на данной странице необходимо разместить две формы – для входа и для регистрации, в случае удачного входа пользователя перенесет на главную страницу.

Приложение Б

Листинг программы

Script.js

```
const conf = require('./config/web-server.js');
const dbconnect = require('./DB/database.js');
var express = require('express');
var path = require('path');
var bodyParser = require('body-parser');
var route = require('./routes/api.js');
var app = express();
var helmet = require('helmet')
var http = require('http');
var server = http.createServer(app);
var passport = require('passport');
require('./config/passport.js');
const xssFilter = require('x-xss-protection')
app.use(xssFilter());
app.use(express.static(path.join(__dirname,'dist/diplom')))
app.use('/static', express.static('./static'));
var helmet = require('helmet')
app.use(helmet())
dbconnect.start();
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended:false}));
app.use('/api', route);
app.use(function(req, res, next) {
//set headers to allow cross origin request.
   res.header("Access-Control-Allow-Origin", "*");
   res.header('Access-Control-Allow-Methods', 'PUT, GET, POST, DELETE, OPTIONS');
   res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type,
Accept");
   next();
});
app.use(passport.initialize());

app.get('*',function (req, res, next) {
        res.sendFile(path.join(__dirname,'dist/diplom/index.html'));
});
server.listen(conf.port, function () {

        console.log('Listining port 1337...');
        });
```

api.js

```
const express = require('express');
```

```
const router = express.Router();
const {check, validationResult} = require('express-validator/check');
var jwt = require('express-jwt');
var auth = jwt({
  secret: 'MY_SECRET'
});
var ctrlProfile = require('../Controllers/profile');
var ctrlPosts = require('../Controllers/posts');
router.post('/login', [
    check('username', 'Not valid username').isLength({ min: 3 }).trim().escape(),
    check('password', 'Not valid password').trim().escape().isLength({min: 5}),
  ], ctrlProfile.login);
router.post('/signup', [
    check('username', 'Not valid username').isLength({ min: 3 }).trim().escape(),
    check('name', 'Not valid name').isLength({ min: 1 }).trim().escape(),
    check('surname', 'Not valid surname').isLength({ min: 1 }).trim().escape(),
    check('password', 'Not valid password').trim().escape().isLength({min: 5}),
  ], ctrlProfile.signup);
router.post('/uploadFile', ctrlProfile.uploadFoto);
router.get('/profile', auth, ctrlProfile.profile);
router.get('/tree', ctrlPosts.getTree);
router.post('/branch', ctrlPosts.getBranch);
router.post('/post', ctrlPosts.getPost);
router.post('/addtree', ctrlPosts.addTree);
router.post('/addbranch', ctrlPosts.addBranch);
router.post('/addpost', ctrlPosts.addPost);
module.exports = router;
```

posts.js

```
var mongoose = require('mongoose');
var Tree = mongoose.model('Tree');
var Branch = mongoose.model('Branch');
var Post = mongoose.model('Post');
var multer = require('multer');
var fs = require('fs');
var storage = multer.diskStorage({
  destination: function (req, file, cb) { cb(null, 'static/img')
            filename: function (req, file, cb) {
    cb(null, Date.now() + '.jpg')
  }
})
var upload = multer({ storage: storage }).single('file');
module.exports.getTree = function(req, res) {
    Tree
      .find({})
      .exec(function(err, post) {
        console.log(post);
        res.status(200).json(post);
```

```
    });
}

module.exports.getBranch = function(req, res) {
  console.log(req.body);
  if(req.body.name){
    Branch
      .find({tree:req.body.name})
      .exec(function(err, post) {
        console.log(post);
        res.status(200).json(post);
      });
    }else{
      Branch
        .find({_id:req.body.id})
        .exec(function(err, post) {
          console.log(post);
          res.status(200).json(post);
        });
    }
}

module.exports.getPost = function(req, res) {
  var name ;
  Branch.find({_id:req.body.id}).exec(function(err, post) {
      name = post[0].name;
      Post
        .find({branch:name})
        .exec(function(err, post) {
          // console.log(post);
          res.status(200).json(post);
              });
    });
}
module.exports.addTree = function(req, res) {
  console.log(req.body);
  var newTree = new Tree({
    name:req.body.tree
  });
  newTree.save(function(err) {
    if (err) {
      console.log(err);
      return res.json({success: false, msg: 'Tree already exists.'});
    }
    res.json({success: true, msg: 'Successful created new tree.'});
  });
}
module.exports.addBranch = function(req, res) {
  console.log(req.body);
```

Продолжение приложения Б

```
  var newTree = new Branch({
    name:req.body.name,
    tree:req.body.tree
  });
  newTree.save(function(err) {
    if (err) {
      console.log(err);
      return res.json({success: false, msg: 'Tree already exists.'});
    }
    Branch
      .find({})
      .exec(function(err, post) {
        console.log(post);
        res.json(post);
      });
  });
}
module.exports.addPost = function(req, res) {
  var path = '';
  var maxsize = 2 * 1024 * 1024;
  var supportMimeTypes = ['image/jpg', 'image/jpeg', 'image/png'];
      upload (req, res, function (err) {
 if(req.file){
 if (req.file.size > maxsize) {
        fs.unlinkSync(req.file.path);
        return res.json({success: false, msg: 'File is so more than 2 Mb'});
       }
       if(supportMimeTypes.indexOf(req.file.mimetype) == -1) {
        fs.unlinkSync(req.file.path);
        return res.json({success: false, msg: 'Unsupported mimetype'});
       }
      }
       if (err) {
         console.log(err);
       }
       if(req.file){
         path=req.file.path;
       }
       var newPost = new Post({
        name:req.body.name,
        tree:req.body.b_name,
        owner:req.body.owner,
        branch:req.body.t_name,
        text:req.body.text,
        img:path
       });
       newPost.save(function(err) {
        if (err) {
          console.log(err);
```

```
          fs.unlinkSync(req.file.path);
          return res.json({success: false, msg: 'Name already exists.'});
        }
          console.log('No Error')
      });

  });
}

module.exports.unfollow = function(req, res){
  User.findByIdAndUpdate(req.body.userId, {$pull: {following: req.body.unfollowId}}, (err,
result) => {
    if (err) {
      return res.status(400).json({
        error: errorHandler.getErrorMessage(err)
      }) }
 })
}
module.exports.follow = function(req, res){
  User.findByIdAndUpdate(req.body.userId, {$push: {following: req.body.followId}}, (err, result)
=> {
    if (err) {
      return res.status(400).json({
        error: errorHandler.getErrorMessage(err)
      })
    }
  })
}
```

Profile.js

```
var passport = require('passport');
var mongoose = require('mongoose');
var sanitize = require('mongo-sanitize');
const {check, validationResult} = require('express-validator/check');
var crypto = require('crypto'),
    algorithm = 'aes-256-ctr',
    password = 'saligus';
var User = mongoose.model('User');
var multer = require('multer');
var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads/')
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + '.jpg') //Appending .jpg
  }
})
var upload = multer({ storage: storage }).single('photo');
```

Продолжение приложения Б

```
module.exports.login = function(req, res, next) {
  const errors = validationResult(req);
   if (!errors.isEmpty()) {
     console.log(errors.array());
     return res.json({
       success : false,
       error : errors.array()
     });
   } else { passport.authenticate('local', function(err, user, info){
 var token;
      // If Passport throws/catches an error
      if (err) {
        console.log(err);
        res.status(404).json(err);
        return;
      }
      // If a user is found
      if(user){
        token = user.generateJwt();
        res.status(200);
        res.json({
          success : true,
          "token" : token
        });
      } else {
        // If user is not found
        res.status(401).json(info);
      }
    })(req, res, next);
  }
}
module.exports.profile = function(req, res) {
  console.log(req.payload);
  if (!req.payload._id) {
    res.status(401).json({
      "message" : "UnauthorizedError: private profile"
    });
  } else {
    User
      .findById(req.payload._id)
      .exec(function(err, user) {
        res.status(200).json(user);
      });
  }
}
module.exports.signup = function(req, res) {
    const errors = validationResult(req);
     if (!errors.isEmpty()) {
       console.log(errors.array());
```

```
      return res.json({
        success : false,
          error : errors.array()
          });
    } else {
     console.log(req.body);
     if (!req.body.username || !req.body.password) {
      res.json({success: false, msg: 'Please pass username and password.'});
     } else {
      var newUser = new User({
       username: req.body.username,
       password: "none",
       name: req.body.name,
       surname: req.body.surname
      });
      newUser.setPassword(req.body.password);
      // save the user
      newUser.save(function(err) {
       if (err) {
         return res.json({success: false, msg: 'Username already exists.'});
       }
       res.json({success: true, msg: 'Successful created new user.'});
      });
     }
    }

}
module.exports.uploadFoto = function(req, res, next) {
  var path = '';
   upload(req, res, function (err) {
     if (err) {
       // An error occurred when uploading
       console.log(err);
       return res.status(422).send("an Error occured")
     }
     console.log(req.file)
    // No error occured.
     path = req.file.path;
     return res.send("Upload Completed for "+path);
});
}
function encrypt(text){
  var cipher = crypto.createCipher(algorithm,password)
  var crypted = cipher.update(text,'utf8','hex')
  crypted += cipher.final('hex');
}
function decrypt(text){
  var decipher = crypto.createDecipher(algorithm,password)
  var dec = decipher.update(text,'hex','utf8')
```

Продолжение приложения Б

```
  dec += decipher.final('utf8');
  return dec;
}
```

Header.component.html

```
<!-- <header>
   <h1>My-Gradient</h1>
   <p *ngIf="flag" >Вы не вошли <a (click)="openDialog()">Войдите </a> или <a
[routerLink]="['/signup']">Зарегестрируйтесь</a></p>
   <a>{{username}}</a>
   <p><button class="btn btn-success" (click)="logout()"*ngIf="!flag" >Выйти</button></p>
</header> -->
<header class="mainhead">
<div class="topone">
   <div class="inputtopone">
     <a href="#"><img width="130px" height="auto" src="assets/img/3.png"></a>
        <!-- Menu -->
        <nav class="menutopone">
           <ul class="hospital">
             <li class="menuitem1" (click)="gomain()"><b>Главная</b></li>
             <li class="menuitem2" (click)="gomain()"><b>О нас</b></li>
             <li class="menuitem3" (click)="gomain()"><b>Контакты</b></li>
             <li class="menuitem4" (click)="gomain()"><b>Помощь</b></li>
           </ul>
        </nav>
       <!-- Menu -->
       <!-- leftmenu -->
         <div class="leftlogin">
           <button class="butttonlastone" (click)="logout()"*ngIf="!flag" >Выйти</button>
           <button class="buttonnumone" *ngIf="flag"><b
(click)="goreg()">Войти/Регистрация</b></button>
           <button class="buttonnumone" *ngIf="!flag" >{{username}}</button>
             <a href=""><img width="43px" height="auto" src="assets/img/4.png"></a>
        </div>
       <!-- leftmenu -->
   </div>
</div>
</header>
```

Header.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MatDialog, MatDialogConfig } from '@angular/material';
import { PopupComponent } from '../popup/popup.component';
import { AuthenticationService } from '../Auth.service';
import { Router } from '@angular/router';

@Component({
```

```
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit {
  datas: any;
  username: String;
  flag: boolean = true;
  constructor(private router: Router, public dialog: MatDialog, private auth:
AuthenticationService) { }
  openDialog() {
    const dialogConfig = new MatDialogConfig();
    dialogConfig.width = "60%";
    this.dialog.open(PopupComponent);
  }
  logout() {
    this.auth.logout();
    const url = this.router.url;
    if(url == '/main'){
      this.router.navigateByUrl('/signup', {skipLocationChange: true}).then(() =>
      this.router.navigate([url]));
    }
  }
  goreg(){
    this.router.navigate(['/signup']);
  }
  gomain(){
    this.router.navigate(['/main']);
  }
  ngOnInit() {
    console.log(this.auth.getUserDetails());
    if(this.auth.isLoggedIn()){
      this.datas = this.auth.getUserDetails();
      this.username = this.datas.username;
     this.flag = false;
    }else{
      this.flag = true;
    }
  }
}
```

Login.component.html

```
<body>
<header class="mainhead">
  <!-- Menu -->
      <app-header></app-header>
    <!-- Menu -->
    <!-- aside -->
```

Продолжение приложения Б

```
    <div class="asideoftext">
      <p class="text00"><span class="text01">Настало время выбрать</span> <span
class="text02"> интересующее направление</span><br><span class="text03">
Выбрал?</span> <span class="text01"> Тогда</span> <span class="text03"> нажимай</span>
<span class="text01"> и</span> <span class="text03"> все</span></p>
    </div>
<div class="shkaff">
                        <div class="engel">
                        <div class="divindivin" *ngFor="let tree of trees" (click)="select($event)"
[id]="tree.name">
                                <p class="imgdiv2"><img  width="75px" [src]="tree.img"></p>
                                <p>{{tree?.name}}</p>
                        </div>
                        </div>
                        <div class="divofbutton"><button class="buttonpux"
(click)="checks()">ВЫБРАТЬ</button></div>
                </div>

   <!-- slider -->
</header>
<!-- header -->
<!-- MAIN -->
  <main class="maincontent" id="maincontent" (click)="dropAll($event)">
   <div class="imgdiv1"><img src="assets/img/14.png" (click)="drops()"><p></p></div>
     <aside class="mainaside">
        <button class="button01" (click)="drop()"><span style="margin-left:-
50px;">Фильтры</span> <img style="position: absolute; right:13px;bottom:-1px"
src="assets/img/15.png"></button>
<div class="dropdown-menu1" id="dropdown-menu1" *ngIf="show" >
<div class="fitstone"><div><img src="assets/img/16.png"></div><span style="display:
flex;margin-left:-2px"><span>По</span> <span> рейтингу</span></span><div><label
style="margin-left:30px" class="container">
<input type="checkbox" checked="checked">
<span class="checkmark"></span>
</label></div></div>
<div class="fitstone"><div><img src="assets/img/17.png"></div><span style="display: flex;
margin-left:-2px"><span>По </span> <span> дате</span></span><div><label style="margin-
left:60px" class="container">
<input type="checkbox" >
<span class="checkmark"></span>
</label></div></div>
<div class="fitstone"><div><img src="assets/img/18.png"></div><span style="display:
flex;margin-left:-2px"><span>По </span> <span> просмотрам</span></span><div><label
style="margin-left:1px" class="container">
<input type="checkbox" >
<span class="checkmark"></span>
</label></div></div>
</div>
      <p class="text05">Доступные сообщества</p>
```

Продолжение приложения Б

```
      <!-- informationbox1 -->
      <div class="boxinformation" *ngFor="let branch of branches" id="branch?.name">
        <div class="box1information">
         <div class="divdiv"><img [src]="branch.img" style="width: 128px;height: 83px;"><div
class="divdivdvi"><span class="text06">Название:</span> <span
class="text07">{{branch?.name}}</span></div><p class="text08">Оценка: <div><img
style="margin-top:8px;margin-left:5px" src="assets/img/20.png"></div></div>
        <p class="text09">{{branch?.text}}</p>
        <div class="fdflqwe"><button class="firstleftbutton">подписаться</button><button
class="firstrightbutton">2</button>
         <button class="butttonlastone" (click)="navigate(branch?.name,branch?._id)">Перейти
→</button>
        </div>
        </div>
      </div>
    </aside>
    <div class="imgbottomone"><img src="assets/img/13.png"></div>
  </main>
<!-- MAIN -->
<!-- footer -->
  <footer class="footercontent">
 <div class="inputdiveonefoot">
      <p class="inputdiveonefoottext">О In-Gradient |  Наша политика безопасности и защиты
данных</p>
      <p class="maintextfoot">
    © Copyright 2019 by In-Gradient. Все права защищены <br><span style="float:right;
margin-top:3px">Site by SAlaG</span></p>
          </div>
  </footer>
  <script src="assets/js/slick.js"></script>
</body>
<!-- <main>
  <table>
    <tr>
      <td>
      <form>
        <p><select [(ngModel)]="currentTree" name="tree">
          <option for="asd" *ngFor="let tree of trees" [value]="tree?.name">
            {{tree?.name}}
          </option>
        </select></p>
       <p><input type="button" value="Выбрать" (click)='checks()'></p>
       <label for="inputEmail" class="sr-only">Название</label>
       <input type="text" class="form-control"[(ngModel)]="newTree" name="newTree" required/>
       <p><input type="button" value="Добавить дерево" (click)='addTree()'></p>
      </form>
      </td>
    </tr>
    <tr>
```

Продолжение приложения Б

```
    <td>
    <form *ngIf="flagT" >
      <p><select [(ngModel)]="currentBranch" name="tree">
       <option for="asd" *ngFor="let branch of branches" [value]="branch?.name">
         {{branch?.name}}
       </option>
      </select></p>
     <p><input type="button" value="Выбрать" (click)='getPosts()'></p>
     <label for="inputEmail" class="sr-only">Название</label>
     <input type="text" class="form-control"[(ngModel)]="newBranch" name="newBranch"
required/>
     <p><input type="button" value="Добавить ветку" (click)='addBranch()'></p>
    </form>
    </td>
</tr>
  <tr>
    <td>
    <form *ngIf="flagP">
     <label for="inputEmail" class="sr-only">Название</label>
     <input type="text" class="form-control"[(ngModel)]="newPost" name="newPost"
required/>
     <label for="inputEmail" class="sr-only">Текст</label>
     <input type="text" class="form-control"[(ngModel)]="newPostText" name="newPostText"
required/>
     <p><input type="button" value="Добавить post" (click)='addPost()'></p>
     <div *ngFor="let post of posts">
       <p>Название leaf - {{post?.name}}</p>
       <p>Название ветки - {{post?.branch}}</p>
       <p>Название дерева - {{post?.tree}}</p>
       <p>{{post?.text}}</p>
       <br>
     </div>
    </form>\
    </td>
    </tr>
 </table>
</main> -->
```

Login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from "@angular/router";
import { Observable } from 'rxjs/Observable';
import { PostService } from '../post.service';
import { ElementRef, Renderer, ViewChild } from '@angular/core';
import * as $ from 'jquery';
import {  FileUploader } from 'ng2-file-upload';
//define the constant url we would be uploading to.
```

Продолжение приложения Б

```
const URL = '/api/uploadFile';
// import { AuthenticationService, TokenPayload } from '../authentication.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  newTree: '';
  newBranch: '';
  newPost: '';
  newPostText: '';
  trees: any;
  branches: any;
  posts: any;
  show: boolean = false;
  currentTree: '';
  currentBranch: '';
  flodClass:string = '';
  flagT: boolean = false;
  flagP: boolean = false;
  public uploader:FileUploader = new FileUploader({url: URL, itemAlias: 'photo'})
  constructor(private el: ElementRef, private post: PostService, private http: HttpClient, private
router: Router, private renderer: Renderer) { }
  @ViewChild('ulEl') ulEl: ElementRef;
  checks(){
    console.log(this.currentTree);
    if(!this.currentTree){
      this.post.getBranch({name:'Программирование'}).subscribe(tree => {
        console.log(tree)
        this.branches = tree;
        }, (err) => {
          console.error(err);
        });
    }else{
    this.post.getBranch({name:this.currentTree}).subscribe(tree => {
      console.log(tree)
      this.flagT = true;
      this.branches = tree;
      }, (err) => {
        console.error(err);
      });
    }
  }
  addTree(){
    this.post.addTree({tree:this.currentTree}).subscribe(tree => {
      console.log(tree);
      if(tree['success']){
```

```
      this.getTree();
}
    }, (err) => {
      console.error(err);
    });
  }
  addBranch(){
    this.post.addBranch({tree:this.currentTree, name:this.newBranch}).subscribe(tree => {
      console.log(tree);
      if(tree['msg']){
        console.log('Error')
      }else{
        this.branches = tree;
      }
      }, (err) => {
        console.error(err);
      });
  }
  addPost(){
    this.post.addPost({name:this.newPost, post:this.newPostText, tree:this.currentTree,
branch:this.currentBranch}).subscribe(tree => {
      console.log(tree);
      if(tree['msg']){
        console.log('Error')
      }else{
        this.posts = tree;
      }
      }, (err) => {
        console.error(err);
      });
  }
  navigate(name,id){
    this.router.navigate(['/branch/'+id]);
  }
  getTree(){
   this.post.getTree().subscribe(tree => {
```



```
     console.log(tree);
     this.trees = tree;
     }, (err) => {
       console.error(err);
     });
 }
 drop(){
  if(this.show == true){
this.show = false;
  }else{
    this.show = true;
```

```
  }
}
  drops(){
    $('html,body').animate({scrollTop:0}, 1300);
   }
   dropAll(event){
     var res = $(".dropdown-menu1");
     if ($(event.target).closest(res).length || $(event.target).closest('.button01').length) return;
     this.show = false;
   }
select(event){
// $('.engel div').css('background-color','rgba(218,128,75,0.4)').css('box-shadow','3px 3px 30px
white').not(this).css('background-color','rgba(37,55,96,0.7)').css('box-shadow','none');
var tar = event.target;
if(tar.nodeName=="DIV"){
  if(event.target.classList.contains('divindivinClick')){
    $(tar).removeClass('divindivinClick');
    return;
  }
  this.currentTree=tar.attributes.id.nodeValue;
  this.clear();
  $(tar).addClass('divindivinClick');
}else{
  for (let i = 0; i < 3; i++) {
      tar = tar.parentNode;
      if(tar.nodeName=="DIV"){
        if(tar.classList.contains('divindivinClick')){
          $(tar).removeClass('divindivinClick');
          break;
        }
        this.clear();
        $(tar).addClass('divindivinClick');
        this.currentTree=tar.attributes.id.nodeValue;
        break;
      }
  }
}
}
clear(){
  for (let i = 0; i < this.trees.length; i++) {
var el = document.getElementById(this.trees[i].name);
      console.log(el);
      $(el).removeClass('divindivinClick');
  }
}
upload() {
    //locate the file element meant for the file upload.
        let inputEl: HTMLInputElement = this.el.nativeElement.querySelector('#photo');
    //get the total amount of files attached to the file input.
```

Продолжение приложения Б

```
      let fileCount: number = inputEl.files.length;
   //create a new fromdata instance
      let formData = new FormData();
   //check if the filecount is greater than zero, to be sure a file was selected.
      if (fileCount > 0) { // a file was selected
         //append the key name 'photo' with the first file in the element
            formData.append('photo', inputEl.files.item(0));
         //call the angular http method
        return this.http.post(URL, formData).subscribe(resp => {
         console.log(resp);
        }, err => {
         // this.message = err.error.msg;
        });
  }
}

 ngOnInit() {
    this.getTree();
    this.checks();
     this.uploader.onAfterAddingFile = (file)=> { file.withCredentials = false; };
     this.uploader.onCompleteItem = (item:any, response:any, status:any, headers:any) => {
         console.log("ImageUpload:uploaded:", item, status, response);
     };
  }
 }
```

Page.component.html

```
<body>
<div class="upheader">
 <header class="mainhead">
<app-header></app-header>
</header>
<main class="twosectioninto" >
   <section class="firstsection">
      <!-- toppanel -->
      <div id="info" *ngFor="let branch of branches" >


      <div class="headdiv">
        <div class="hedarrows">
          <img src="assets/img/25.png">
          <p class="baclbutt" (click)="back()">Назад</p>
        </div>

        <div class="imgdivforhex"><p class="textname">{{branch?.name}}</p></div>
         <p class="textautoehex">Автор:<span> Салих</span></p>
       </div>
      <!-- toppanel -->
```

Продолжение приложения Б

```
      <!-- lline -->
       <p class="textimglasdt"><img src="assets/img/27.png"></p>
      <!-- lline -->

       <!-- informbox -->
       <div class="informationblockopen">
         <div><img [src]="branch.img"  style="width: 184px;height: auto;"><br>
           <button class="trybuttoncommen">Подписаться</button><button
class="trybuttoncommen1">2</button>
         </div>
         <div class=textinformationblok1>
           <p>{{branch?.text}}</p>
         </div>

       </div>
       <!-- informbox -->
       <div class="irels">
       <div class="starspresent">Оценка:<div><img src="assets/img/29.png"></div></div>
       </div>

       <div class="dopolnitelni1">
         <p>Дополнительно:</p>
       </div>
       <p class="butcenter"><button type="button" *ngIf="name"
(click)="open(content)">Кнопка для поста</button></p>
</div>
<!-- filter -->
   <div class="widthfilterbox">
            <button class="button01" (click)="drop()"><span style="margin-left:-
50px;">Фильтры</span> <img style="position: absolute; right:13px;bottom:-1px"
src="assets/img/15.png"></button>
            <div class="dropdown-menu1" id="dropdown-menu1"[ngStyle]="{'display':show ?
'block':'none'}">
            <div class="fitstone"><div><img src="assets/img/16.png"></div><span style="display:
flex;margin-left:-2px"><span>По</span> <span> рейтингу</span></span><div><label
style="margin-left:30px" class="container">
            <input type="checkbox" checked="checked">
            <span class="checkmark"></span>
            </label></div></div>
            <div class="fitstone"><div><img src="assets/img/17.png"></div><span style="display:
flex; margin-left:-2px"><span>По </span> <span> дате</span></span><div><label
style="margin-left:60px" class="container">
            <input type="checkbox" >
            <span class="checkmark"></span>
            </label></div></div>
            <div class="fitstone"><div><img src="assets/img/18.png"></div><span style="display:
flex;margin-left:-2px"><span>По </span> <span> просмотрам</span></span><div><label
style="margin-left:1px" class="container">
            <input type="checkbox" >
```

```
            <span class="checkmark"></span>
            </label></div></div>
            </div>
          </div>
 <!-- filter -->
 <p class="iemadw1231"><img  src="assets/img/30.png"></p>
  <div class="salihinfoblock1" *ngFor="let post of posts" [id]="post._id">
     <p class="boxnumberinfinity">Название: <span>{{post?.name}}</span></p>
       <p class="boxnumberinfinity1">Автор: <span>{{post?.owner}}</span></p>
        <p class="boxnumberinfinity3">Текст:<span>{{post?.text}}</span></p>
    </div>
  </section>

  <!-- popap -->
 <!--   <section class="secondsection">
    <form>
    <div class="textdov2000">
      <p>Название:</p>
<textarea class="twxtareanum2" type="text" maxlength="20" placeholder="Введите
название"></textarea>
    </div>
    <div class="dinintosecondsection">
      <p class="text51231">Текст:</p><textarea class="twxtareanum1"></textarea>

    </div>
    <p class="butcenter"><button>опубликовать пост</button></p>


</form>
   </section> -->
  <!-- popap -->
  <div class="imgbottomone"><img src="assets/img/13.png"></div>
</main>


 <footer class="footercontent">
    <div class="inputdiveonefoot">
      <p class="inputdiveonefoottext">О In-Gradient |  Наша политика безопасности и защиты
данных</p>
      <p class="maintextfoot">
    © Copyright 2019 by In-Gradient. Все права защищены <br><span style="float:right;
margin-top:3px">Site by SAlaG</span></p>
          </div>
 </footer>
</div>
<!-- Button trigger modal -->


<!-- Modal -->
```

Продолжение приложения Б

```
<ng-template #content let-modal >
  <button type="button" class="close" aria-label="Close" (click)="modal.dismiss('Cross click')">
     <span aria-hidden="true" style="float:right; margin-top:10px; margin-
right:15px">&times;</span>
  </button>
 <div class="modal-body">
   <section class="secondsection">
     <form>
     <div class="textdov2000">
       <p>Название:</p>
       <textarea class="twxtareanum2" type="text" maxlength="20" placeholder="Введите
название" [(ngModel)]="newPost.name" [ngModelOptions]="{standalone: true}"></textarea>
</div>
     <div class="dinintosecondsection">
       <p class="text51231">Текст:</p><textarea class="twxtareanum1"
[(ngModel)]="newPost.text" [ngModelOptions]="{standalone: true}"></textarea>
     </div>
     <label class="btn btn-default">
        <input #myInput type="file" class="imput" id="filein" (change)="selectFile($event)">
     </label>
     <p class="butcenter butcenter1"><button (click)="addPost(); modal.dismiss('Cross
click')">опубликовать пост</button></p>
</form>
</section>
</div>
</ng-template>
</body>
```

Page.component.ts

```
import { ViewChild,ElementRef,Component, OnInit, Renderer2 } from '@angular/core';
import { FormBuilder, FormGroup } from  '@angular/forms';
import { HttpClient } from '@angular/common/http';
import { ActivatedRoute } from "@angular/router";
import { Router, NavigationEnd } from "@angular/router";
import { trigger, state, style, animate, transition } from '@angular/animations';
import { NgbModal, ModalDismissReasons } from '@ng-bootstrap/ng-bootstrap';
import { PostService } from '../post.service';
import { AuthenticationService } from '../Auth.service';
// import { environment as env } from '../../img';
import * as $ from 'jquery';

@Component({
  selector: 'app-page',
  templateUrl: './page.component.html',
  styleUrls: ['./page.component.css'],
  animations: [
    trigger('fade', [
      state('in', style({ opacity: 1, transform: 'rotateX(0deg)' })),
```

```
      transition(':enter', [style({ opacity: 0, transform: 'rotateX(90deg)' }), animate('0.2s')]),
      transition(':leave', [style({ opacity: 1, transform: 'rotateX(0deg)' }), animate('0.2s')])
    ])
  ]
})
export class PageComponent implements OnInit {
show: boolean = false;
closeResult = '';
posts: any;
name: any;
newPost:{
  name:string,
  text:string
};
cur_tree:string;
cur_branch:string;
id_branch:string;
selectedFile: File = null;
branches: any;
  constructor(private router: Router, private route: ActivatedRoute, private el: ElementRef, private
modalService: NgbModal, private post: PostService, private auth: AuthenticationService) { }
  ngOnInit() {
     window.scrollTo(0,0);

    this.id_branch=this.route.snapshot.params['id'];
    this.post.getBranch({id:this.id_branch}).subscribe(tree => {
      console.log(tree);
      this.branches = tree;
      this.cur_tree = tree[0].name;
      this.cur_branch = tree[0].tree;
      }, (err) => {
        console.error(err);
      });
    this.getPosts()
    this.getName();
    this.newPost={
      name:'',
      text:''
    };
  }
drop(){
  this.show = !this.show;
}
open(content) {
  if(this.name){
    this.modalService.open(content, { windowClass: 'my-class'});
  }else{
    console.log('No logg in');
  }
```

```
back(){
  this.router.navigate(['main']);
}
getPosts(){
  this.post.getPosts({id: this.id_branch}).subscribe(posts => {
    console.log(posts);
    this.posts = posts;
    }, (err) => {
      console.error(err);
    });
}
selectFile(event) {

    this.selectedFile = event.target.files[0];

  }
addPost(){
  if(this.newPost.name&&this.newPost.text){
    console.log("err1");
  const fd = new FormData();
   fd.append('file', this.selectedFile);
   fd.append('name',this.newPost.name)
   fd.append('b_name',this.cur_branch)
   fd.append('t_name',this.cur_tree)
   fd.append('text',this.newPost.text)
   fd.append('owner',this.name)
   this.post.addPost(fd).subscribe(resp => {
         console.log(resp);
        }, err => {
         console.log(err);
        });
    setTimeout(() => { this.getPosts(); }, 500);
  }else{
    console.log("err");
  }
}

getName(){
  this.name=this.auth.getUserDetails();
  if(this.name==null){
    console.log("Not logg in")
}else{
    this.name=this.name.username;
  }
}
}
```

Auth.service.ts

Продолжение приложения Б

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { map } from 'rxjs/operators/map';
import { Router } from '@angular/router';
import { of } from 'rxjs/observable/of';

export interface UserDetails {
  _id: string;
  name: string;
}

interface TokenResponse {
  token: string;
}

export interface TokenPayload {
  password: string;
  name?: string;
}

@Injectable()
export class AuthenticationService {
  private token: string;

  constructor(private http: HttpClient, private router: Router) {}

  public saveToken(token: string): void {
    localStorage.setItem('mean-token', token);
    this.token = token;
  }

  private getToken(): string {
    if (!this.token) {
      this.token = localStorage.getItem('mean-token');
}
    return this.token;
  }

  public getUserDetails(): UserDetails {
     const token = this.getToken();
     console.log(localStorage.getItem('mean-token'));
     let payload;
     if (token) {
       payload = token.split('.')[1];
       payload = window.atob(payload);
       return JSON.parse(payload);
     } else {
       return null;
```

```
    }
  }
  public isLoggedIn(): boolean {
   const user = this.getUserDetails();
   if (user) {
     return true;
   } else {
     return false;
   }
  }
  public register(user) {
   return this.http.post('/api/signup', user).subscribe(resp => {
     console.log(resp);
     if(resp['success']){
       this.router.navigateByUrl('/main', {skipLocationChange: true}).then(() =>
       this.router.navigate(['signup']));
     }
   }, err => {
     // this.message = err.error.msg;
   });
  }

  public login(user){
   if(!this.isLoggedIn()){
   return this.http.post('/api/login',user);
   }else{
     const url = this.router.url;
     if(url == '/main'){
       this.router.navigateByUrl('/popup', {skipLocationChange: true}).then(() =>
  this.router.navigate([url]));
     }
   }
   }

  public profile(): Observable<any> {
   return this.http.get('/api/profile',{ headers: { Authorization: `Bearer ${this.getToken()}` }});
  }

  public logout(){
   this.token = '';
   window.localStorage.removeItem('mean-token');
   this.router.navigateByUrl('/')
  }
}
```

Post.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
```

Продолжение приложения Б

```
import { Observable } from 'rxjs/Observable';
import { Router } from '@angular/router';

@Injectable()
export class PostService {
  private token: string;

  constructor(private http: HttpClient, private router: Router) {}

  public getTree(): Observable<any> {
    return this.http.get('/api/tree',{});
  }
  public getBranch(tree): Observable<any> {
    return this.http.post('/api/branch', tree);
  }
  public getPosts(tree): Observable<any> {
    return this.http.post('/api/post', tree);
  }
  public addTree(tree): Observable<any> {
    return this.http.post('/api/addtree', tree);
  }
  public addBranch(branch): Observable<any> {
    console.log(branch);
    return this.http.post('/api/addbranch', branch);
  }
  public addPost(post): Observable<any> {
    console.log(post);
    return this.http.post('/api/addpost', post)
  }
}
```

Приложение В

Акт

О принятии к внедрению результатов дипломной работы Мамашева Салиха Ильдаровича, студента 4-го курса АУЭС на тему «Технологии разработки современных Web-приложений»

Настоящий акт свидетельствует, что предоставленный набор веб-технологий, а так же система защиты, разработанная Мамашевым С.И., внедрено в ТОО «Engineering & Security Company».

Процесс внедрения проходил с 10 по 20 мая 2019 г.

Предоставленный набор технологий и модулей представляет следущие приемущества:

-высокая скорость разработки;

-удобство при взаимодействии в команде разработчиков;

-высокая скорость работы сервера по сравнению с ранее использованными технологиями;

-удобство загрузки и взаимодействия пользователей с веб приложением.

Помимо этого была внедрена система защита для приложения, разработанного с помощью заданных технологий.

В ходе опытной эксплуатации программного обеспечения подтверждено, что оно обладает всеми заявленными преимуществами, а созданая система защиты прошла, все внутренние тесты и подтвердила свою надежность.

Директор: _____ Степанкевич А. Э.
Исполнитель: _____ Степанкевич А. Э.