МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
им. ГУМАРБЕКА ДАУКЕЕВА»
Кафедра IT – инжиниринг

«ДОПУЩЕН К ЗАЩИТЕ»
Зав. кафедрой PhD, доцент Досжанова А.А
_____ «____» _____2020 г.

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Разработка web-приложения для ТОО «ARLAN SI»
Специальность 5В060200 – Информатика
Выполнил: Курбанов Р.А Группа Инф-16-2
Научный руководитель: PhD, доцент Маликова Ф.У

Консультанты:
по экономической части: к.э.н., профессор Габелашвили К.Р
(учѐная степень, звание, Ф.И.О.)
_____ «____» _____2020 г.
по безопасности жизнедеятельности: ассистент Тыщенко Е.М
(учѐная степень, звание, Ф.И.О.)
_____ «____» _____2020 г.
по программному обеспечению: ст.преп. Майкотов М.Н
(учѐная степень, звание, Ф.И.О.)
_____ «____» _____2020 г.
Нормоконтролер: ст.преп. Абсатарова Б.Р
(учѐная степень, звание, Ф.И.О.)
_____ «____» _____2020 г.
Рецензент:_____
(учѐная степень, звание, Ф.И.О.)
_____ «____» _____2020 г.

Алматы 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
им. ГУМАРБЕКА ДАУКЕЕВА»

Институт систем управления и информационных технологий

Специальность 5В060200 – «Информатика»

Кафедра IT- инжиниринг

ЗАДАНИЕ
на выполнение дипломного проекта

Студенту Курбанову Расиму Анваровичу

Тема проекта: Разработка web-приложения для ТОО «ARLAN SI»

Утверждена приказом по университету № ___ от «___» _____2020 г.

Срок сдачи законченного проекта «___» _____ 2020 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): MongoDB-среда разработки БД, Node.js – среда программирования, Visual Studio Code – интегрированная среда разработки.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта:

а) аналитическая часть;
б) технологии разработки приложения;
в) функционал программного продукта;
г) экономическая эффективность проекта;
д) вопросы безопасности жизнедеятельности и охраны труда.

Перечень графического материала (с точным указанием обязательных чертежей): представлены 11 таблиц, 93 иллюстрации.

Основная рекомендуемая литература:

1 Бэнкер, К. MongoDB в действии / К. Бэнкер. - Москва: Высшая школа, 2016. - 287 с.

2 Мерсер Drupal 6. Создание надежных и полнофункциональных веб-сайтов,блогов, форумов, порталов и сайтов-сообществ / Мерсер, Дэвид. - М.: Вильямс, 2016. - 272 с.

3 Прамодкумар Дж. Садаладж, Фаулер Мартин NoSQL. Новая методология разработки нереляционных баз данных; Вильямс - М., 2015. - 192 с.

Консультация по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
Экономическая часть	Габелашвили К.Р.	24.04.2020	
Безопасности жизнедеятельности	Приходько Н.Г.	23.04.2020	
Специальная часть			
Программная часть	Майкотов М.Н.	15.05.2020	
Нормоконтролер	Абсатарова Б.Р.		

ГРАФИК
подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечания
Анализ предметной области	13.01.2020 - 20.01.2020	
Проектирование сайта	21.01.2020 - 15.02.2020	
Экспериментальная часть	16.02.2020 - 11.03.2020	
Экономическое обоснование	24.04.2020	
Безопасность жизнедеятельности	23.04.2020	

Дата выдачи задания «____» _____2020г.

Заведующий кафедрой _____А.А. Досжанова

Научный руководитель проекта _____Ф.У. Маликова

Задание принял к исполнению студент _____Р.А. Курбанов

Аңдатпа

Дипломдық жұмыстың тақырыбы: «ARLAN SI» ЖШС үшін веб-қосымшаны құру.

Дипломдық жұмыстың мақсаты - пайдаланушыларға телекоммуникациялық шешімдерді табуға мүмкіндік беретін бағдарламалық өнімді әзірлеу.

Осы мақсатқа жету үшін келесі технологиялар пайдаланылды:

- клиент интерфейсін дамыту: HTML, CSS, JavaScript;
- сервер жағын дамыту: Node.js, Express.js;
- дерекқордың дизайны: MongoDB ДББЖ.

Дипломдық жұмыс кіріспеден, 5 бөлімнен және қорытындыдан тұрады.

Кіріспе жобаның негізгі мақсатын ашады, веб-сайттың маңыздылығын сипаттайды. Келесі 3 тарауда сайттың дамуы сипатталған. 4-тарауда еңбекті санитарлық-гигиеналық бағалау сипатталады, сонымен қатар кондицияны есептейді. Бесінші тарауда әзірленіп жатқан жобаның техникалық-экономикалық негіздемесі жасалды.

Аннотация

Тема дипломного проекта: «Разработка web-приложения для ТОО «ARLAN SI».

Цель дипломного проекта – разработка программного продукта предоставляющего пользователям возможность найти решения по телекоммуникации.

Для выполнение поставленной цели использовались технологии:

- разработка клиентского интерфейса: HTML, CSS, JavaScript;
- разработка серверной части: Node.js, Express.js;
- проектирование базы данных: СУБД MongoDB.

Дипломный проект состоит из введения, 5-и глав и заключения.

Введение раскрывает основную цель проекта, описывает актуальность веб-сайта. В последующих 3-х главах следует описание разработку сайта. В 4 главе описывается санитарно-гигиеническая оценка труда, а также производятся расчеты кондиционирования. В пятой главе было произведено технико-экономическое обоснования целесообразности разрабатываемого проекта.

Annotation

Theme of the graduation project: “Web-application development for «ARLAN SI» LLP.

The aim of the graduation project is to develop a software product that provides users with the opportunity to find telecommunication solutions.

To achieve this goal, the following technologies were used:

- development of the client interface: HTML, CSS, JavaScript;
- development of the server side: Node.js, Express.js;
- database design: DBMS MongoDB.

The graduation project consists of introduction, 5 chapters and conclusion.

The introduction reveals the main goal of the project, describes the relevance of the website. The following 3 chapters describe the development of the site. Chapter 4 describes the sanitary-hygienic assessment of labor, and also calculates conditioning. In the fifth chapter, a feasibility study was made on the feasibility of the project being developed.

Содержание

Введение

В настоящее время интернет развивается с невообразимой скоростью, открывает новые возможности в разных сферах деятельности. Существует множество компаний в сфере телекоммуникаций и одна из лидирующих компаний в Казахстане является ТОО «ARLAN SI», которая занимается системным интегрированием, а также поставкой телекоммуникационного оборудования, коротко говоря, компания занимается развитием телекоммуникаций в нашей стране. Для развития и продвижения компании, держать лидирующее место в сфере телекоммуникаций становится насущной необходимостью разработка сайта. На сегодняшние дни сайт выступает средством коммуникации привлечения клиентов. Огромное количество компаний во всем миру видят в Интернете большой коммерческий потенциал и возможность перевода своего бизнеса, своей деятельности на совершено новый уровень. Большую часть Интернета составляют сайты, рекламирующие компании занимающиеся разного рода деятельности.

Цель дипломной работы является создания web-приложения, предоставляющее размещение и рекламирование услуг в сфере телекоммуникаций компании ТОО «ARLAN SI». Для достижения данной цели необходимо было создать сайт, отвечающий следующим современным требованиям:

- удобство навигации по сайту;
- дизайн отвечающий современным требованиям;
- дружественный интерфейс.

Достигнутой целью стал сайт для ТОО «ARLAN SI», представляющий собой страницы логически взаимодействующих между друг другом. Главный принцип дизайна, созданного в работе, стал принцип удобства и простоты использования.

Основные средства, использованные при создании сайта:

- Visual Studio Code – редактор исходного кода.

1 Аналитическая часть

1.1 IT решения для компаний в сфере телекоммуникаций

В настоящее время Агентством информатизации и связи проводятся мероприятия по развитию Государственного регистра информационно-телекоммуникационных ресурсов, созданию Депозитария программных кодов и документации информационных ресурсов, формированию системы экспертизы и сертификации программ, а также по сопровождению справочника и сайта официальных электронных адресов.

На сегодняшний день в республике приняты законы “Об информатизации” и “Об электронном документе и электронной цифровой подписи”, а также ряд постановлений правительства республики, регулирующих отношения в сфере информатизации. Наряду с этим Агентством разрабатываются нормативные акты, касающиеся применения электронного документооборота и электронной цифровой подписи, а также регулирующие отношения в области развития национального сегмента “kz” глобальной сети интернет.

Сотовая связь является одним из наиболее динамично развивающихся сегментов рынка телекоммуникаций. Первую лицензию на осуществление в республике услуг мобильной связи получила в 1994 году компания “Алтел”, работавшая в стандарте AMPS [11]. Однако к моменту, когда в 1999 году на рынок вышли еще два оператора с цифровым стандартом GSM, в списке абонентов компании значилось только 50 тысяч человек. По прошествии четырех лет, к началу 2004 года количество пользователей сотовой связи уже составляло около 1,5 млн. человек. Весьма показательно, что в 2002 году этот показатель был равен только 900 тысячам.

Для настройки тарифов и вывода личного кабинета IT решения предоставляют мобильные веб-приложения, которые комфорты в использовании и не требуют выполнения множества действия для работы с тарифным планом сотовой связи.

Все больше появляются компании предоставляющие услуги в телекоммуникации, именно поэтому каждая компания должна быть конкурентно способной. Для улучшения конкурентно способности IT решения предлагают веб-сайты, мобильные приложения и т.п.

В время информационных технологий каждая компания должна иметь свой собственный веб-сайт. Именно благодаря веб-сайту компания может предоставлять свои услуги онлайн, находить новых бизнес-клиентов, держать в курсе новостей о продвижении компании, об запланированных мероприятий, встреч.

Помимо того, что у компании имеется сайт, необходимо чтобы он был современным, удовлетворял тренду новых технологий, имел легкую и понятную навигацию по сайту, адаптивную верстку.

Телекоммуникации никогда не стоят на месте, каждый день создаются новые возможности, функции, технологии, компании следят за всем новым и стараются внедрять новые технологии в ногу со временем.

1.2 Постановка задач

Целью данного дипломного проекта является описание создания веб-сайта для компании «ARLAN SI»:

- увеличение притока клиентов благодаря современному сайту;
- улучшение конкурентно способности;
- реализация авторизации пользователей;
- создание комментариев к услуге компании;
- установка прав администратора;
- возможность создания, редактирования, удаления услуги;
- реализация формы обратной связи;
- лэндиг страница;
- создание user-friendly интерфейса;
- создание комфортной навигации по сайту.

2 Архитектура Web программы

2.1 Этапы проектирования web-приложения

Несмотря на общепринятые взгляды, основная часть разработки и ддизайна веб-сайтов не является необходимой для процесса кодирования. В самом деле, такие технологии, как HTML, CSS и JavaScript, придают веб-сайту свою форму и определяют способ взаимодействия с информацией, а серверная которая состоит из Node.js и фрейморка для нода Express.js, которые отвечают за динамичность нашего сайта. Но то, что обычно остается за кадром и в то же время остается важной частью жизненного цикла разработки веб-сайтов, - это этапы предварительного сбора информации, детального планирования и обслуживания после запуска.

Общее количество этапов разработки обычно варьируется от пяти до восьми, но каждый раз, когда картина в целом остается одинаковой.

2.1.1 Сбор информации: цель, основные цели и целевая аудитория

Данный этап сбора информации, определяет, как будут выглядеть последующие шаги. Наиболее важной задачей на данный момент является получение четкого понимания наших будущих целей веб-сайта, основных целей, которые мы хотим получить, и целевой аудитории, которую мы хотим привлечь на свой сайт (рисунок 2.1.1.1). Такая анкета для разработки веб-сайтов помогает разработать лучшую стратегию дальнейшего управления проектами.

Новостной портал отличается от развлекательных сайтов, а онлайн-ресурсы для подростков выглядят иначе, чем сайты для взрослых. Различные типы веб-сайтов предоставляют посетителям различную функциональность, что означает, что разные технологии должны использоваться в соответствии с целями. Тщательно описанный и подробный план, основанный на этих данных перед разработкой, может защитить вас от расходования дополнительных ресурсов на решение непредвиденных проблем, таких как изменение дизайна или добавление функциональности, которая изначально не планировалась.

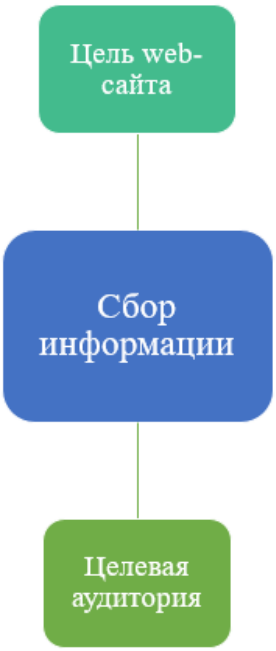

Рисунок 2.1.1.1 – Сбор информации

2.1.2 Планирование: карта сайта и создание каркаса

На этом этапе цикла разработки сайта разработчик создает данные, которые позволяют клиенту судить, как будет выглядеть весь сайт.

На основе информации, собранной на предыдущем этапе, создается карта сайта (рисунок 2.1.2.1).

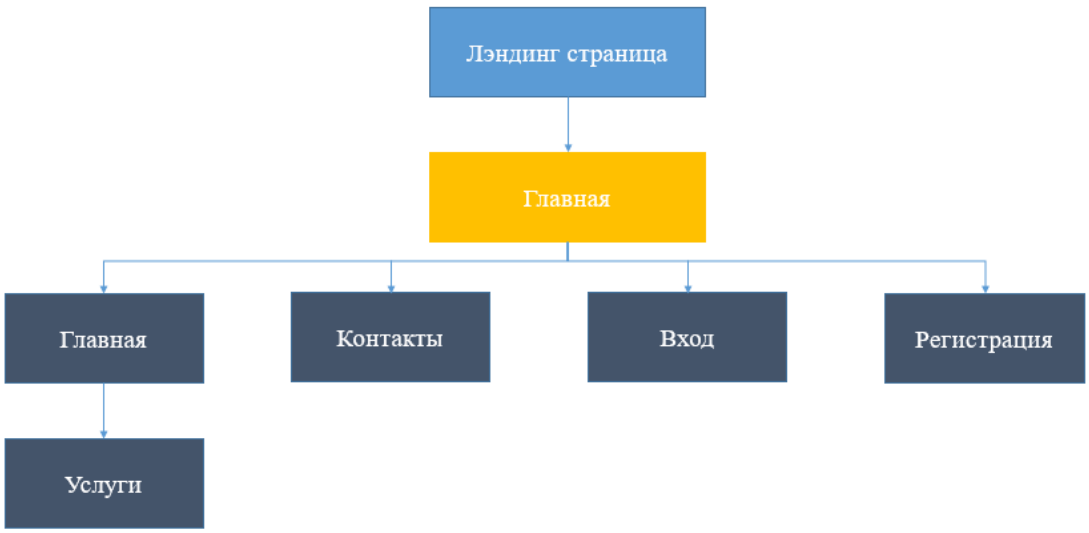

Рисунок 2.1.2.1 – Карта сайта ТОО «ARLAN SI»

Карта сайта должна описывать отношения между основными областями вашего сайта. Такое представление может помочь понять, насколько полезным будет конечный продукт. Он может показать вам «взаимосвязь» между различными страницами веб-сайта, поэтому вы можете судить, насколько легко конечному пользователю будет найти необходимую

информацию или услугу, если он начнет с главной страницы. Основной причиной создания карты сайта является создание удобного и простого в использовании веб-сайта.

Карта сайта позволяет понять, как выглядит внутренняя структура сайта, но не описывает пользовательский интерфейс. Иногда, прежде чем вы начнете кодировать или даже работать над дизайном, необходимо получить от клиента подтверждение, что все выглядит хорошо, чтобы вы могли перейти к следующему этапу разработки. В этом случае создается каркас или макет. Каркас – это визуальное представление пользовательского интерфейса, который вы собираетесь создать. Но он не содержит никаких элементов дизайна, таких как цвета, логотипы и т. д. Он описывает только элементы, которые будут добавлены на страницу, и их расположение. Это бесхитростный и дешевый в производстве эскиз.

2.1.3 Дизайн

На этапе дизайна сайт принимает форму. Создается весь визуальный контент, например, изображения, фотографии и видео. Вся информация, которая была собрана на первом этапе, имеет решающее значение. При работе над дизайном необходимо учитывать клиента и целевую аудиторию.

Макет сайта является результатом работы дизайнера. Это может быть графический эскиз или реальный графический дизайн. Основная функция макета заключается в представлении информационной структуры, визуализации контента и демонстрации основных функций. Макеты содержат цвета, логотипы, изображения и могут дать общее представление о будущем продукте.

Рисунок 2.1.3.1 – Проектирование дизайна

2.1.4 Разработка функциональной части

На этом этапе вы можете, наконец, начать создавать сам сайт. Графические элементы, которые были разработаны на предыдущих этапах, должны использоваться для создания реального веб-сайта. Обычно сначала создается домашняя страница, а затем добавляются все под страницы в соответствии с иерархией веб-сайта, которая ранее была создана в форме карты сайта.

Все статические элементы веб-страницы, которые были разработаны во время создания макета, должны быть созданы и протестированы. Затем следует добавить специальные функции и интерактивность. Глубокое понимание каждой технологии разработки веб-сайтов, которую вы собираетесь использовать, имеет решающее значение на этом этапе.

Рисунок 2.1.4.1 – Разработка функционала сайта

2.1.5 Тестирование, проверка и запуск

Тестирование, вероятно, самая рутинная часть процесса. Каждая ссылка должна быть проверена, чтобы убедиться, что среди них нет сломанных. Вы должны проверить каждую форму, каждый скрипт, запустить программу для проверки орфографии, чтобы найти возможные опечатки. Используйте средства проверки кода, чтобы проверить, соответствует ли ваш код текущим веб-стандартам. Действительный код необходим, например, если для вас критически важна совместимость с разными браузерами.

После того, как вы проверите и перепроверите свой веб-сайт, пришло время загрузить его на сервер. Для этой цели используется программное обеспечение FTP. После того, как вы развернули файлы, вы должны запустить еще один финальный тест, чтобы убедиться, что все ваши файлы установлены правильно.

2.1.6 Реализация

На этом этапе проект должен заполняться реальным контентом, он устанавливается на реальных серверах, и все внутренние системы переводятся из тестового режима в живой. Ваш проект становится доступным для всех по всему миру.

2.1.7 Поддержка и обслуживание

Важно помнить, что веб-сайт - это скорее услуга, чем продукт. Недостаточно «доставить» сайт пользователю. Вы также должны убедиться, что все работает нормально, все довольны и всегда готовы внести изменения в другом случае.

Система обратной связи, добавленная на сайт, позволит вам обнаружить возможные проблемы, с которыми сталкиваются конечные пользователи. В данном случае приоритетной задачей является устранение проблемы как можно быстрее. Если вы этого не сделаете, однажды ваши пользователи предпочтут использовать другой веб-сайт, а не мириться с неудобствами.

Другой важной вещью является поддержание вашего сайта в актуальном состоянии.

Рисунок 2.1.7.1 – Поддержка и обслуживание сайта

2.2 Обоснования выбора инструментов разработки

На сегодняшний в день web-разработки существует множество языков с различными структурами, обширными библиотеками и функциями для решения определенных задач.

Два ключевых слова, чтобы понять, как работают веб-сайты – это внешний интерфейс и внутренний интерфейс или если говорить языком программистов Front-end и Back-end.

2.2.1 Front-End (Внешний интерфейс)

Front-End, также называемый программированием клиентской части - это то, что происходит в браузере. Это все, что видит и с чем взаимодействует пользователь.

Часть веб-сайта, с которой пользователь взаимодействует напрямую, называется клиентской частью. Он также называется «клиентской стороной» приложения. Он включает в себя все, что непосредственно воспринимается

пользователями: цвета и стили текста, изображения, графики и таблицы, кнопки, цвета и меню навигации. HTML, CSS и JavaScript - это языки, используемые для разработки Front-End. Структура, дизайн, поведение и содержание всего, что видно на экране браузера при открытии веб-сайтов, веб-приложений или мобильных приложений, реализуется разработчиками переднего плана. Отзывчивость и производительность - две основные цели переднего конца. Разработчик должен убедиться, что сайт реагирует, т. Е. Он правильно отображается на устройствах любого размера, ни одна из частей сайта не должна вести себя ненормально независимо от размера экрана.

Важно отметить, что разработка интерфейсной части значительно изменилась за последние 10–15 лет благодаря взрывному росту JavaScript, который не был столь распространенным на переднем конце, как сейчас, или даже не столь распространенным на внутреннем.

HTML расшифровывается как Hyper Text Markup Language. Он используется для разработки внешнего интерфейса веб-страниц с использованием языка разметки. При создании веб-сайта невозможно обойтись без HTML, который играет роль скелета нашего сайта.

Каскадные таблицы стилей, или же просто CSS, - это разработанный язык, предназначенный для упрощения процесса придания веб-страниц презентабельности. CSS позволяет применять стили к веб-страницам, будь то шрифт текста, цвет, различная анимация, указания размеров полей, отступов и т.д.

JavaScript отвечает за всю логику сайта, используется для интерактивности элементов, как navbar (выпадающее меню), блоки информации.

Большинство сайтов написаны только с использованием Front-End части, то есть клиентской, так как для компаний, которые хотят предоставить информацию о себе, этого вполне достаточно. Если компания не собирается организовывать систему покупок, авторизацию, то нет необходимости подключать серверную часть и базу данных, такие сайты называются статическими.

2.2.2 Back-End (Внутренний интерфейс)

Back-End является серверной частью сайта. Он хранит и упорядочивает данные, а также следит за тем, чтобы все на стороне клиента сайта работало нормально. Это та часть сайта, которую вы не можете видеть и взаимодействовать с ней. Это часть программного обеспечения, которая не вступает в прямой контакт с пользователями. Части и характеристики, разработанные дизайнерами бэкэнда, доступны пользователям через интерфейсное приложение. Действия, такие как написание API, создание библиотек и работа с компонентами системы без пользовательских интерфейсов или даже систем научного программирования, также включены в бэкэнд.

Back-End создается с использованием некоторых языков, которые описаны ниже:

1) PHP – это серверный язык сценариев, разработанный специально для веб-разработки. Поскольку PHP-код выполняется на стороне сервера, он называется серверным языком сценариев.

2) C ++ – это язык программирования общего назначения, который в наши дни широко используется для конкурентного программирования. Он также используется в качестве языка бэкэнда.

3) Java является одним из самых популярных и широко используемых языков программирования и платформ.

4) Python – это язык программирования, который позволяет вам работать быстрее и более эффективно интегрировать системы.

5) JavaScript может использоваться как в качестве клиентской, так и серверной частях web-приложения.

6) Node.js – это кроссплатформенная среда выполнения с открытым исходным кодом для выполнения кода JavaScript вне браузера. Node.js – это не фреймворк и не язык программирования. Node.js часто применяют для создания внутренних сервисов, таких как API, таких как web-приложение или мобильное приложение. Он используется в производстве крупными компаниями, такими как Paypal, Uber, Netflix, Wallmart и так далее.

Таблица 2.2.2.1 – Сравнительная таблица Front-End и Back-End

Front-End	**Back-End**
Состоит из всего, что связано с визуальными и пользовательскими аспектами ввода веб-сайта	Состоит из веб-сервера, который имеет соединение с базой данных для ответа на запросы, предоставляемые Front-End системой.
Клиентская часть сайта или приложения	Серверная часть сайта или приложения
Собирает данные ввода пользователя	Обрабатывает пользовательский ввод
Графический интерфейс пользователя (GUI), благодаря которому пользователи могут пользоваться различными услугами в сети.	Предположим, что это мозг сайта, из-за которого он работает эффективно.
Отвечает за доступность, поисковую оптимизацию	Отвечает за безопасность
Веб-языки используются для внешнего интерфейса, такого как HTML, CSS, Javascript	Для серверной части используются такие языки программирования как Python, Ruby, Node.js, PHP, .Net и т.д.

2.2.3 Выбор IDE

Редактирование HTML и CSS кода может быть сделано без каких-либо специальных инструментов. Писать код можно и в простых текстовых редакторах как «блокнот». Однако использовать тот же «блокнот» для разработки веб-приложения не лучший вариант.

Сегодня можно легко найти отличную и бесплатную среду разработки и без труда разрабатывать приложения. Если вам нужна JavaScript IDE, HTML IDE или любая другая IDE для веб-разработки, все они существуют, и многие из них имеют открытый исходный код.

Вкратце, что из себя представляет IDE – программное обеспечение, которое вы можете загрузить на свой компьютер. IDE предназначена для упрощения процесса веб-разработки, как упоминалось ранее.

В мире существует множество сред для разработки приложений таких как: IntelliJ IDEA, Visual Studio, Visual Studio Code, WebStorm, NetBeans и др.

Для нашего web-приложения отлично подойдет программа Visual Studio Code, который находится на первой строчке по полярности среди web-разработчиков.

Visual Studio Code или же VS Code не только поддерживает JavaScript, но также поддерживает Node.js, TypeScript и поставляется с целой экосистемой расширений для других языков, включая C ++, C #, Python, PHP и т.д. Он обеспечивает отличную подсветку синтаксиса и автоматическое заполнение с помощью IntelliSense на основе типов переменных, определений функций и импортированных модулей. Он также позволяет вам отлаживать код, запуская или присоединяя к отлаженным приложениям с помощью точек останова, стеков вызовов и интерактивной консоли. Вы можете легко интегрировать библиотеку пользовательского интерфейса JavaScript в код Visual Studio. В общем, эта IDE для JavaScript, безусловно, один, чтобы проверить. Немало важно, то что его можно использовать совершенно бесплатно. Также стоит отметить, что в VS Code предоставляет возможность интеграции с GitHub для работы с Git (контроль версий) не выходя из самой IDE.

Для установки Visual Studio Code необходимо перейти на официальный сайт «Visual Studio Code» (рисунок 2.2.3.1) или перейти по данной ссылке https://code.visualstudio.com/docs/setup/windows для скачивания версии под Windows, так как я использую операционную систему Windows, после чего нажать на кнопку установки.

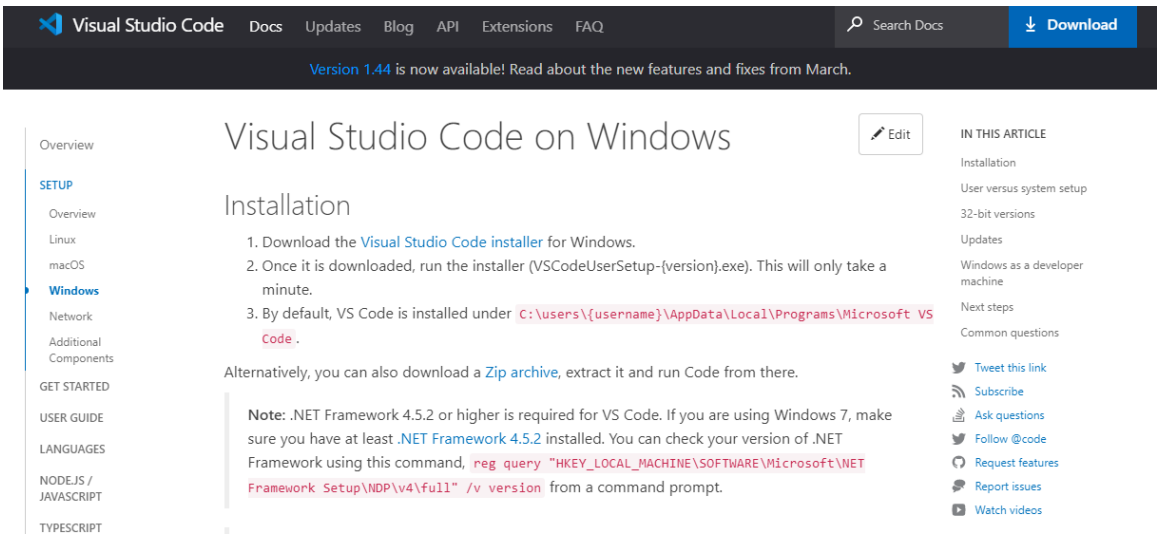

Рисунок 2.2.3.1 – Сайт для установки VS Code

После того как загрузка завершится, необходимо запустить установщик самой программы (VSCodeUserSetup-{version}.exe). Установка займет не больше минуты.

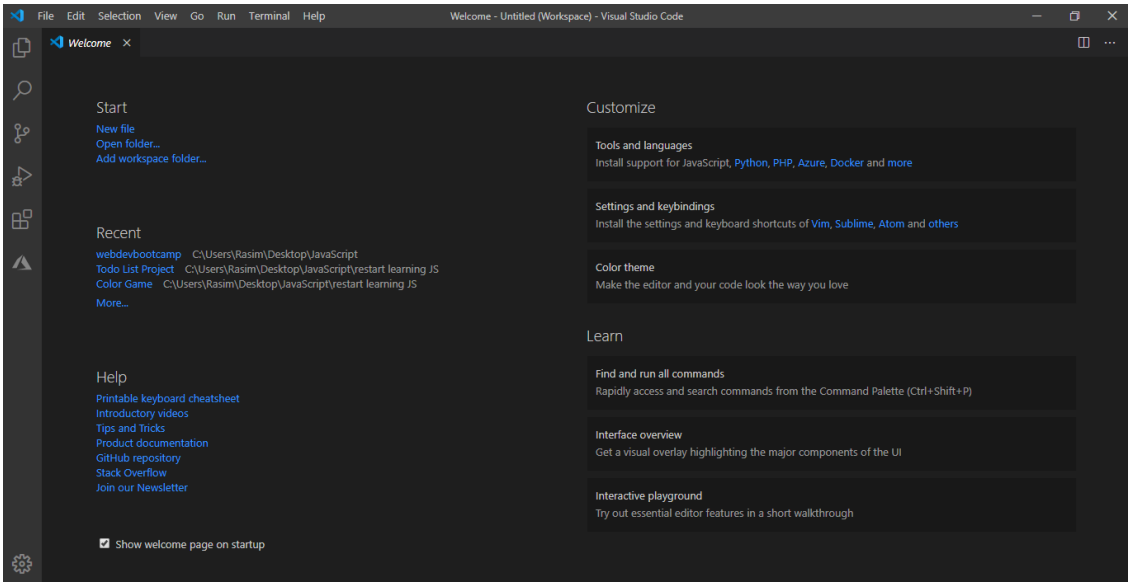

Рисунок 2.2.3.2 – Окно приветствия VS Code

2.2.4 Язык гипертекстовой разметки HTML

HTML (Hypertext Markup Language) - это текстовый подход к описанию структуры контента, содержащегося в файле HTML. Эта разметка сообщает веб-браузеру, как отображать текст, изображения и другие виды мультимедиа на веб-странице.

HTML является официальной рекомендацией Консорциума World Wide Web (W3C) и, как правило, соблюдается всеми основными веб-браузерами,

включая настольные и мобильные веб-браузеры. HTML5 - это последняя версия спецификации.

HTML - это текстовый файл, содержащий определенный синтаксис, соглашения о присвоении имен файлам и именам, которые показывают компьютеру и веб-серверу, что он находится в HTML и должны читаться как таковые. Применяя эти условные обозначения HTML к текстовому файлу практически в любом текстовом редакторе, пользователь может написать и создать базовую веб-страницу, а затем загрузить ее в Интернет.

Основным условием HTML является включение декларации типа документа в начало текстового файла. Это всегда на первом месте в документе, потому что это кусок, который утвердительно сообщает компьютеру, что это HTML-файл. Заголовок документа обычно выглядит так: <! DOCTYPE html>. Он всегда должен быть написан таким образом, без какого-либо содержания внутри или разбивки. Любой контент, представленный до этого объявления, не будет распознаваться компьютером как HTML [1].

Типы документов используются не только для HTML, они могут применяться для создания любого документа, использующего SGML (стандартный обобщенный язык разметки). SGML - это стандарт для определения используемого языка разметки. HTML является одним из нескольких языков разметки, к которым применяются объявления SGML и doctype.

Другим важным требованием для создания файла HTML является сохранение его с расширением .html. В то время как объявление doctype сигнализирует HTML на компьютер изнутри файла, расширение файла сигнализирует HTML на компьютер из-за пределов файла. Имея и то, и другое, компьютер может сказать, что это файл HTML, независимо от того, читает он файл или нет. Это становится особенно важным при загрузке файлов в Интернет, поскольку веб-сервер должен знать, что делать с файлами, прежде чем он сможет отправить их на клиентский компьютер для считывания внутреннего содержимого [1].

После написания документа и сохранения его в виде файла HTML пользователь может реализовать все другие синтаксические инструменты HTML для настройки веб-страницы. После завершения они, вероятно, будут иметь несколько файлов HTML, соответствующих различным страницам сайта. Важно, чтобы пользователь загружал эти файлы в той же иерархии, в которой они их сохранили, поскольку каждая страница ссылается на конкретные пути к файлам на других страницах, позволяя создавать ссылки между ними. Загрузка их в другом порядке приведет к разрыву ссылок и потере страниц, поскольку указанные пути к файлам не будут соответствовать страницам.

Используя HTML, текстовый файл дополнительно размечается дополнительным текстом, описывающим, как должен отображаться документ. Чтобы отделить разметку от фактического содержимого файла

HTML, используется специальный отличительный синтаксис HTML. Эти специальные компоненты известны как теги HTML. Теги могут содержать пары имя-значение, известные как атрибуты, а часть содержимого, заключенная в тег, называется элементом HTML.

Элементы HTML всегда имеют открывающие теги, содержимое в середине и закрывающие теги. Атрибуты могут предоставить дополнительную информацию об элементе и включены в открывающий тег. Элементы могут быть описаны одним из двух способов:

- элементы уровня блока начинаются с новой строки в документе и занимают свое собственное место. Примеры этих элементов включают заголовки и теги абзаца;
- встроенные элементы не начинаются с новой строки в документе и занимают только необходимое место. Эти элементы обычно форматируют содержимое элементов уровня блока. Примеры встроенных элементов включают гиперссылки и теги текстового формата.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    Hello World!
</body>
</html>
```

Рисунок 2.2.4.1 – Структура HTML документа

В первые дни всемирной паутины разметка текстовых документов с использованием синтаксиса HTML была более чем достаточной для облегчения обмена академическими документами и техническими заметками. Однако по мере того, как Интернет выходил за рамки академических кругов и становился домом для широких слоев населения, на веб-страницы предъявлялся повышенный спрос с точки зрения форматирования и интерактивности.

HTML 4.01 был выпущен в 1999 году, когда Интернет еще не был нарицательным, а HTML5 не был стандартизирован до 2014 года. За это время разметка HTML перешла от простого описания структуры документа содержимого веб-страницы до роли. также описания того, как должен выглядеть контент, когда веб-страница отображает его [1].

В результате веб-страницы на основе HTML4 часто включают в тег информацию о том, какой шрифт использовать при отображении текста, какой цвет следует использовать для фона и как выравнивать содержимое. Описание в HTML-теге того, как должен быть отформатирован HTML-

элемент при отображении на веб-странице, считается антипаттерном HTML. Как правило, HTML должен описывать структуру контента, а не его стилизацию и отображение в браузере. Другие языки разметки лучше подходят для этой задачи.

Одно из основных различий между HTML4 и HTML5 заключается в том, что шаблон разделения интересов более строго соблюдается в HTML5, чем в HTML4. В HTML5 теги <b> и курсив <i> выделены жирным шрифтом. Для тега абзаца атрибут align полностью удален из спецификации HTML.

Одной из наиболее ожидаемых особенностей HTML5 является встроенная поддержка встраивания аудио и видео. Вместо использования Flash Player мы можем просто вставлять видео и аудио файлы на наши веб-страницы, используя новые теги <audio> </ audio> и <video> </ video>. Он также включает встроенную поддержку масштабируемой векторной графики (SVG) и MathML для математических и научных формул.

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>audio</title>
 </head>
 <body>
  <audio controls>
    <source src="audio/music.ogg" type="audio/ogg; codecs=vorbis">
    <source src="audio/music.mp3" type="audio/mpeg">
  </audio>
 </body>
</html>
```

Рисунок 2.2.4.2 – Пример использования тега <audio>

HTML5 также внес несколько семантических улучшений. Новые семантические теги информируют браузеры о значении контента, что выгодно как читателям, так и поисковым системам.

2.2.5 Каскадные таблицы стилей CSS

CSS был впервые разработан в 1997 году как способ для веб-разработчиков определить внешний вид создаваемых ими веб-страниц. Он был предназначен для того, чтобы позволить веб-профессионалам отделить содержание и структуру кода веб-сайта от визуального дизайна, что было невозможно до этого времени [1].

Разделение структуры и стиля позволяет HTML выполнять больше функций, на которых он изначально основывался – разметку контента, не беспокоясь о дизайне и макете самой страницы, что обычно называют «внешним видом» страницы.

CSS не завоевывал популярность до 2000 года, когда веб-браузеры начали использовать не только основные шрифтовые и цветовые аспекты

этого языка разметки. Сегодня все современные браузеры поддерживают все уровни CSS 1, большую часть уровня CSS 2 и даже большинство аспектов уровня 3. Поскольку CSS продолжает развиваться и появляются новые стили, веб-браузеры начали реализовывать модули, которые обеспечивают новую поддержку CSS.

Слово «таблица стилей» относится к самому документу (подобно HTML, файлы CSS на самом деле являются просто текстовыми документами, которые можно редактировать с помощью различных программ). Таблицы стилей использовались для оформления документов в течение многих лет. Это технические характеристики макета, будь то печать или онлайн. Дизайнеры печати давно используют таблицы стилей, чтобы гарантировать, что их дизайны будут напечатаны точно по их спецификациям. Таблица стилей для веб-страницы служит той же цели, но с добавленной функциональностью также сообщает веб-браузеру, как визуализировать просматриваемый документ. Сегодня таблицы стилей CSS также могут использовать медиазапросы для изменения внешнего вида страницы для различных устройств и размеров экрана. Это невероятно важно, поскольку позволяет отображать один HTML-документ по-разному в зависимости от экрана, используемого для доступа к нему [1].

Каскад - это действительно особенная часть термина «каскадная таблица стилей». Веб-таблица стилей предназначена для каскадного прохождения ряда стилей на этом листе, как река над водопадом. Вода в реке поражает все камни в водопаде, но только те, которые находятся внизу, точно определяют, куда будет течь вода. То же самое относится и к каскаду в таблицах стилей сайта.

На каждую веб-страницу влияет хотя бы одна таблица стилей, даже если веб-дизайнер не применяет никаких стилей. Эта таблица стилей является таблицей стилей агента пользователя - также известной как стили по умолчанию, которые веб-браузер будет использовать для отображения страницы, если не предоставлено никаких других инструкций. Например, по умолчанию гиперссылки выделены синим цветом и подчеркнуты. Эти стили взяты из таблицы стилей веб-браузера по умолчанию. Однако, если веб-дизайнер предоставляет другие инструкции, браузер должен знать, какие инструкции имеют приоритет. Все браузеры имеют свои собственные стили по умолчанию, но многие из этих значений по умолчанию (например, текстовые ссылки, выделенные синим цветом) доступны для всех или для большинства основных браузеров и версий.

В качестве другого примера настройки браузера по умолчанию в нашем веб-браузере используется шрифт по умолчанию «Times New Roman», отображаемый в размере 16. Однако почти ни одна из страниц, которые мы посещаем, не отображается в этом семействе шрифтов и размере. Это потому, что каскад определяет, что вторые таблицы стилей, которые устанавливаются самими дизайнерами, переопределяют размер шрифта и семейство, переопределяя настройки по умолчанию нашего веб-браузера.

Любые таблицы стилей, которые вы создаете для веб-страницы, будут более специфичными, чем стили браузера по умолчанию, поэтому эти значения по умолчанию будут применяться только в том случае, если ваша таблица стилей не переопределит их. Если вы хотите, чтобы ссылки были голубыми и подчеркнутыми, вам не нужно ничего делать, так как это значение по умолчанию, но если в файле CSS вашего сайта написано, что ссылки должны быть зелеными, этот цвет заменит синий по умолчанию. Подчеркивание останется в этом примере, так как вы не указали иначе.

CSS также можно использовать для определения того, как должны выглядеть веб-страницы при просмотре на других носителях, кроме веб-браузера. Например, вы можете создать таблицу стилей печати, которая будет определять, как должна распечатываться веб-страница. Поскольку элементы веб-страницы, такие как кнопки навигации или веб-формы, не будут иметь смысла на напечатанной странице, можно использовать Таблицу стилей для печати, чтобы «отключить» эти области при печати страницы. Хотя это не совсем обычная практика на многих сайтах, возможность создания таблиц стилей печати является мощной и привлекательной [1].

CSS код подключается или же внедряется в HTML контент тремя способами, такими как внешний, внутренний и встроенный.

Чтобы использовать внешний способ, ваши .html-файлы должны включать заголовочный раздел, который ссылается на внешнюю таблицу стилей и выглядит как на рисунке 2.2.5.1:

Рисунок 2.2.5.1 – Внешний способ внедрения CSS кода

На рисунке 2.2.5.1 показано как внутри тега <head> есть тег <link>, в котором указывается путь к CSS файлу. После этого все стили CSS будут применены к html странице.

Внутренние способ – это код CSS, записанный непосредственно в заголовок конкретной страницы .html (рисунок 2.2.5.2).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <style>
      body {
        background-color: rgb(207, 202, 207);
      }
      p {
        font-size: 20px;
        color: rgb(65, 205, 0);
      }
    </style>
  </head>
```

Рисунок 2.2.5.2 – Внутренний способ внедрения CSS кода

На рисунке мы видим, что к тегу <body> применяется цвет фона указанный в CSS коде и к тегу <p> применяется размер шрифта в 20 px и цвет текста указанный в коде.

Встроенные стили - это фрагменты CSS, записанные непосредственно в код HTML, и применимые только к одному экземпляру кода (рисунок 2.2.5.3).

```
<body>
  <h1 style="font-size: 40px; color: rgb(197, 238, 130);">Hello World!</h1>
</body>
```

Рисунок 2.2.5.3 – Встроенный способ внедрения CSS кода

Код встроенного способа на рисунке 2.2.5.3 приведет к тому, что конкретный заголовок (h1), на одной странице .html, будет отображаться указанным цветом и шрифтом в 40 точек.

Ниже на рисунке 2.8 продемонстрирована часть кода файла main.css, который является основным файлом CSS нашего проекта, в котором описаны стили всех кнопок, различных форм, указаны шрифты от Google, картинка заднего фона сайта и др.

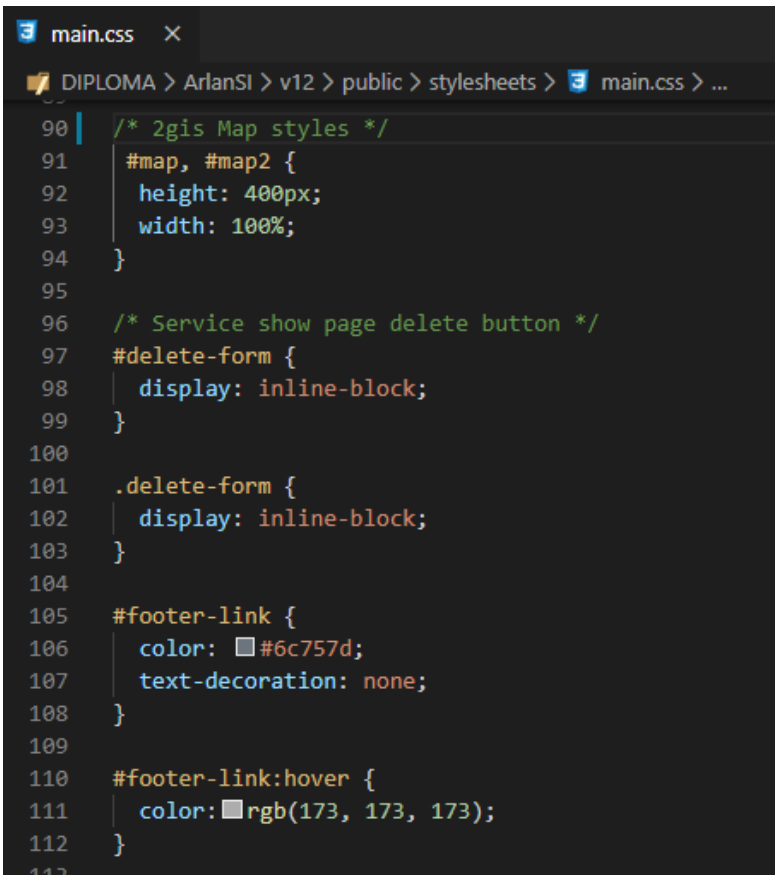

Рисунок 2.2.5.4 – Структура файла main.css

2.2.6 Сценарный язык программирования JavaScript

JavaScript (JS) – это интерпретируемый язык программирования высокого уровня, который также считается динамическим, слабо типизированным, основанным на прототипах и мультипарадигмальным. Наряду с языком гипертекстовой разметки (HTML) и каскадными таблицами стилей (CSS), это одна из трех основных технологий Всемирной паутины (WWW) [2]. Его характеристики позволяют создавать динамические веб-страницы, которые могут быть интерактивными с пользователями. Его можно найти на большинстве современных веб-сайтов, в то время как все современные браузеры поддерживают его без использования какого-либо плагина с помощью встроенного движка JavaScript. Каждый JS Engine представляет отдельную реализацию JS и все основаны на спецификации ECMAScript, при этом некоторые не поддерживают спецификацию полностью, в то время как многие другие поддерживают ее полностью, а другие функции за ее пределами.

Из-за своей мультипарадигмальной характеристики JS поддерживает управляемые событиями, функциональные и императивные (в том числе объектно-ориентированные и основанные на прототипах) стили программирования; делая это довольно разнообразным. Его API способен работать с текстом, массивами, датами, регулярными выражениями и базовыми манипуляциями с объектной моделью документа (DOM); однако

он не способен выполнять какие-либо операции ввода / вывода (I / O); такие как сеть, хранилище или графические объекты. Это связано с тем, что сам язык не включает эти функции и вместо этого полагается на среду хоста, в которую он встроен для этих функций.

Первоначально язык был реализован на клиентской стороне веб-браузеров ради конечного пользователя; однако в наше время движки JS расширились и могут быть реализованы во многих других типах программного обеспечения хоста; включая серверные базы данных, среды выполнения, позволяющие использовать JS в разработке мобильных и настольных приложений, и даже текстовые процессоры без веб-интерфейса и программное обеспечение PDF.

Следует также отметить, что, несмотря на многие сходства JavaScript с Java (имя, синтаксис и стандартные библиотеки), языки сильно отличаются по дизайну и отличаются друг от друга, поскольку на JS оказали влияние языки программирования, такие как Self и Scheme.

Когда был создан JavaScript, у него изначально было другое имя: «LiveScript». Но в то время Java была очень популярна, поэтому было решено, что позиционирование нового языка как «младшего брата» Java поможет.

Но по мере развития JavaScript стал полностью независимым языком со своей собственной спецификацией под названием ECMAScript, и теперь он вообще не имеет никакого отношения к Java.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или фактически на любом устройстве, на котором есть специальная программа, называемая JavaScript engine.

Браузер имеет встроенный движок, который иногда называют «виртуальной машиной JavaScript».

Разные движки имеют разные «кодовые имена». Например:

- V8 - в Chrome и Opera;
- SpiderMonkey - в Firefox;
- существуют другие кодовые имена, такие как «Trident» и «Chakra» для разных версий IE, «ChakraCore» для Microsoft Edge, «Nitro» и «SquirrelFish» для Safari и т.д.

Работа движка состоит из следующих процессов:

1) движок (встроенный, если это браузер) читает («разбирает») скрипт;
2) затем он преобразует («компилирует») скрипт в машинный язык;
3) и тогда машинный код работает, довольно быстро.

Движок применяет оптимизацию на каждом этапе процесса. Он даже наблюдает за скомпилированным сценарием во время его выполнения, анализирует данные, проходящие через него, и дополнительно оптимизирует машинный код на основе этих знаний [2].

Современный JavaScript - это «безопасный» язык программирования. Он не обеспечивает низкоуровневый доступ к памяти или процессору, поскольку изначально был создан для браузеров, которые этого не требуют.

Возможности JavaScript в значительной степени зависят от среды, в которой он работает. Например, Node.js поддерживает функции, которые позволяют JavaScript считывать, делать записи произвольных файлов, выполненять сетевые запросы и т.д.

JavaScript в браузере может делать все, что связано с манипулированием веб-страницей, взаимодействием с пользователем и веб-сервером, например:

- добавлять новый HTML на страницу, изменять существующий контент, изменять стили;
- реагировать на действия пользователя, запускать по щелчкам мыши что либо, взаимодействовать с движениями указателя, также реагировать на нажатие клавиш;
- отправлять запросы по сети на удаленные серверы, загрузка и выгрузка файлов (так называемые технологии AJAX и COMET).

Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель состоит в том, чтобы предотвратить доступ злоумышленной веб-страницы к частной информации или нанесение ущерба данным пользователя.

Примеры таких ограничений включают в себя:

- JavaScript на веб-странице не может читать или записывать произвольные файлы на жестком диске, копировать их или выполнять программы. У него нет прямого доступа к функциям ОС);
- современные браузеры позволяют ему работать с файлами, но доступ к нему ограничен и предоставляется только в том случае, если пользователь выполняет определенные действия, такие как «перетаскивание» файла в окно браузера или выбор его с помощью тега <input>;
- есть способы взаимодействия с камерой или микрофоном и другими устройствами, но они требуют явного разрешения пользователя. Таким образом, страница с поддержкой JavaScript не может незаметно включить веб-камеру, наблюдать за окружением и отправлять информацию в АНБ;
- различные вкладки или окна вообще не знают друг о друге. Иногда это происходит, например, когда одно окно использует JavaScript для открытия другого. Но даже в этом случае JavaScript с одной страницы может не получить доступ к другой, если они приходят с разных сайтов (из другого домена, протокола или порта);
- «Политика единого происхождения». Чтобы обойти это, обе страницы должны согласовать обмен данными и содержать специальный код JavaScript, который его обрабатывает. Мы рассмотрим это в учебнике;
- страница с http://luboisite.com, которую открыл пользователь, не должна иметь доступа к другой вкладке браузера с URL http://gmail.com и краже информации оттуда;
- JavaScript может легко общаться через сеть с сервером, с которого пришла текущая страница. Но его способность получать данные с других сайтов или доменов ограничена. Хотя это возможно, это требует явного

согласия (выраженного в заголовках HTTP) с удаленной стороны. Еще раз, это ограничение безопасности.

JavaScript выделяется как минимум тремя вещами:

- простые вещи делаются просто;
- поддержка всеми основными браузерами включена по умолчанию;

JavaScript – единственная браузерная технология, которая объединяет эти три вещи.

На рисунке 2.2.6.1 представлен исходный код, демонстрирующий логику небольшого приложения.

```
var todos = ["Buy a bread"];
window.setTimeout(function() {
  var input = prompt("What would you like to do?");
  while (input != "quit") {
    //list of todos
    if (input === "list") {
      console.log("*****************");
      listTodos();
      console.log("*****************");
    }
    //add new todo
    else if (input === "new") {
      addTodo();
    }
    //delete todo
    else if (input === "delete") {
      deleteTodo();
    }
    var input = prompt("What would you like to do?");
  }
  console.log("You quit the app");
  //function to show the list of todos
  function listTodos() {
    todos.forEach(function(todo, i) {
      console.log(i + ": " + todo);
    });
  }
  //function to add new todo
  function addTodo() {
    var newTodo = prompt("Enter new todo");
    todos.push(newTodo);
    console.log("Added todo");
```

Рисунок 2.2.6.1 – Программа написанная на JavaScript

Для того чтобы запустить JavaScript код вместе с HTML можно воспользоваться двумя способами:

- встроенный в HTML документ;
- внешний, благодаря которому браузер будет загружать JS код вместе с HTML – документом.

Вы можете добавить код JavaScript в HTML-документ, используя специальный тег HTML <script>, в который помещается код JavaScript.

Тег <script> может быть размещен в разделе <head> вашего HTML, в разделе <body> или после тега </ body> close, в зависимости от того, когда вы хотите загрузить JavaScript.

Как правило, код JavaScript может входить в раздел документа <head> для того, чтобы они содержались вне основного содержимого вашего HTML-документа.

Однако, если ваш сценарий должен выполняться в определенный момент в макете страницы - например, при использовании document.write для генерации контента - вы должны поместить его в нужный вам момент, где он должен быть вызван, обычно в разделе <body>.

На рисунке 2.2.6.2 приведен пример встроенного способа запуска кода JavaScript, где прописано, что код выведет текущую дату.

```
<head>
  <title>Сегодняшняя дата</title>
  <script>
      let todayDate = new Date();
      alert("Сегоднящняя дата " + todayDate);
  </script>
</head>
```

Рисунок 2.2.6.3 – Встроенный способ запуска кода JavaScript

Для размещения более крупных сценариев или сценариев, которые будут использоваться на нескольких страницах, код JavaScript обычно находится в одном или нескольких файлах js, на которые имеются ссылки в документах HTML, аналогично тому, как ссылаются на внешние ресурсы, такие как CSS.

Преимущества использования отдельного файла JavaScript включают в себя:

- разделение разметки HTML и кода JavaScript, чтобы сделать оба более простыми;
- отдельные файлы облегчают обслуживание;
- когда файлы JavaScript кэшируются, страницы загружаются быстрее.

На рисунке 2.2.6.4 показан внешний способ запуска JavaScript кода вместе с HTML страницей. Тег <script> указывает на файл script.js, который размещен в каталоге вместе с HTML документом.

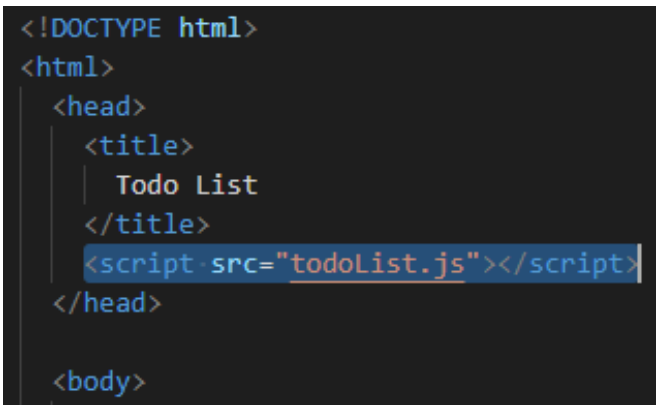

Рисунок 2.2.6.4 – Внешний способ запуска кода JavaScript

2.3 Особенности использования нереляционного СУБД MongoDB

MongoDB - это документно-ориентированная база данных с открытым исходным кодом, которая предназначена для хранения большого объема данных, а также позволяет очень эффективно работать с этими данными. Он относится к базе данных NoSQL, поскольку хранение и поиск данных в MongoDB не в форме таблиц [6].

База данных MongoDB разработана и управляется MongoDB.Inc в рамках SSPL (Public Server Public License) и первоначально выпущена в феврале 2009 года. Она также обеспечивает официальную поддержку драйверов для всех популярных языков, таких как C, C ++, C # и .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid. Таким образом, вы можете создать приложение, используя любой из этих языков. В настоящее время существует так много компаний, которые используют MongoDB, такие как Facebook, Nokia, eBay, Adobe, Google и т. Д. Для хранения своего большого объема данных .

MongoDB является сервером баз данных и данные хранятся в этих базах данных. Другими словами, среда MongoDB предоставляет вам сервер, который вы можете запустить, а затем создать на нем несколько баз данных, используя MongoDB.

Благодаря базе данных NoSQL данные хранятся в коллекциях и документах.

База данных MongoDB содержит коллекции так же, как база данных MySQL содержит таблицы. Вам разрешено создавать несколько баз данных и несколько коллекций.

Теперь внутри коллекции у нас есть документы. Эти документы содержат данные, которые мы хотим сохранить в базе данных MongoDB, и одна коллекция может содержать несколько документов, и вы не используете схемы, поэтому нет необходимости, чтобы один документ был похож на другой.

Документы создаются с использованием полей. Поля - это пары ключ-значение в документах, это как столбцы в базе данных отношений. Значения

полей могут быть любых типов данных BSON, таких как double, string, boolean и т. Д.

Данные, хранящиеся в MongoDB, имеют формат документов BSON. Здесь BSON обозначает двоичное представление документов JSON. Иными словами, в бэкэнде сервер MongoDB преобразует данные JSON в двоичную форму, известную как BSON, и этот BSON сохраняется и запрашивается более эффективно.

В документах MongoDB вы можете хранить вложенные данные. Такое вложение данных позволяет создавать сложные отношения между данными и сохранять их в одном документе, что делает обработку и выборку данных чрезвычайно эффективной по сравнению с SQL. В SQL вам нужно написать сложные объединения, чтобы получить данные из таблицы 1 и таблицы 2. Максимальный размер документа BSON составляет 16 МБ.

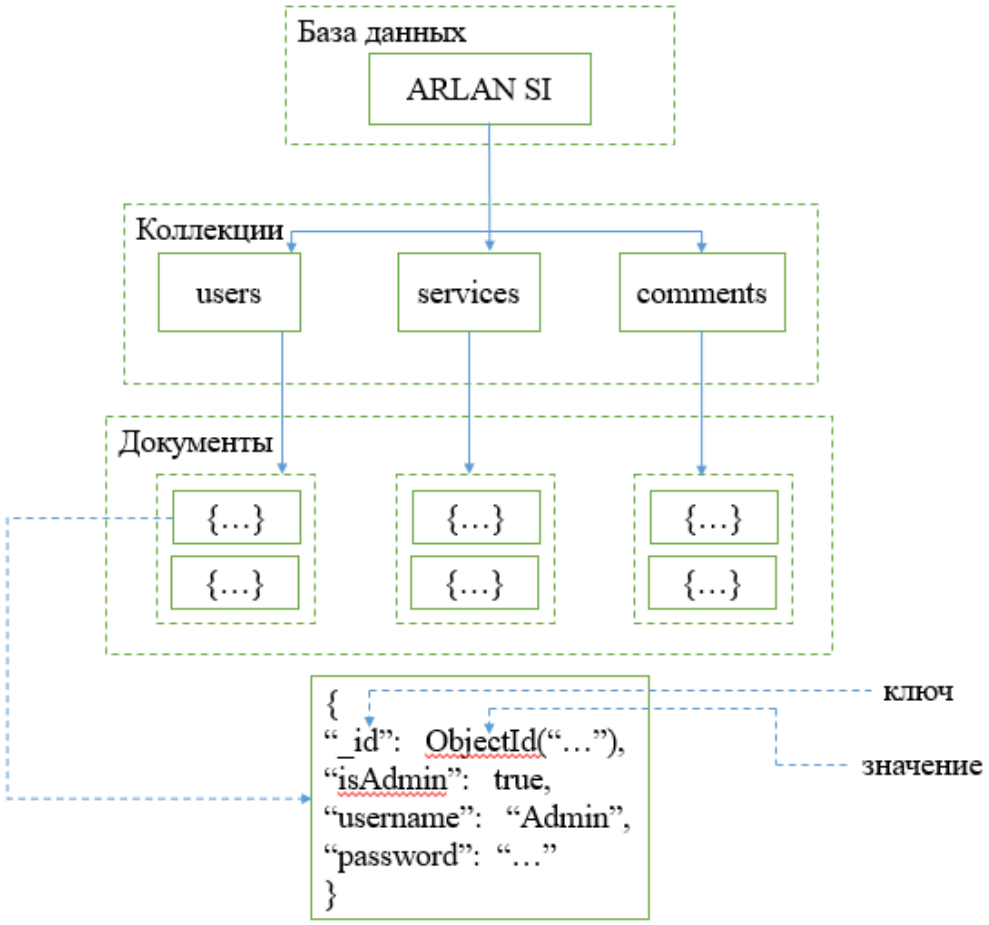

Рисунок 2.3.1 – Структура базы данных ТОО «ARLAN SI»

На рисунке 2.3.1 изображена структура базы данных ТОО «ARLAN SI» с использованием MongoDB. Внутри базы данных у нас есть три коллекции, и в этих коллекциях есть три документа. Документы содержат данные в виде полей.

MongoDB выделяется значительным количеством особенностей.

База данных без схемы: это отличная функция, предоставляемая MongoDB. База данных без схемы означает, что в одной коллекции могут храниться документы разных типов. Иными словами, в базе данных MongoDB одна коллекция может содержать несколько документов, и эти документы могут состоять из различного количества полей, содержимого и размера. Нет необходимости, чтобы один документ был похож на другой документ, как в реляционных базах данных. Благодаря этой замечательной функции MongoDB обеспечивает большую гибкость для баз данных.

Документно-ориентированная база данных: в MongoDB все данные хранятся в документах, а не в таблицах, как в РСУБД [6]. В этих документах данные хранятся в полях (пара ключ-значение) вместо строк и столбцов, что делает данные гораздо более гибкими по сравнению с РСУБД. И каждый документ содержит свой уникальный идентификатор объекта.

Индексирование. В базе данных MongoDB каждое поле в документах индексируется с помощью первичных и вторичных индексов, что облегчает задачу и занимает меньше времени для получения или поиска данных из пула данных. Если данные не проиндексированы, тогда база данных ищет каждый документ по указанному запросу, который занимает много времени и не очень эффективен.

Масштабируемость. MongoDB обеспечивает горизонтальную масштабируемость с помощью шардинга. Разделение означает распределение данных по нескольким серверам, при этом большой объем данных разделяется на блоки данных с помощью ключа сегмента, и эти блоки данных равномерно распределяются между фрагментами, которые находятся на многих физических серверах. Это также добавит новые машины в работающую базу данных.

Репликация. MongoDB обеспечивает высокую доступность и избыточность с помощью репликации, создает несколько копий данных и отправляет эти копии на другой сервер, чтобы в случае сбоя одного сервера данные извлекались с другого сервера.

Агрегация: позволяет выполнять операции над сгруппированными данными и получать один результат или вычисленный результат. Это похоже на предложение SQL GROUP BY. Он обеспечивает три различных агрегации, т.е. Конвейер агрегации, функцию уменьшения карты и методы агрегации с одним назначением.

Высокая производительность: производительность MongoDB очень высока и устойчивость данных по сравнению с другой базой данных благодаря таким функциям, как масштабируемость, индексация, репликация и т.д.

Таблица 2.3.1 – Сравнительная таблица MongoDB и РСУБД

MongoDB	**РСУБД**
Нереляционная и документно-ориентированная база данных.	Реляционная база данных.
Основанный на документе	Основанный на строках
Основанный на полях	Основанный на столбцах
Основанный на коллекциях и пар ключ-значение	Основанный на таблицах
С точки зрения производительности это намного быстрее, чем RDBMS.	С точки зрения производительности, это медленнее, чем MongoDB.
Дает JavaScript пользователям делать запросы	Не дает JavaScript пользователям делать запросы
Нет уязвимости SQL-внедрений	Довольно уязвима для SQL-внедрений
Имеет динамическую схему и идеально подходит для иерархического хранения данных	Имеет предопределенную схему и не подходит для иерархического хранения данных
В 100 раз быстрее и масштабируется по горизонтали за счет шардинга	За счет увеличения оперативной памяти может произойти вертикальное масштабирование

3 Проектирование web-приложени

При разработке сайта используется язык-программирования Node.js отвечающий за серверную часть web-приложения, также используются HTML, CSS, JavaScript отвечающие за клиентскую часть.

Node.js – это кроссплатформенная среда выполнения JavaScript с открытым исходным кодом и библиотека для запуска веб-приложений вне браузера клиента. Райан Даль разработал его в 2009 году, а его последняя версия v13.8.0 была выпущена 30 января [4]. Node.js используется для создания серверных веб-приложений и идеально подходит для приложений, интенсивно использующих данные, поскольку он использует асинхронное событие. Несмотря на то, что JS лежит в основе всех процессов сборки приложений, в качестве внутренней среды разработки Node.js отличается от внешней среды. Он имеет уникальные API-интерфейсы, которые поддерживают HTTP-запросы, файловые системы и другие функции на стороне сервера

Для облегчения и ускорения работы с Node.js используется минималистичный фреймворк Express.js. Expres.js предоставляет функции для маршрутизации, управления запросами и представлениями.

Для установки различных пакетов и запуска приложения используется npm

npm – это инструмент командной строки, который помогает взаимодействовать с онлайн-платформами, такими как браузеры и серверы. Эта утилита помогает в установке и удалении пакетов, управлении версиями и управлении зависимостями, необходимыми для запуска проекта.

3.1 Реализация функционала сайта

Для начала необходимо создать директорию где будет размещен проект. В терминале bash, который расположен в IDE Visual Code и вызывается с помощью комбинации клавиш «Ctrl + `», создается папка «ArlanSI» (рисунок 3.1). После создания папки необходимо перейти в саму папку с помощью команды «cd» (рисунок 3.2).

Рисунок 3.1.1 – Создание директории ArlanSI

Рисунок 3.1.2 – Переход в директорию ArlanSI

В терминале bash, используя менеджер пакетов npm, создается файл package.json (рисунок 3.1.3) предназначенный для хранения информации о приложении, будь то установленные пакеты, версия пакетов, название проекта, автор, а также имя файла который отвечает за запуск приложения. Данный файл формата JSON хранит все данные в виде пар-ключей (key: value).

```
$ npm init
```

Рисунок 3.1.3 – Создание JSON файла

Прописанная выше команда первым делом предлагает названия пакета, указывает версию, пишется краткое описание проекта, определяется имя главного файла, запускающий проект, по умолчанию предлагают имя index.js, необходима переименовать на app.js для дальнейшего удобства.

```
package name: (arlansi)
version: (1.0.0)
description: Diploma project
entry point: (index.js) app.js
test command:
git repository:
keywords:
author: Rassim Kurbanov
license: (ISC)
```

Рисунок 3.1.4 – Создание JSON файла

После того как была завершена установка JSON файл можно устанавливать пакеты необходимые для создания web-приложения. Первым делом устанавливается express.js и сохраняется в зависимость (рисунок 3.1.5).

```
$ npm install express --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN arlansi@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in 8.392s
found 0 vulnerabilities
```

Рисунок 3.1.5 – Установка express пакета

Также понадобится установить модуль body-parser, который отвечает за обработку POST запросов. Данный модуль необходим, так как будут

вноситься данные в формы и сохраняться в базе данных, то нужна возможность извлекать данные из форм.

Аналогичным образом, как и с express устанавливается body-parser.

```
$ npm install body-parser --save
npm WARN diploma@1.0.0 No repository field.

+ body-parser@1.19.0
updated 1 package and audited 200 packages in 4.174s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 3.1.6 – Установка body-parser модуля

HTML не поддерживает PUT и DELETE запросы, поэтому нужно установить модуль method-override позволяющий использовать данные запросы.

```
$ npm install method-override --save
npm WARN diploma@1.0.0 No repository field.

+ method-override@3.0.0
added 2 packages from 5 contributors and audited 206 packages in 2.01s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 3.1.7 – Установка method-override модуля

Необходимые модули были установлены, далее внедряется express в файл app.js.

```
const express = require("express"),
  app = express(),
```

Рисунок 3.1.8 – Внедрение express в код

Подключается body-parser и так как указано, что переменная app отвечает за работу функции express(), с помощью «app.use» приложение использует нужный нам модуль.

```
const bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));
```

Рисунок 3.1.8 – Внедрение body-parser в код

В нижней части проекта создается web-сервер, слушающий порт 3000 с которого будет запускаться приложение. При запуске программы с помощью терминала используя команду «nodemon» запускается приложение (рисунок 3.1.10). При запуске идет проверка на обнаружение ошибок в коде, в случае если все проходит успешно в консоли появляется сообщение об успешном запуске сервера и на каком порте.

```
app.listen(3000, function () {
  console.log("Server listening on 3000 port");
});
```

Рисунок 3.1.9 – Создание web-сервера

```
$ nodemon
[nodemon] 2.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server listening on 3000 port
```

Рисунок 3.1.10 – Создание web-сервера

Сервер настроен далее необходимо поделить проект на MVC (models-views-controllers):

- разделение разметки HTML и кода JavaScript, чтобы сделать оба более простыми;
- models;
- views;
- routes (controller).

Помимо данных MVC имеются:

- public;
- middleware.

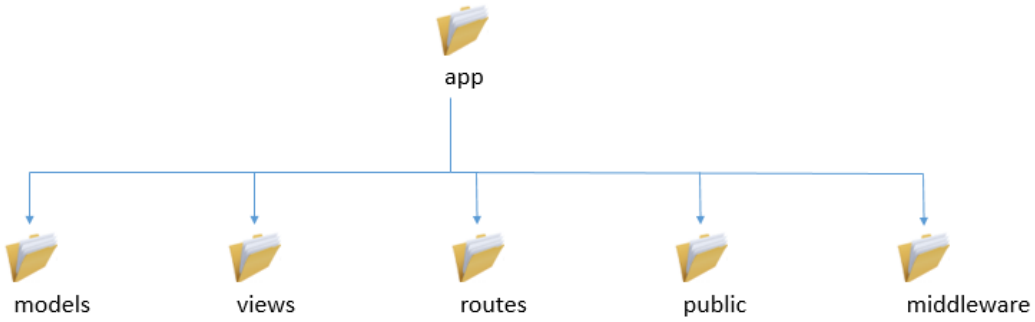

Рисунок 3.1.11 – Архитектура MVC

MVC представляет из себя шаблон архитектуры программного приложения. Он разделяет приложение на следующие компоненты:

- модели (для обработки данных);
- контроллеры (для обработки пользовательского интерфейса и приложения);
- представления (для обработки объектов графического интерфейса пользователя).

На рисунке 3.1.12 показана обработка пользовательских запросов. Сначала Браузер (на клиенте) отправляет запрос на страницу контроллеру на сервере. Затем Контроллер получает необходимые данные из модели, чтобы ответить на запрос. Контроллер передает полученные данные в представление. Представление отображается и отправляется обратно клиенту для отображения в браузере.

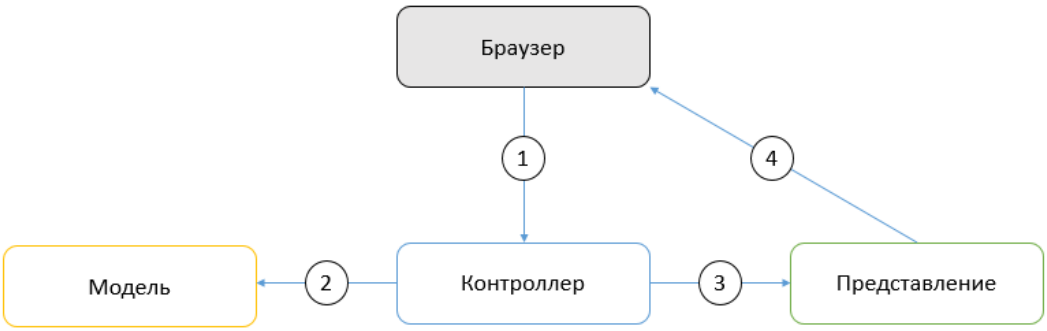

Рисунок 3.1.12 – Архитектура MVC

Разделение программного приложения на отдельные компоненты является хорошим решением по ряду причин, в том числе:

- улучшенная масштабируемость (способность приложения расти) – например, если приложение начинает испытывать проблемы с производительностью из-за медленного доступа к базе данных, можно обновить оборудование, на котором работает база данных, без влияния на другие компоненты;
- простота обслуживания – поскольку компоненты слабо зависят друг от друга, внесение изменений в один (для исправления ошибок или изменения функциональности) не влияет на другой;

- основным помощником в создании дизайна сайта отвечает Bootstrap 4.

Bootstrap – это фреймворк построенный на HTML, CSS и JavaScript для облегчения разработки адаптивных сайтов и приложений для мобильных устройств [3].

Bootstrap включает компоненты пользовательского интерфейса, макеты и инструменты JS, а также инфраструктуру для реализации. Программное обеспечение доступно предварительно скомпилированным или в виде исходного кода.

Данный фреймворк поможет сделать сайт адаптивным ко все размерам экранов, также облегчит создание стилей и анимации.

В проекте Bootstrap используется для создания навигационной шапки сайта, подвала, предание стилей кнопкам, формам, создание блоков содержащих информацию об услугах.

Bootstrap можно подключить двумя способами:

- с помощью ссылки CDN (Content Delivery Network), с которой берутся стили Bootstrap и устанавливаются в проект. Минус данного способа подключения является необходимость иметь доступ в интернет, так как все необходимые стили находятся на сервере Bootstrap;
- установить Bootstrap использую npm, либо скачать с официального сайта, плюсом является то, что нет необходимости иметь доступ в интернет, так как Bootstrap уже внедрен в проект.

```
<!-- Bootstrap style  -->
<link
  rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
  integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
  crossorigin="anonymous"
/>
```

Рисунок 3.1.13 – Подключение Bootstrap с помощью CDN

На рисунке 3.1.13 показано подключение Bootstrap в проект с помощью CDN. Данная часть кода размещена в файле header.ejs в тегах <head>.

Самое время создать навигационную шапку сайта. Шапка сайта или же хедер является навигационным элементом сайта, который помогает быстро сориентироваться и найти нужную информацию на сайте. Хедер размещает в себе все ключевые сведения о сайте.

В проекте хедер размещает в себе бренд компании, кнопки входа и регистрации на сайте, данные о компании, способ обратной связи, вывод всех услуг.

Для создание навигационной шапки сайта необходимо создать директории где будет размещен файл header.ejs.

```
$ mkdir views/partials
```

Рисунок 3.1.14 – Создание представлений

После того как были созданы директории в папке partials создается файл header.ejs, где и будет размещена шапка сайта и размещены подключения к стилям Bootstrap, CSS.

```
$ touch views/partials/header.ejs
```

Рисунок 3.1.15 – Создание файла header.ejs

```
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
  <a class="navbar-brand" href="/">ArlanSI</a>
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="#navbarCollapse"
    aria-controls="navbarCollapse"
    aria-expanded="false"
    aria-label="Toggle navigation"
  >
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarCollapse">
    <ul class="navbar-nav mr-auto">
      <li
        class="nav-item <%= typeof page !== 'undefined' && page === 'services' ? 'active' : '' %>"
      >
        <a href="/services" class="nav-link">Главная</a>
      </li>
      <li
        class="nav-item <%= typeof page !== 'undefined' && page === 'contact' ? 'active' : '' %>"
      >
        <a href="/contacts" class="nav-link">Контакты</a>
      </li>
    </ul>
    <ul class="navbar-nav navbar-right">
      <% if(!currentUser){ %>
      <li
        class="nav-item <%= typeof page !== 'undefined' && page === 'login' ? 'active' : '' %>"
      >
        <a href="/login" class="btn btn-outline-success my-2 mr-2 my-sm-0">Вход</a>
      </li>
      <li
        class="nav-item <%= typeof page !== 'undefined' && page === 'register' ? 'active' : '' %>"
      >
        <a href="/register" class="btn btn-outline-warning my-2 my-sm-0">Регистрация</a>
      </li>
      <% } else { %>
      <li class="nav-item">
        <a href="#" class="nav-link">Вы вошли как <%= currentUser.username %></a>
      </li>
      <li class="nav-item">
        <a href="/logout" class="btn btn-outline-danger my-2 my-sm-0">Выход</a>
      </li>
      <% } %>
    </ul>
  </div>
</nav>
```

Рисунок 3.1.16 – Реализация создания шапки сайта в файле header.ejs

На рисунке 3.1.16 показан код реализации навигационной шапки с использование Bootstrap фреймворка.

Рисунок 3.1.17 – Навигационная шапка сайта

Шапка сайта должна быть размещена на каждой странице, так как она отвечает за навигацию. Для того чтобы каждый раз не вставлять код шапки сайта, то есть уменьшить объем кода, к каждому ejs файлу добавляется команда include в которой указывается путь к файлу header.ejs.

```
<%- include("./partials/header") %>
```

Рисунок 3.1.18 – Подключение хедера

У каждого сайта есть шапка и подвал или же header и footer. В подвале размещаются переходы по сайту, кем был создан сайт, а также обратная связь с компанией.

Аналогично, как и header.ejs создается footer.ejs в папке views/partials

```
$  touch views/partials footer.ejs
```

Рисунок 3.1.19 – Создание файла footer.ejs

```
<footer class="footer bg-dark">
  <div class="container contFoot">
    <p class="text-muted">&copy; Arlan System Integration 2020 |
      <a id="footer-link" href="/services">Домашняя страница</a> |
      <a id="footer-link" href="/services/new">Новая услуга</a> |
      <a id="footer-link" href="/contacts">Контакты</a></p>
  </div>
</footer>
```

Рисунок 3.1.20 – Реализация кода footer.ejs

Также, как и хедер, футер должен быть размещен на каждой страницы сайта. Для этого в конце каждого файла ejs прописывается команда include и указывается путь размещения footer.ejs.

```
<%- include("./partials/footer") %>
```

Рисунок 3.1.21 – Подключение подвала (footer)

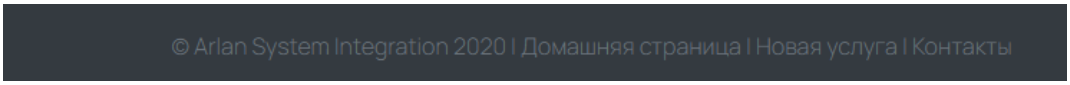

Рисунок 3.1.22 – Подвал (footer) сайта

3.2 Проектирование базы данных

MongoDB отвечает за всю базу данных сайта компании «ARLAN SI», где хранятся информация об услугах компании, данные зарегистрированных пользователей, их комментарии оставленные под определенной услугой, данные администратора.

В отличие MySQL, PostgreSQL и всех остальных реляционных баз данных MongoDB предоставляет высокую масштабируемость, быстроту работы и легкость в изменении данных.

3.2.1 Установка MongoDB в проект и создание БД

Перед тем как создавать базу данных, необходимо скачать установщик MongoDB с официального сайта выбрав версию для Windows под 64 разрядности.

Установка успешно завершена, первым делом создаем директорию для MongoDB. Для этого открываем терминал Windows Powershell и вводим следующую команду (рисунок 3.2.1.1):

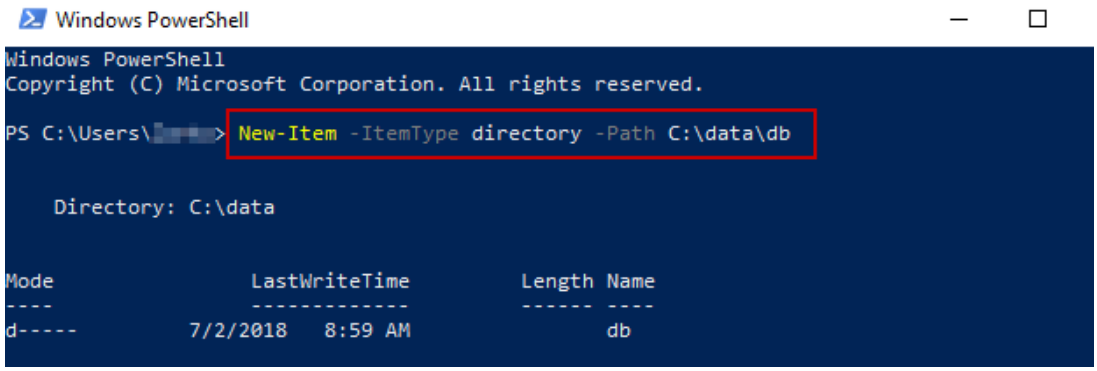

Рисунок 3.2.1.1 – Создание директории для MongoDB

Чтобы легко запустить сервер mongodb и терминал mongo, создадим файл пакетного сценария Windows (.bat), который будет размещен на рабочем столе в виде ярлыка.

Открываем редактор файлов Visual Code, создаем файл txt и вводим следующие команды:

```
@echo off
cd "C:\mongodb\bin"
```

start mongod.exe
timeout 4
start mongo.exe
exit

После этого сохраняем ранее созданный файл в формате .bat (рисунок 3.2.1.2):

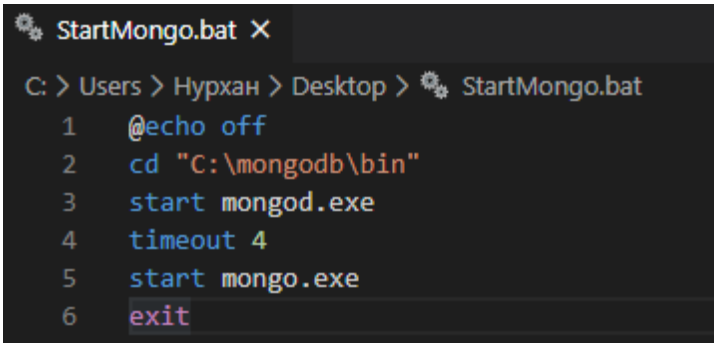

Рисунок 3.2.1.2 – Создание файла StartMongo.bat

Теперь можно запускать сервер базы данных MongoDB и терминал mongo дважды щелкнув на файл StartMongo.bat на рабочем столе, выводится окно с терминалом, в котором расписана версия mongo, к какому серверу идет подключение (по стандарту запуск сервера производится на порту 27017) (рисунок 3.2.1.3).

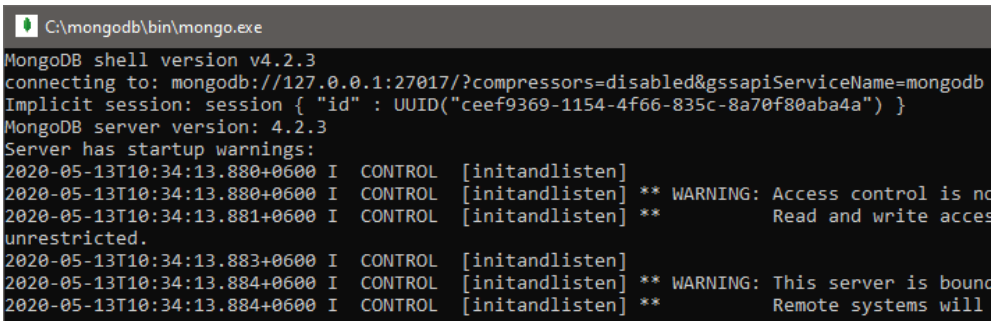

Рисунок 3.2.1.3 – Терминал MongoDB

Сервер mongo запущен, теперь создаем базу данных компании «ARLAN SI». В терминале mongo производим ввод следующих команд:

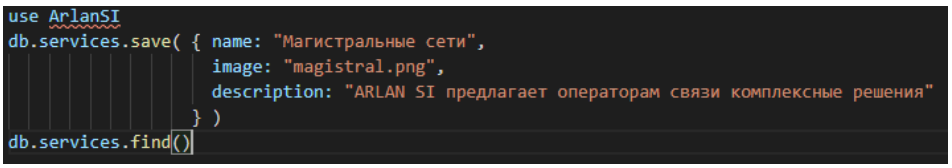

Рисунок 3.2.1.4 – Команды для работы с MongoDB

Команда use ArlanSI переключает в базу данных ArlanSI, в случае, если такой базы данных не существуют база создается автоматически. Для работы с данной базой используется переменная db. Далее создается коллекция services, аналогично, если такой коллекции не существует она создается автоматически в отличии от реляционной БД, где необходимо отдельно создавать таблицы. В команде save() создается объект, который хранит данные об услуге, такие как «{ name: “Магистральные сети” }», где name – это ключ, а значение после двоеточия – значение. То есть объект состоит из имени, изображения и описания. Данные хранятся в формате JSON, который является одним из самых удобных и легко-читабельных форматов. После сохранения объекта в БД выводится сообщение WriteResult({ "nInserted" : 1 }). Чтобы проверить был ли добавлен объект в коллекцию прописывается команда find(), которая выводит все объекты хранящиеся в данной коллекции в виде формата JSON (рисунок 3.2.1.5).

```
> use ArlanSI
switched to db ArlanSI
> db.services.save( {name: "Магистральные сети", image: "magistral.png",
                     description: "ARLAN SI предлагает операторам связи комплексные решения"} )
WriteResult({ "nInserted" : 1 })
> db.services.find()
{ "_id" : ObjectId("5ebbb88a63b78adc09edd77e"), "name" : "Магистральные сети", "image" : "magistral.png", "description"
: "ARLAN SI предлагает операторам связи комплексные решения" }
```

Рисунок 3.2.1.5 – Создание базы и коллекции

3.2.2 Создание Модели БД

В пункте «3.1 Реализация функционала сайта» было произведено разделение проекта на MVC. В созданной директории models с помощью команды на рисунке 3.2.2 создаются 3 модели сайта:

- services;
- users;
- comments.

```
$ mkdir models
```

Рисунок 3.2.2.1 – Создание директории models

Модель service представляет из себя коллекцию в которой содержатся такие данные как имя, изображение, описание услуги, также кем и когда была создана услуга, и комментарии к этой услуге от зарегистрированных пользователей сайта.

На рисунке 3.2.2.2 представлена реализация создания коллекции с данными об услуги. Объект createdAt отвечает за определение времени как давно была создана услуга. Объект author отвечает кем была создана услуга. И последний объект comments содержит комментарии.

```
var mongoose = require("mongoose");

//Создание коллекции Услуги
var serviceSchema = new mongoose.Schema({
    name: String,
    image: String,
    imageId: String,
    description: String,
    createdAt: { type: Date, default: Date.now },
    author: {
        id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        },
        username: String
    },
    comments: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Comment"
        }
    ]
});

module.exports = mongoose.model("Service", serviceSchema);
```

Рисунок 3.2.2.2 – Создание коллекции service

Вторая модель users создается в файле user.js. Модель user содержит в себе коллекцию user хранящую данные о пользователе, логин, пароль.

```
var mongoose = require("mongoose");
var passportLocalMongoose = require("passport-local-mongoose");

var UserSchema = new mongoose.Schema({
    username: String,
    password: String,
    isAdmin: {type: Boolean, default: false}
});

UserSchema.plugin(passportLocalMongoose);

module.exports = mongoose.model("User", UserSchema);
```

Рисунок 3.2.2.3 – Создание коллекции user

Также коллекция user имеет данные об администраторе типа Boolean, которое имеет значение true в случае, если пользователь вошел как админ или false, если это обычный пользователь.

В модели users используется плагин passport-local-mongoose, который устанавливается через командную строку с помощью npm менеджера. Данный плагин способствует зашифровки пароля в базе данных для безопасности данных пользователей. Если злоумышленник попытается узнать пароль пользователя используя консоль mongo выйдут данные, но пароль будет зашифрован благодаря плагину passport-local-mongoose (рисунок 3.2.2.4).

Рисунок 3.2.2.4 – Вывод данных пользователя с зашифрованным паролем

В третьей модели comment содержится коллекция comment хранящая в себе комментарии, кем были созданы комментарии и когда.

```
var mongoose = require("mongoose");

var commentSchema = new mongoose.Schema({
    text: String,
    createdAt: { type: Date, default: Date.now },
    author: {
        id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        },
        username: String
    }
});

module.exports = mongoose.model("Comment", commentSchema);
```

Рисунок 3.2.2.5 – Создание коллекции comment

Для того чтобы использовать данные модели в проекте необходимо их экспортировать в файл, где будут применятся модели. В конце каждой модели прописан код позволяющий глобально использовать модели, для каждой модели вводятся их соответствующие коллекции. Например для модели услуг пишется код module.exports = mongoose.model("Service", serviceSchema).

```
const Service = require("./models/service");
const Comment = require("./models/comment");
const User = require("./models/user");
```

Рисунок 3.2.2.5 – Подключение моделей

3.3 Создание запросов

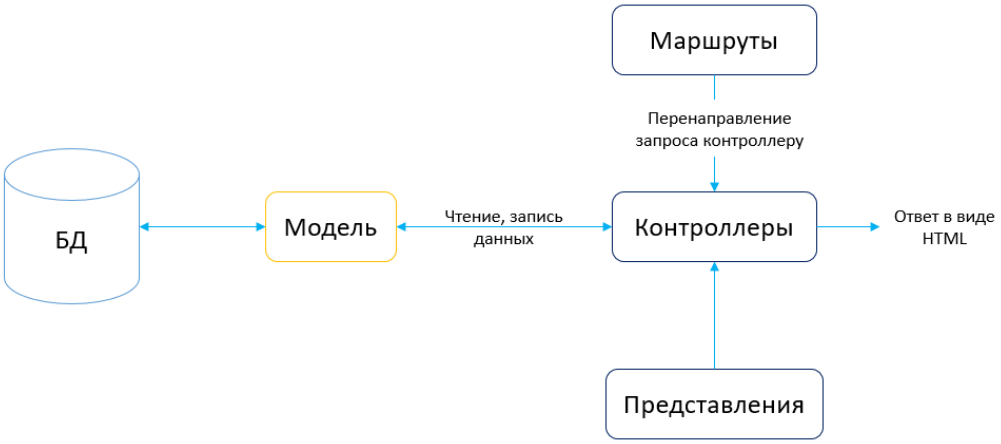

Рисунок 3.3.1 – Взаимодействие MVC

HTTP - запросы – это двоичные информационные пакеты, которые компьютер или клиент отправляет на другой компьютер или сервер для обмена данными.

Маршрутизация в основном определяется как процесс выбора пути для трафика или запроса в сети или в веб-среде. В Express можно получить доступ к веб-страницам, файлам, изображениям, сценариям, таблицам стилей и т.д. По разным путям, заданным маршрутом. Чтобы определить маршрут в Express, мы используем app.method (путь, обработчик).

Эта функция используется для определения различных маршрутов для разных типов запросов. Путь – это адрес маршрута, а обработчик – это функция, которая будет принимать запрос и давать ответ на этот запрос.

В таблице представлены маршруты запросов сайта, отвечающие за вывод всех услуг, комментариев компании, вывод определенной услуги, создание редактирование, удаление услуг, комментариев компании «ARLAN SI».

Таблица 3.3.1 – REST маршруты запросов

Имя	Путь	HTTP запрос	Описание
Index	/services	GET	Список всех услуг
New	/services/new	GET	Показ формы для создания новой услуги
Create	/services	POST	Создание новой услуги
Show	/services/:id	GET	Показ информации одной из услуг
Edit	/services/:id/edit	GET	Показ формы редактирования одной из услуг
Update	/services/:id	PUT	Обновление определенной услуги
Destroy	/services/:id	DELETE	Удаление определенной услуги

```
var express = require("express");
var router = express.Router();
```

Рисунок 3.3.2 – Подключение роутера

На рисунке представлен код создания запроса, который выводит все услуги компании используя функцию Service.find(), которая ищет в базе данных созданные услуги. В случае обнаружения ошибки, услуги не будут показаны.

```
//INDEX - show all services
router.get("/", function(req, res) {
    //Get all services from DB
    Service.find({}, function(err, allServices) {
      if (err) {
        console.log(err);
      } else {
        res.render("servicess/index", { services: allServices, currentUser: req.user });
      }
    });
  });
```

Рисунок 3.3.3 – Создание запроса для вывода всех услуг

После успешного обнаружения услуг, срабатывает представление, которое перенаправляет на файл index.ejs отвечающий за визуальную часть, где будут выведен список услуг.

```
<div class="row text-center" style="display: flex; flex-wrap: wrap;">
  <% services.forEach(function(service){ %>
  <div class="col-lg-3 col-md-4 col-sm-6 mb-4">
    <div class="card">
      <img
        src="<%= service.image %>"
        class="card-img-top"
        alt="<%= service.name %>"
      />
      <div class="card-body">
        <h5 class="card-title"><%= service.name %></h5>
        <a href="/services/<%= service._id %>" class="btn btn-primary"
          >Подробней</a
        >
      </div>
    </div>
  </div>
  <% }); %>
</div>
</div>
```

Рисунок 3.3.4 – Создание блоков содержащих услуги

Express позволяет добавлять JavaScript код в HTML документ в специальные скобки <% %>. С помощью цикла проходимся по всем услугам и выводим их на главную страницу. Услуги содержат изображение, имя и кнопку перенаправляющую на страницу подробной информации выбранной услуги.

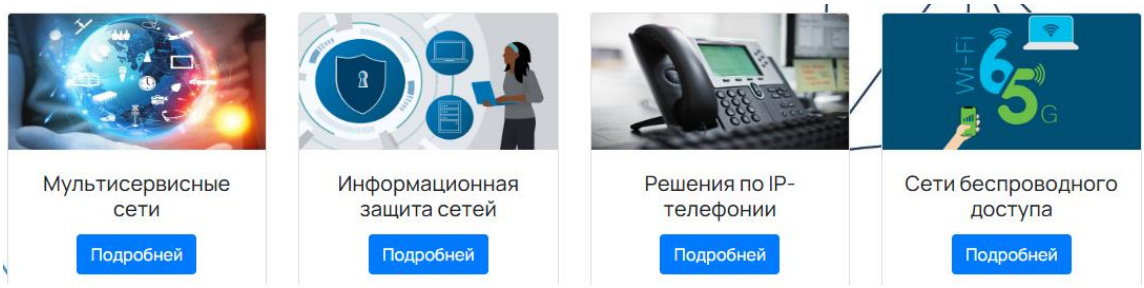

Рисунок 3.3.4 – Вывод услуг

При создании новой услуги применяется функция Service.create(), которая добавляет услугу в базу данных. С помощью запроса POST отправляются введенные данные в БД.

Для создания или же редактирования услуг необходимо обладать правами администратора. В случае если обычный пользователь попытается зайти на страницу создания услуги, появится сообщение о том что пользователь не обладает разрешением для создания услуги.

```
//CREATE - add new service to DB
router.post("/", middleware.isLoggedIn, upload.single('image'), function(req, res) {
  //get data from form and to services array
  cloudinary.v2.uploader.upload(req.file.path, function(err, result) {
    if(err) {
      req.flash('error', err.message);
      return res.redirect('back');
    }
    // add cloudinary url for the image to the service object under image property
    req.body.service.image = result.secure_url;
    // add image's public_id to campground object
    req.body.service.imageId = result.public_id;
    // add author to campground
  req.body.service.author = {
    id: req.user._id,
    username: req.user.username
  }
  //Create a new service and save to DB
  Service.create(req.body.service, function(err, service) {
    if (err) {
      req.flash('error', err.message);
      return res.redirect('back');
    } else {
      //redirect back to service page
      console.log(service);
      res.redirect('/services/' + service.id);
    }
  });
 });
});
```

Рисунок 3.3.5 – Запрос на создание новой услуги

На рисунке 3.3.6 создается код файла ejs для создания услуги, выводящий поля для заполнения названия, описания и выбор файла для изображения.

```
<div class="row justify-content-center">
  <div class="col-md-12 text-center">
    <h1 class="h3 mb-3 font-weight-normal text-center">
      <span>Создать новую услугу</span>
    </h1>
  </div>
  <div class="col-sm-12 col-md-6 col-lg-4">
    <form action="/services" method="POST" enctype="multipart/form-data">
      <div class="form-group">
        <label for="name">Название</label>
        <input class="form-control" type="text" name="service[name]" placeholder="name"
        />
      </div>
      <div class="form-group">
        <label for="image">Фотография</label>
        <input type="file" class="form-control-file" size="60" id="image" name="image" accept="image/*" required />
      </div>
      <div class="form-group">
        <label for="description">Описание</label>
        <textarea class="form-control" type="text" name="service[description]" placeholder="description"
        ></textarea>
      </div>
      <div class="form-group">
        <button class="btn btn-lg btn-primary btn-block">Отправить</button>
      </div>
    </form>
```

Рисунок 3.3.6 – Файл new.ejs для создания услуги

Как показано на рисунке 3.3.7 имеются поля для заполнения данных услуги которую необходимо создать. После ввода данных в поля нажимается кнопка «Отправить», которая отправляет запрос POST для создания новой услуги и добавления ее в базу данных, если услуга прошла успешно запрос, то выводится страница уже с созданной услугой.

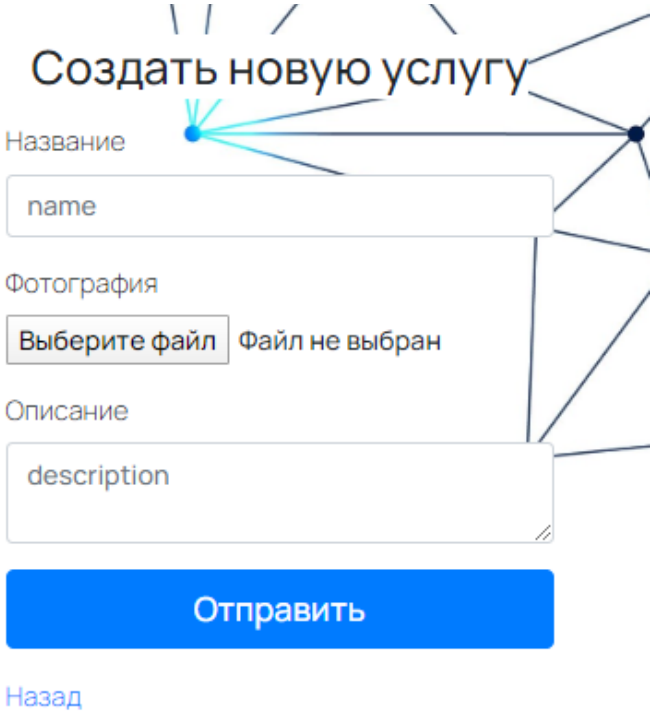

Рисунок 3.3.7 – Форма создания услуги

Созданная услуга может нуждаться в редактировании описания, изменения изображения. Для этого необходимо реализовать запрос PUT, которой будет обновлять услугу.

Для начала реализуем переход на страницу редактирования определенной услуги.

```
// EDIT SERVICE ROUTE
router.get("/:id/edit", middleware.checkServiceOwnership, function(req, res) {
  Service.findById(req.params.id, function(err, foundService) {
      res.render("servicess/edit", {service: foundService});
  });
});
```

Рисунок 3.3.8 – Создание запроса для вывода страницы редактирования услуги

Создается страница редактирования услуги edit.ejs. в которой находится форма содержащая данные необходимые для изменения, такие как изображения, описание или название.

```
<div class="row justify-content-center">
  <div class="col-md-12 text-center">
    <h1 class="h3 mb-3 font-weight-normal text-center"><span>Редактировать<br />
      "<%= service.name %> "</span></h1>
  </div>
  <div class="col-sm-12 col-md-6 col-lg-4">
    <form action="/services/<%= service._id %>?_method=PUT " method="POST" enctype="multipart/form-data">
      <div class="form-group">
        <label for="name">Название</label>
        <input
          class="form-control"
          type="text"
          name="service[name]"
          value="<%= service.name %> "
        />
      </div>
      <div class="form-group">
        <label for="image">Фотография</label>
        <input type="file" class="form-control-file" size="60" id="image" name="image" accept="image/*" />
      </div>
      <div class="form-group">
        <label for="description">Описание</label>
        <input
          class="form-control"
          type="text"
          name="service[description]"
          value="<%= service.description %>"
        />
      </div>
```

Рисунок 3.3.9 – Создание страницы редактирования

В тегах <form> создается форма редактирования услуги, в которых указываются параметры действий выводящие необходимую услугу, так как в HTML нет метода PUT необходимо установить библиотеку method-override, который переписывает метод POST в метод PUT или же DELETE. Чтобы использовать библиотеку, устанавливаем параметр ?_method=PUT.

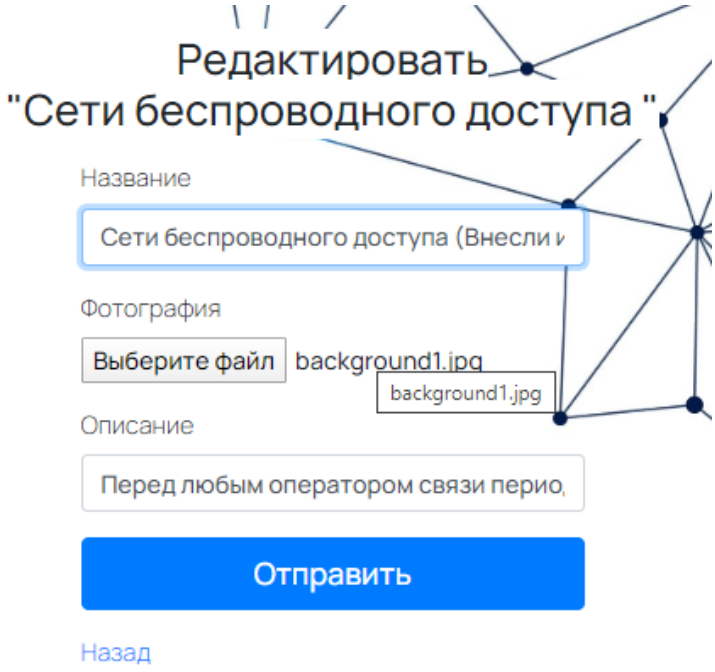

Рисунок 3.3.10 – Страница редактирование услуги

На рисунке 3.3.10 показана форма редактирования услуги «Сети беспроводного доступа», в полях названия, описания и изображение уже имеются данные, которые можно редактировать.

Страница редактирования была успешно создана, далее реализуется логика обновления записи с помощью метода PUT. Если не было обнаружено ошибок, обновленные данные сохраняются в базе данных с помощью функции service.save().

```
// UPDATE SERVICE ROUTE
router.put("/:id", middleware.checkServiceOwnership, upload.single('image'), function(req, res) {
  //find and update the correct service
  Service.findById(req.params.id, async function(err, service) {
    if(err) {
      req.flash("error", err.message);
      res.redirect("back");
    } else {
      if (req.file) {
        try {
          await cloudinary.v2.uploader.destroy(service.imageId);
          var result = await cloudinary.v2.uploader.upload(req.file.path);
           service.imageId = result.public_id;
           service.image = result.secure_url;
        } catch(err) {
          req.flash("error", err.message);
          return res.redirect("back");
        }
      }
        service.name = req.body.service.name;
        service.description = req.body.service.description;
        service.save();
      req.flash("success", "Успешно обновлено!");
      res.redirect("/services/" + req.params.id);
    }
  });
});
```

Рисунок 3.3.11 – Создание запроса для вывода страницы редактирования услуги

Помимо редактирования имеется возможность удаления услуги при необходимости. Данная функция реализовывается с помощью запроса DELETE, которого нет в HTML. Аналогично, как и с запросом PUT применяется method-override.

Ниже представлена реализация удаления услуги. Каждая услуга имеет свое id, применяя метод findById() находится услуга в базе данных и с помощью функции service.remove() услуга удаляется из базы данных и выводится flash-сообщение об успешном удаление. Так как страница услуги больше не существует осуществляется перенаправление на главную страницу.

```
// DELETE SERVICE ROUTE
router.delete("/:id", middleware.checkServiceOwnership, function(req, res) {
  Service.findById(req.params.id, async function(err, service) {
    if(err) {
      req.flash("error", err.message);
      return res.redirect("back");
    }
    try {
      await cloudinary.v2.uploader.destroy(service.imageId);
      service.remove();
      req.flash("success", "Успешно удалено!");
      res.redirect("/services");
    } catch(err) {
      if(err) {
        req.flash("error", err.message);
        return res.redirect("back");
      }
    }
  });
});
```

Рисунок 3.3.12 – Создание запроса для вывода страницы редактирования услуги

Как говорилось ранее, каждая услуга имеет свою страницу, в которой содержится подробная информация и комментарии к ней.

Создается файл show.ejs для вывода страницы услуги.

```
<div class="col-md-9">
  <div class="card mb-3">
    <img
      src="<%= service.image %>"
      class="card-img-top"
      alt="<%= service.name %>"
    />
    <div class="card-body">
      <h5 class="card-title"><%= service.name %></h5>
      <p>
        <em>
          Создано пользователем <%= service.author.username %>, <%=
          moment(service.createdAt).fromNow() %>
        </em>
      </p>
      <hr />
      <p class="card-text text-justify"><%= service.description %></p>
      <% if(currentUser && currentUser.isAdmin) { %>
      <a
        class="btn btn-sm btn-warning"
        href="/services/<%= service.id %>/edit"
        >Редактировать</a
      >
      <form
        id="delete-form"
        action="/services/<%= service._id %>?_method=DELETE"
        method="POST"
      >
        <button class="btn btn-sm btn-danger">Удалить</button>
      </form>
      <% } %>
    </div>
```

Рисунок 3.3.13 – Создание блока вывода услуги

Для создания стиля используется гибкий и расширяемый контейнер в Bootstrap 4 – card. Card является шаблоном, который можно видоизменять под себя.

Каждая веб-страница состоит из 12 частей. Bootstrap позволяет использовать свои заготовки позволяющие заполонять страницу разными размерами элементов. Для комфортного просмотра услуги используется параметр col-md-9 означающий, что контейнер с услугой будет заполнять 9/12 части страницы. Контейнер состоит из изображения, названия услуги, кем и когда была создана услуга, а также подробное описание услуги. В случае, если вход был осуществлен администратором появляются кнопки редактирования и удаления.

Рисунок 3.3.14 – Вывод услуги

Для того чтобы проверить является ли пользователь администратором и обладает ли он разрешением изменять данные, применяется функция промежуточной обработки.

Функции промежуточной обработки – это функции, имеющие доступ к объекту запроса (req), объекту ответа (res) и к следующей функции промежуточной обработки в цикле «запрос-ответ» приложения. Следующая функция промежуточной обработки, как правило, обозначается переменной next.

На рисунке 3.3.15 реализована промежуточная функция, которая проверяет, если пользователь зарегистрирован и является администратором, то он имеет право изменять данные.

```
middlewareObj.checkServiceOwnership = function(req, res, next) {
    if(req.isAuthenticated()) {
      Service.findById(req.params.id, function(err, foundService) {
        if(err) {
          req.flash("error", "Услуга не найдена!");
          res.redirect("back");
        } else {
          if(/*foundService.author.id.equals(req.user._id)||*/ req.user.isAdmin) {
            next();
          } else {
            req.flash("error", "У вас нет доступа!")
            res.redirect("back");
          }
        }
      });
    } else {
      req.flash("error", "Вы не вошли в аккаунт!");
      res.redirect("back");
    }
  }
```

Рисунок 3.3.15 – Реализация промежуточной обработки

В случае, если обычный пользователь попытается зайти на страницу редактирования, будет выведено сообщение о том, что у него нет доступа изменять данные.

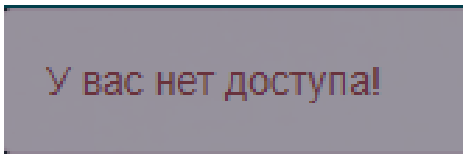

Рисунок 3.3.16 – Flash-сообщение

3.4 Создание авторизации

Прежде чем создавать комментарии необходимо быть зарегистрированным пользователем. Чтобы осуществить регистрацию в приложении, подключается модуль Passport.

Этот модуль позволяет проходить аутентификацию, используя имя пользователя и пароль в приложении Node.js. При подключении к Passport локальная аутентификация может быть легко и незаметно интегрирована в любое приложение или среду, поддерживающую промежуточное ПО в стиле Connect, включая Express.

```
const passport = require("passport"),
  LocalStrategy = require("passport-local"),
```

Рисунок 3.4.1 – Подключение модуля Passport

Конфигурация требует обратного вызова проверки, который принимает эти учетные данные и вызовы, выполненные с предоставлением пользователю.

```
// PASSPORT CONFIGURATION
app.use(
  require("express-session")({
    secret: "Where is detonator?",
    resave: false,
    saveUninitialized: false,
  })
);
app.use(passport.initialize());
app.use(passport.session());
app.locals.moment = require("moment");
passport.use(new LocalStrategy(User.authenticate()));
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());
```

Рисунок 3.4.2 – Конфигурация Passport

Создаются файлы login.ejs и register.ejs, где находятся формы ввода логина и пароля.

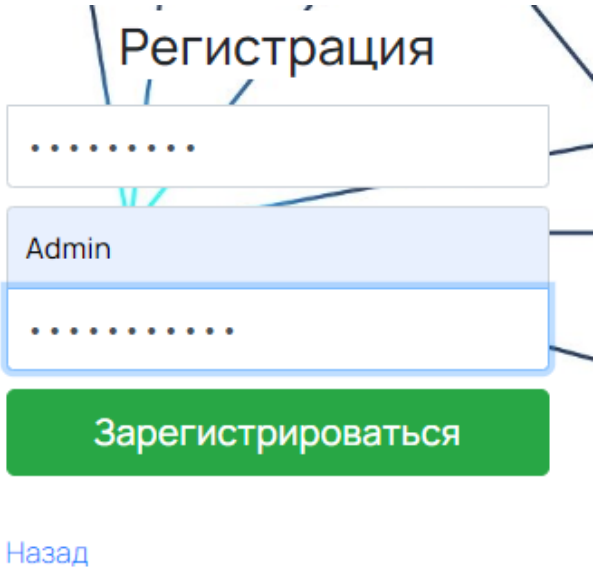

Рисунок 3.4.3 – Страница регистрации

Реализуя логику регистрации, создается объект пользователя, который будет занесен в базу данных. Если будет произведен ввод секретного кода в поле для кода администратора, то пользователь регистрируется как администратор, поле кода можно пропустить, если это обычный пользователь.

```
//handle sign up logic
router.post("/register", function(req, res)  {
  var newUser = new User({ username: req.body.username });
  //eval(require('locus'));
  if(req.body.adminCode === 'secretcode5987') {
    newUser.isAdmin = true;
  }
  User.register(newUser, req.body.password, function(err, user) {
    if (err)  {
      return res.render("register", {"error": err.message});
    }
    passport.authenticate("local")(req, res, function() {
      req.flash("success", "Добро пожаловать в Arlan SI " + user.username);
      res.redirect("/services");
    });
  });
});
```

Рисунок 3.4.4 – Создание нового пользователя

3.5 Создание комментариев

Комментарии создаются аналогичным методом, как и услуги компании. Создается запрос, который будет выводить страницу содержащую форму для создания комментариев.

```
// Comments new
router.get("/new", middleware.isLoggedIn, function(req, res) {
    //find services by id
    Service.findById(req.params.id, function(err, service) {
      if(err){
        console.log(err);
      } else {
        res.render("comments/new", { service: service });
      }
    });
  });
```

Рисунок 3.5.1 – Создание комментариев

Создаем файл new.ejs в папке views/comments, который содержит форму ввода комментария.

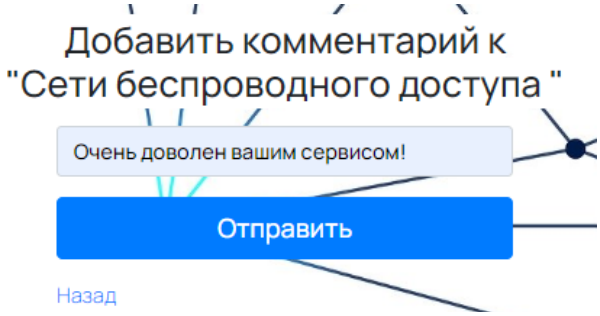

Рисунок 3.5.2 – Добавление комментария

С помощью метода POST вносим созданный комментарий к определенной услуге и сохраняем в базу данных. В случае не обнаружения ошибок, выводится сообщение об успешном добавлении комментария.

```
// Comments create
router.post("/", middleware.isLoggedIn, function(req, res) {
  //lookup services using ID
  Service.findById(req.params.id, function(err, service) {
    if (err) {
      req.redirect("/services");
    } else {
      Comment.create(req.body.comment, function(err, comment) {
        if (err) {
          console.log(err);
        } else {
          //add username and id to a comment
          comment.author.id = req.user._id;
          comment.author.username = req.user.username;
          //save comment
          comment.save();
          service.comments.push(comment);
          service.save();
          req.flash("success", "Комментарий успешно добавлен!")
          res.redirect("/services/" + service._id);
        }
      });
    }
  });
});
```

Рисунок 3.5.3 – Создание комментариев

Вывод комментария в контейнере на страницу услуги.

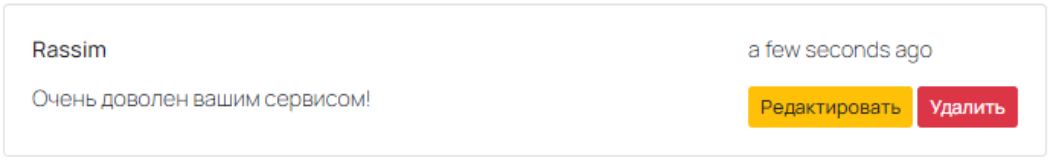

Рисунок 3.5.4 – Контейнер с комментарием

В контейнере имеются кнопки редактирования и удаления, позволяющие изменять данные комментария.

Чтобы редактировать комментарий создаем контроллер с методом findByIdAndUpdate(), который будет искать комментарий по id и обновлять его данные.

```
// COMMENT UPDATE ROUTE
router.put("/:comment_id", middleware.checkCommentOwnership, function(req, res) {
  Comment.findByIdAndUpdate(req.params.comment_id, req.body.comment, function(err, updatedComment) {
    if(err) {
      res.redirect("back");
    } else {
      res.redirect("/services/" + req.params.id);
    }
  });
});
```

Рисунок 3.5.5 – Обновление комментария

Для удаления комментария создаем контроллер с методом findByIdAndRemove, который ищет нужный комментарий и удаляет его.

```
// COMMENT DELETE ROUTE
router.delete("/:comment_id", middleware.checkCommentOwnership, function(req, res) {
  //findByIdAndRemove
  Comment.findByIdAndRemove(req.params.comment_id, function(err) {
    if(err) {
      res.redirect("back");
    } else {
      req.flash("success", "Комментарий успешно удален!")
      res.redirect("/services/" + req.params.id);
    }
  });
});
```

Рисунок 3.5.6 – Удаление комментария

3.6 Создание карты

Страница услуги помимо своей информации имеет блок, в котором размещена карта с адресом компании. Применяется карта 2GIS, так как на сегодняшний день 2GIS является самым популярным справочником с картой в Казахстане.

2GIS предоставляет свой API для использования на веб-страницах совершенно бесплатно. Подключается API в тегах <script>, в src указывается ссылка ссылка на API 2GIS. Далее создается объект map, в котором находятся координаты компании в формате широты и долготы [16].

```
<!-- 2gis Maps -->
<script src="https://maps.api.2gis.ru/2.0/loader.js?pkg=full"></script>
<script type="text/javascript">
  var map;
  var map2;

  DG.then(function () {
      map = DG.map('map', {
          center: [43.22217, 76.949965],
          zoom: 18
      });
      DG.marker([43.22217, 76.949965]).addTo(map).bindPopup('ул. Кажымукана, 86, 3 этаж');
  });
```

Рисунок 3.6.1 – Создание карты

Создаем контейнер, к котором будет размещена карта.

```
<!-- 2gis Maps -->
<li class="list-group-item">
  <h5 class="text-center">Алматы</h5>
  <div id="map" class=""></div>
</li>
```

Рисунок 3.6.2 – Создание карты

Контейнер принимает стили CSS благодаря указанному id равному «map». В главном CSS файле устанавливается высота и ширина карты к id.

```
/* 2gis Map styles */
 #map, #map2 {
  height: 400px;
  width: 100%;
}
```

Рисунок 3.6.3 – Создание стилей

В конечном итоге карта принимает вид в высоту 400 пикселей и 100 % ширину контейнера. Карта имеет свойства увеличивать и отдалять на расстояние, также имеется возможность открытия карты на полный экран.

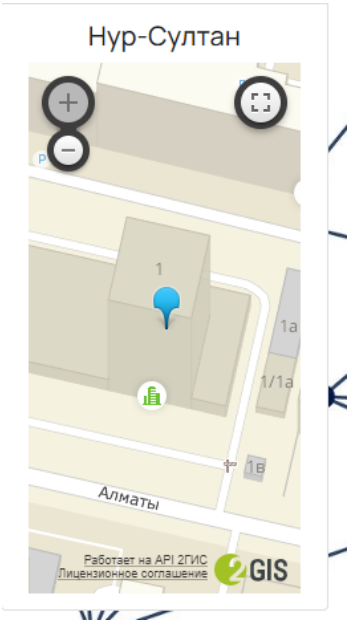

Рисунок 3.6.4 – Вывод карты

После создания всех нужных функций и реализации карты получаем окончательный вид страницы услуги.

Информация об услуги занимает 9/12 части всего экрана, с размещенными под услугой кнопками редактирования и удаления. В левой части располагаются контактные данные компании, а также карта с геопозицией компании, занимающие 3/12 части экрана. Также под услугой имеется отдельный контейнер, в котором содержатся комментарии созданные зарегистрированными пользователями сайта.

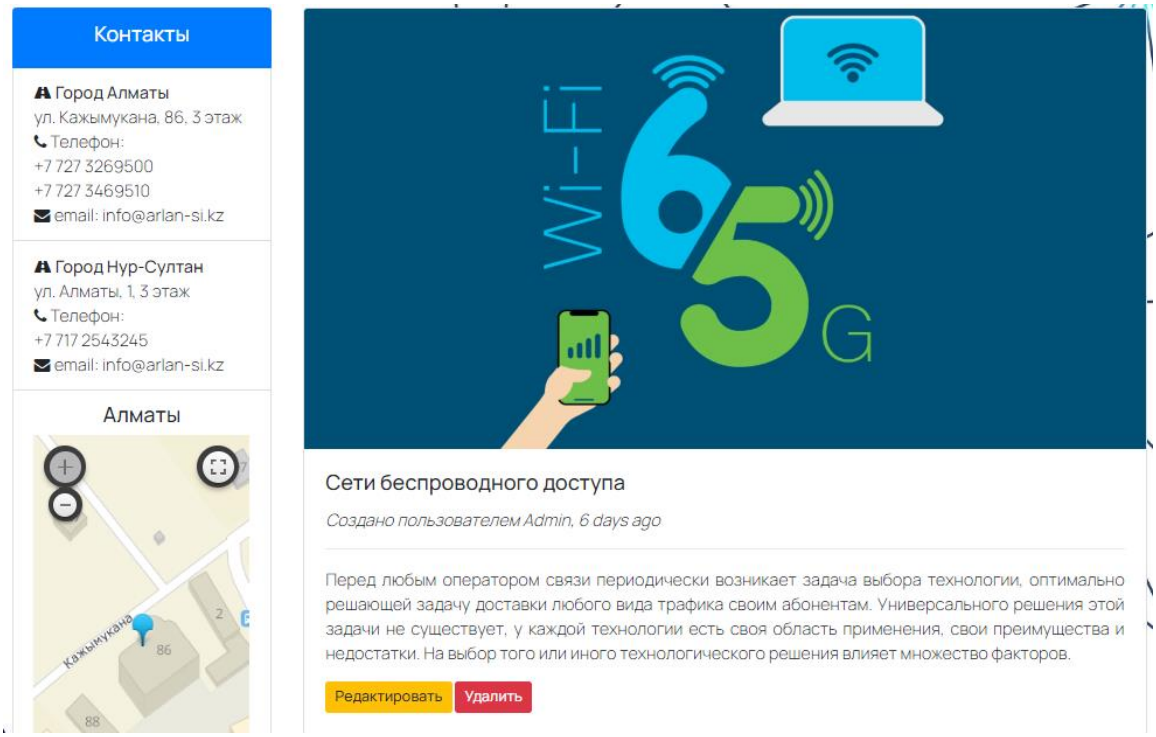

Рисунок 3.6.5 – Итоговая страница услуги

3.7 Создание формы обратной связи

Компания «ARLAN SI» должна находить новых клиентов, создавать новые отношения, связи. Для этого и используется обратная связь на сайте. Благодаря форме обратной связи клиент может связаться с компанией, указав свои данные и сообщение о предложении дальнейшего сотрудничества.

Для начала создадим файл contact.ejs, где будет размещена форма. В тегах <input> создаются поля ввода имени, названия компании, почты, номера телефона, и само сообщение, размещенное в тегах <textarea>, так как данный тег имеет поле большего размера, тег больше подходит для набирания текста для письма. Введенные данные отправляются в обработку благодаря свойству name, в котором устанавливается значение для каждого поля.

```
<div class="contact">
  <h3>Связаться с нами</h3>
  <form method="POST" action="/contacts">
    <p>
      <label>Имя</label>
      <input type="text" name="name" required />
    </p>
    <p>
      <label>Компания</label>
      <input type="text" name="company" required />
    </p>
    <p>
      <label>Почта</label>
      <input type="email" name="email" required />
    </p>
    <p>
      <label>Номер телефона</label>
      <input type="text" name="phone" required />
    </p>
    <p class="full">
      <label>Ваше видение направлений сотрудничества</label>
      <textarea name="message" rows="5" required ></textarea>
    </p>
    <p class="full">
      <button type="submit">Отправить</button>
    </p>
  </form>
```

Рисунок 3.7.1 – Создание формы обратной связи

Форма создана, теперь необходимо задать стиль форме с использованием CSS.

```
/* FORM STYLES */
.contact form{
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-gap:20px;
}
.contact form label{
  display:block;
}
.contact form p{
  margin:0;
}
.contact form .full{
  grid-column: 1 / 3;
}
.contact form button, .contact form input, .contact form textarea{
  width:100%;
  padding:1em;
  border:1px solid #c9e6ff;
}
.contact form button{
  background:#c9e6ff;
  border:0;
  text-transform: uppercase;
}
.contact form button:hover,.contact form button:focus{
  background:#92bde7;
  color:#fff;
  outline:0;
  transition: background-color 2s ease-out;
}
```

Рисунок 3.7.2 – Создание формы обратной связи

Окончательный вид формы показан на рисунке 3.7.3. Форма разделена на две части: обратная связи и контактные данные компании.

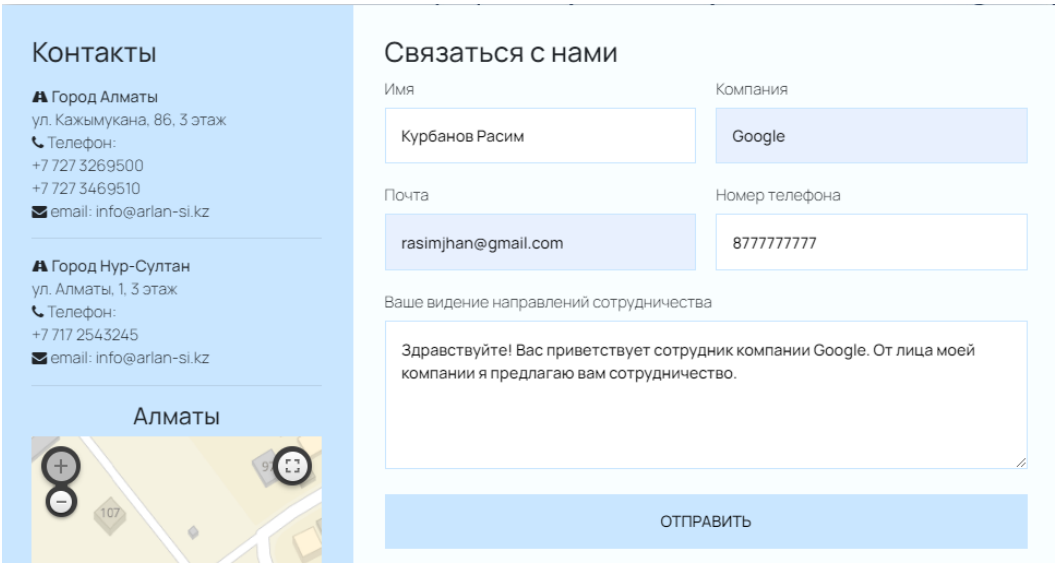

Рисунок 3.7.3 – Форма обратной связи

Форма успешно создана, реализуем логику отправки сообщения на почту компании. Для этого применяется сервис SendGrid для взаимодействия с почтовой конечной точкой API.

Устанавливаем пакет @sendgrid/mail в JSON файл, и объявляем его в приложении. Также необходимо указать API ключ для работы с сервисом SendGrid.

```
// require sendgrid/mail
const sgMail = require('@sendgrid/mail');
sgMail.setApiKey(process.env.SENDGRID_API_KEY);
```

Рисунок 3.7.4 – Подключение пакета @sendgrid/mail

Создается контроллер с запросом POST отправляющим внесенные данные на почту компании. Благодаря методу req производится запрос полей, в которых указывались данные. Переменная msg содержит данные кому отправляется сообщение и от кого, также указываются предмет сообщения и само сообщение.

```
// POST /contact
router.post('/contacts', async (req, res) => {
  let { name, company, email, phone, message } = req.body;
  name = req.sanitize(name);
  company = req.sanitize(company);
  email = req.sanitize(email);
  phone = req.sanitize(phone);
  message = req.sanitize(message);
  const msg = {
    to: 'arlan-si@gmail.com',
    from: email,
    subject: `ARLAN-SI Contact Form Submission from ${name}`,
    text: message,
    html: `
    <h1>Привет, это письмо от ${name}</h1>
    <ul>
    <li>Почта: ${email}</li>
    <li>Компания: ${company}</li>
      <li>Телефон: ${phone}</li>
    </ul>
    <p>${message}</p>
    `,
  };
```

Рисунок 3.7.5 – Реализация отправки сообщения

На рисунке 3.7.5 показан код проверки отправки сообщения. Try и catch позволяют выполнять действия асинхронно. В полях try выполняется код и выводит сообщение об успешной отправки письма. Catch проверяет на наличие ошибок при отправке.

```
try {
  await sgMail.send(msg);
  req.flash('success', 'Спасибо за ваше письмо, в скором времени мы с вами свяжемся!');
  res.redirect('/contacts');
} catch (error) {
  console.error(error);
  if (error.response) {
    console.error(error.response.body)
  }
  req.flash('error', 'Извините, что-то пошло не так, пожалуйста, свяжитесь с админом rasimjhan@gmail.com!');
  res.redirect('back');
}
```

Рисунок 3.7.5 – Проверка отправки

Логика отправки сообщения была реализована. После нажатия кнопки «отправить», сообщение отправляется на почту (рисунок 3.7.6).

Рисунок 3.7.6 – Письмо от пользователя сайта

4 Безопасность жизнедеятельности

Для дипломного проекта было выбрано web-приложение (сайт) для компании занимающейся системным интегрированием и поставкой телекоммуникационного оборудования. Сайт состоит из двух этапов: первый этап является разработкой технического задания, второй этап – программная разработка. При разработке web-сайта были соблюдены все правила техники безопасности и созданы комфортные условия для работы.

4.1 Санитарно-гигиеническая оценка условий труда

Описание офисного помещения, где было разработано web-приложении ТОО «ARLAN SI»:

- офис расположен на 2 этаже здания из 6 этажей;
- высота помещения составляет 2,7 м;
- ширина – 7,5 м;
- длина 8,6 м;
- площадь – 64,5 м^2;
- два окна размерностью (2,1 х 1,4);
- 3 компьютера и 1 принтер;
- в помещении работают 3 человека, работая с 9:00 до 18:00.

Схема помещения показана на рисунке 1.1.

Рисунок 4.1.1 – Схема рабочего места

Площадь на одно рабочее место в помещениях пользователей ПК и ВТ на базе плоских дискретных экранов (жидкокристаллические, плазменные) при любом расположении – 4 м^2 [13].

$S = n * S_0 = 3 * 4 = 12$ м^2,

где S – площадь,

n – количество возможных работников,

S_0 – необходимая площадь для работы.

Работа программиста является легкой и относится к группе 1а (работа, производимая сидя, не требующая сильного физического напряжения). Для данной категории работ по уровню энергозатратности оптимальные величины показателей микроклимата на рабочем месте рабочих помещений приведены в таблице 4.1.1.

Таблица 4.1.1 – Оптимальные величины показателей микроклимата

Период года	Температура воздуха, °С	Температура поверхностей, °С	Относительная влажность воздуха, %	Скорость движения воздуха, м/с
Холодный	20-23	20-23	42-60	0,1
Теплый	23-25	21-25	42-60	0,1

Для работы в офисном помещении важно создать комфортные условия труда для работников офиса, так как работа программиста связана с взаимодействием с компьютером и является сидячей, то выделяют следующие основные факторы: длительно сидячее положение оказывает нагрузку на позвоночник, шею, также выделяют факторы нагрузки зрения, перегрузка кистевых суставов и др.

Рабочие места соответствуют санитарно-эпидемиологическим требованиям. Площадь одного рабочее место на базе плоских дискретных экранов при любом расположении составляет не менее 4 м^2. Расстояние между рабочими столами с мониторами между боковыми поверхностями мониторов составляет не менее 1,2 м. Монитор находится от глаз на расстоянии 600 - 700 миллиметров, но не ближе 500 мм [9]. Схема размещения рабочих мест приведена на рисунке 4.1.2.

Рисунок 4.1.2 – Схема размещения рабочих мест

Рабочее место при выполнении работ сидя организуется в соответствии с ГОСТ 12.2.032-78 «Система стандартов безопасности труда. Рабочее место при выполнении работ сидя. Общие эргономические требования».

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, должно быть расположено в зоне легкой досягаемости рабочего пространства.

К требованиям эргономичного рабочего места относится следующее:

- площадь рабочего места;
- высота стола;
- положение кресла и размеры пространства для ног;
- возможность регулировки рабочего места и положения кресла;
- расстояние и углы обзора от монитора;
- расположение элементов рабочего пространства с соблюдением общих средних антропометрических показателей работника.

В кабинете имеются рекомендуемые размеры рабочего стола для средне статического взрослого человека:

- площадь рабочего места;
- ширина стола – 1200 мм;
- глубина стола – 850 мм;
- высота – 730 мм.

Также основные размеры рабочего места при работе с компьютерами высота края стола и высота пространства для ног соответствуют росту, согласно приложению 1 к настоящим Санитарным правилам. Не используются табуретки или скамейки [14].

Кресло работников регулируется по высоте. Высота сиденья варьируется от 400 до 430 см. Спинка так же может изменять угол наклона. Угол наклона спинки кресла варьируется от 90 до 110 градусов. Это позволяет работнику произвести настройку под необходимые параметры и обеспечить максимальное удобство при работе.

Для улучшения микроклимата офисного помещения используются каналы естественной вентиляции, но данной системы вентиляции недостаточно для поддержания комфортного условия труда. Кондиционер, хорошо сказывается на работе персонала, и поэтому необходимо установить кондиционер в офисном помещение, так как кондиционер способствует поддержки оптимальной температуры как летом, так и зимой. Поэтому я решил, что в данном разделе Безопасности Жизнедеятельности необходимо будет произвести расчет аспирационных систем, по результатам расчетов будет выбран подходящий кондиционер.

4.2 Расчет системы кондиционирования

Определим необходимое количество кондиционеров для создания комфортных условий труда в помещении. В помещении за счёт тепловыделений производственного оборудования могут иметь место значительные избытки тепла (разность между тепловыделениями в помещении и теплоотдачей через стены, окна, двери и т.д.), удаление которых, прежде всего, должна обеспечить система вентиляции. Избыточное тепло определяется по формуле:

$$Q_{ИЗБ} = Q_{ОБ} + Q_{ОСВ} + Q_{Л} + Q_{Р} - Q_{ОТД} \quad (4.2.1)$$

где, $Q_{ОБ}$, $Q_{ОСВ}$, $Q_{Л}$ – тепло, выделяемое производственным оборудованием, системой искусственного освещения помещения и работающим персоналом (людьми) соответственно, ккал/ч;

$Q_{Р}$ – тепло, вносимое в помещение солнцем (солнечная радиация), ккал/ч;

$Q_{ОТД}$ – теплоотдача естественным путём, ккал/ч.

Теплоизлучение, выделяемое производственным оборудованием (ПК), определяется по формуле:

$$Q_{ОБ} = 860 * P_{ОБ} * \eta \quad (4.2.2)$$

где, 860 – тепловой эквивалент 1 кВт/ч;

$P_{ОБ}$ – мощность, потребляемая оборудованием, кВт/ч;

η – коэффициент перехода тепла в помещение. Значение η = 0,95 – норма потерь потребляемой мощности на тепловыделения компьютерного оборудования.

Для трех ПК имеем:

$$Q_{\text{ОБ}} = 860 * (3 * 0{,}28) * 0{,}95 = 686{,}28 \text{ ккал/час}$$

Теплопоступление, выделяемое осветительными установками, рассчитывается по формуле:

$$Q_{\text{ОСВ}} = 860 * N * \eta \qquad (4.2.3)$$

где N – расходуемая мощность светильников, кВт;

η = 0,55 – норма потерь потребляемой мощности на тепловыделения люминесцентных ламп.

$$Q_{\text{ОСВ}} = 860 * 0{,}55 * 0{,}70 = 331{,}10$$

Теплопоступление, выделяемое людьми, рассчитывается по формуле:

$$Q_{\text{Л}} = \text{К}_{\text{Л}} * (q - q_{\text{ИСП}}) \qquad (4.2.4)$$

где $\text{К}_{\text{Л}}$ – количество работающих;

$(q - q_{\text{ИСП}})$ – явное тепло, ккал/ч;

q – тепловыделения одного человека при данной категории работ (Iа) ккал/ч.

Работа, производимая в помещении, относится к I категории работ: q=100 Вт, или 0,1 кВт для офисных помещений.

$$Q_{\text{Л}} = 3 * 860 * 0{,}1 = 258 \text{ ккал/ч}$$

Тепло, вносимое солнечной радиацией, рассчитывается по формуле:

$$Q_P = m * F * q_{\text{ОСТ}} \qquad (4.2.5)$$

где m – количество окон в помещении;

F – площадь одного окна, м^2;

$q_{\text{ОСТ}}$ – солнечная радиация через остеклённую поверхность, т.е. количество тепла, вносимое за один час через остеклённую поверхность площадью 1 м^2.

Для окна с двойным остеклением $q_{\text{ОСТ}}$ =397,670 (окна выходят на Юго-Восток, город Алматы находиться на 77 градусов восточной долготы и 43 градуса северной широты).

Количество окон равно: 2.

Площадь окна равна: 2,1*1,4 = 2,94 м^2.

$$Q_P = 2 * 2{,}94 * 397{,}670 = 2338{,}3 \text{ ккал/ч}$$

Для тёплого периода года при расчётах можно принять $Q_{отд} = 0$.

$$Q_{ИЗБ} = 686{,}28 + 331{,}10 + 258 + 2338{,}3 = 3613{,}68 \text{ ккал/ч}$$

При наличии тепло избытков количество воздуха, которое необходимо удалить из помещения рассчитываем по формуле:

$$L_b = \frac{Q_{ИЗБ} * 860}{C_b * \Delta t * \gamma_b} \quad (4.2.6)$$

где $Q_{ИЗБ}$ – избыточное тепло, ккал/ч;
C_b – – теплоёмкость воздуха (0,24 ккал/кг°С);
$\Delta t = t_{вых} - t_{вх}$;
$t_{вых}$ – температура воздуха, выходящего из помещения, °С;
$t_{вх}$ – температура воздуха, поступающего в помещение, °С;
$\gamma_b = 1{,}206$ кг/м^3 – удельная масса приточного воздуха.

Величина Δt при расчётах выбирается в зависимости от тепло напряжённости воздуха и рассчитываются по формуле 1.7

$$Q_Н = \frac{Q_{ИЗБ*860}}{V_П} \quad (4.2.7)$$

где $V_П$ – объем помещения.

$$V_П = 8{,}6 * 7{,}5 * 2{,}7 = 174{,}15 \text{ м}^3$$

$$Q_Н = \frac{36{,}137 * 860}{174{,}15} = 178{,}454 \text{ ккал/м}^3$$

Если тепло напряжённость воздуха $Q_Н < 20$ ккал/м^3, то принимают $\Delta t = 6$°С, если $Q_Н > 20$ ккал/м^3, то $\Delta t = 8$°С.

В нашем случае $Q_Н > 20$ ккал/м^3, поэтому берем $\Delta t = 8$°С.

$$L_b = \frac{36{,}137 * 860}{0{,}24 * 8 * 1{,}206 * 10^4} = 368{,}73 \text{ м}^3/\text{ч}$$

Отношение количества воздуха, поступающего в помещение за один час, к объему помещения называется кратностью воздухообмена:

$$K = \frac{L_b}{V_n} \tag{4.2.8}$$

где $V_п$ – объем помещения, м^3

$$K = \frac{368{,}73}{178{,}454} = 2{,}066$$

По найденному значению количества воздуха необходимо выбрать соответствующую модель кондиционера для обеспечения комфортного микроклимота.

Исходя из полученных данных, выберем настенный кондиционер сплит-систем.

Основные характеристики выбранного кондиционера.

Таблица 4.2.1 Основные технические характеристики настенного кондиционера серии AUX-(H)24B4/EQ

Мощность охлаж./обогрев, кВт	Потр. эл мощн,охлаж./обогрев, кВт	Потребл. ток, А	Уровень шума, дБ/А	Произв. по воздуху, м3 /ч	Размер (внешн. блок) мм	Размер (внутр. блок) мм
7/7,3	2,326/2,274	10,11/ 9,89	42	950	L 800 H 300 B 590	L 1095 H 312 B 205

Во внешнем блоке находятся компрессор для сжатия фреона и нагнетания его в холодильный контур, конденсатор, где охлаждается и конденсируется фреон и вентилятор для обеспечения циркуляции воздуха через воздушный теплообменник.

Во внутреннем блоке находятся вентилятор для обеспечения циркуляции охлажденного воздуха в помещении, воздушный теплообменник для охлаждения воздуха, жалюзи для регулировки направления воздушного потока, система фильтров грубой и тонкой очистки.

Внешний блок можно установить на стене здания, на крыше, на балконе, но при этом должен устанавливаться таким образом, чтобы исключить его повреждение падающими сосульками, заливанием дождем или заваливанием снегом. Возможна установка внешнего блока на незастекленном балконе. Место установки внешнего блока должно обеспечивать удобство осмотра и проведения техобслуживания.

Внутренний блок устанавливается непосредственно в кондиционируемом помещении таким образом, чтобы струя воздуха не была направлена непосредственно на людей. Если не соблюдать эти правила размещения, то во время жары неверное размещение кондиционера может спровоцировать простудные заболевания. Сам кондиционер 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 32-42 дБ).

Управление работой настенного кондиционера производится с дистанционного пульта, который позволяет задать режим работы кондиционера: выбор скорости вентилятора, поставить автоматическое качание жалюзи, автоматический режим благодаря которому кондиционер, в зависимости от суточных колебания температуры, выбирает нужный режим работы, обогрев, охлаждение, вентиляцию, ночной режим; задать требуемую температуру; выбрать режим работы вентилятора: настроить таймер, который включит или выключит кондиционер в заданное время.

Так как количества воздуха, необходимое для поступления в помещение равно 368,73 $м^3$/час, то будет использован один кондиционер AUX-(H)24B4/EQ, который выдает необходимый нам расход воздуха.

Рисунок 4.2.1 – Общий вид кондиционера

4.3 Вывод по безопасности и жизнедеятельности

В данной части дипломного проекта была произведена санитарно-гигиеническая оценка условий труда в которой удостоверились соблюдениям всем требования условий труда.

Расчитали необходимое количество подаваемого воздуха, после того как узнали количество воздуха сделали выбор подходящего кондиционера сплит-систем по его характеристикам.

Тем самым можно сделать вывод, что безопасность жизнедеятельности учавствует во всех аспектах жизнедеятельности человека в любых профессиях и быте. На примере разработки моего проекта были соблюдены все нормы вентиляции, установлен кондционер. Также было расчитано теплоизлучение выделяемое производственным оборудованием, теплопоступление выделяемое осветительными установками, выделяемое людьми, тепло вносимое солнечной радиацией.

При работе с данной части дипломного проекта можно сделать вывод, что комфортное условие труда в частности комфортный микроклимат обеспечивают наилучшую работоспособность и самочувствие работников.

5 Технико-экономическое обоснование

5.1 Описание работы и обоснование необходимости

Темой дипломного проекта является «Разработка web-приложения для ТОО ARLAN SI».

В данной дипломном проекте описан проект по разработке web-приложения для тех, кто ищет решения в сфере телекоммуникаций, будь то поставка телекоммуникационного оборудования или системное интегрирование. Используя данное web-приложение можно найти нужную услугу и всю информацию о ней, которую предоставляет компания ТОО «ARLAN SI» в области телекоммуникаций.

Целью данного раздела является расчет трудовых, финансовых и временных затрат на создание программного продукта и расчет себестоимости прикладной программы и определение экономической эффективности.

Чтобы определить себестоимость программы, необходимо также принять:

Прямые расходы:

1) трудоемкость разработки программного продукта;
2) материальные затраты;
3) затраты на оплату труда;
4) социальный налог.

Косвенные расходы:

5) амортизация основных фондов.

5.2 Расчет трудоемкости разработки web-приложения компании ТОО «ARLAN SI»

Таблица 5.2.1 – Распределение работ по этапам, видам и оценка их трудоемкости

Этап разработки	Вид работы	Трудоемкость разработки, чел.×ч.
1	Составление задачи	24
2	Создание алгоритмов, блок схемы, дизайн сайта	20
3	Дизайн сайта	10
4	Создание клиентской части программного проекта	66
5	Создание серверной части проекта	118
6	Тестирование	18

Продолжение таблицы 5.2.1

Этап разработки	Вид работы	Трудоемкость разработки, чел.×ч.
7	Проверка и сдача отчета	50
ИТОГО трудоемкость выполнения дипломной работы		306

Чтобы определить сколько это будет в днях, нам нужно разделить итоговое число на 8(рабочий день): 306 / 8 ≈ 38

5.3 Расчет затрат на разработку программного продукта

Для вычисления затрат на реализацию ПП необходимо найти через составления сметы, которая учитывает следующие статьи:

1) материальные затраты;
2) затраты на оплату труда;
3) социальный налог;
4) амортизация основных фондов.

Таблица 5.3.1 – Затраты на материальные ресурсы

Наименование материального ресурса	Единица измерения	Количество	Цена за единицу, тг	Сумма, тг
Ноутбук Acer E5-573G-52 HQ	шт.	1	180000	180000
Мышь проводная A4Tech Bloody V8M	шт.	1	9200	9200
ПО ARLAN SI	шт.	1	80000	80000
Картридж	шт.	1	3700	3700
Бумага А4	шт.	1	1250	1250
Visual Studio Code	шт.	1	Бесплатно	Бесплатно
Postman	шт.	1	Бесплатно	Бесплатно
ИТОГО				274150

Общая сумма затрат на материальные ресурсы ($З_М$) определяется по формуле:

$$З_М = \sum_{i=1}^{n} Р_i \times Ц_i \quad (5.3.1)$$

где $Р_i$ – 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.

$$З_{ноутбук} = 1 \times 180000 = 180000 \text{ тг}$$
$$З_{мышь} = 1 \times 9200 = 9200 \text{ тг}$$
$$З_{ПО} = 1 \times 80000 = 80000 \text{ тг}$$
$$З_{картридж} = 1 \times 3700 = 3700 \text{ тг}$$
$$З_{бумага} = 1 \times 1250 = 1250 \text{ тг}$$

$$З_{общие} = 180000 + 9200 + 80000 + 3700 + 1250 = 274150$$

Если для разработки ПП используется электрооборудование, то необходимо рассчитать затраты на электроэнергию по форме, приведенной в таблице 5.3.2

Общая сумма затрат на электроэнергию ($З_Э$) рассчитывается по формуле:

$$З_Э = \sum_{i=1}^{n} М_i \times К_i \times Т_i \times Ц \quad (5.3.2)$$

где $М_i$ – паспортная мощность i-го электрооборудования, кВт;

$К_i$ – коэффициент использования мощности i-го электрооборудования (принимается $К_i$=0,9);

$Т_i$ – время работы i-го оборудования за весь период разработки ПП ч;

Ц – цена электроэнергии, тг/кВт×ч;

i – вид электрооборудования;

n – количество электрооборудования.

Затраты на электроэнергию находятся исходя из продолжительности периода разработки ПО, количества кВт/ч, затраченных на проектирование ПО и тарифа за 1 кВт/ч. Тариф по городу Алматы для юридических лиц в 2020 году составляет 19,17 тенге за 1 кВт/ч с учетом НДС (согласно данным представленным на официальном сайте ТОО «АлматыЭнергоСбыт»).

$$З_э = 0,9 \cdot 0,9 \cdot 306 \cdot 19,17 = 4751,48 \text{ тг}$$
$$З_э = 0,3 \cdot 0,8 \cdot 306 \cdot 19,17 = 1407,85 \text{ тг}$$

Таблица 5.3.2 – Затраты на электроэнергию

Наименование оборудования	Паспортная мощность, кВт	Коэффициент использования мощности	Время работы оборудования для разработки ПП, ч	Цена электро энергии, тг/кВт·ч	Сумма, тг
Ноутбук ASUS ROG V502	0,9	0,9	306	19,17	4751,48
Освещение	0,3	0,8	306	19,17	1407,85
ИТОГО					6159,33

5.4 Расчет расходов на оплату труда

В разработке программного продукта заняты 3 сотрудника: инженер-разработчик и программист. Средняя заработная плата web-разработчика в 2020 году составляет 250 000 тг, программиста 200 000 тг, web-дизайнера 150000 тг (для города Алматы).

Рабочие часы сотрудника за месяц определяются по формуле:

$$Ч_м = N_м \cdot Ч_{рд}, \quad (5.4.1)$$

где $Ч_м$— рабочие часы сотрудника за месяц;
$N_м$—количество рабочих дней за месяц;
$Ч_{рд}$—количество рабочих часов в день.

$Ч_м = 21 \times 8 = 168$ ч.

Часовая ставка работника может быть рассчитана по формуле:

$$ЧС_i = \frac{ЗП_i}{ФРВ_i} \quad (5.4.2)$$

Web-разработчик:

$$ЧС_i = \frac{250000}{168} = 1488{,}09$$

Программист:

$$ЧС_i = \frac{200000}{168} = 1190{,}48$$

Web-дизайнер:

$$ЧС_i = \frac{150000}{168} = 892{,}85$$

где $ЗП_i$– месячная заработная плата i-го работника, тг;

$ФРВ_i$– месячный фонд рабочего времени i-го работника, час.

Для определения трудоемкости разработки ПП используются данные из таблицы 5.4.1.

Трудоемкость разработки ПП web-разработчика равна 178 чел.×ч(составление задачи, разработка алгоритмов и блок схемы, реализация клиентской части проекта, тестирование, проверка и сдача отчета).

$$Т_2 = 24+20+66+18+50 = 178 \text{ чел.×ч.}$$

Трудоемкость разработки ПП программиста равна 156 чел.×ч.(разработка блок-схемы алгоритма, разработка серверной части проекта, тестирование программы).

$$Т_2 = 20 + 118 + 18 = 156 \text{ чел.×ч.}$$

Трудоемкость разработки ПП web-дизайнера равна 76 чел.×ч.(разработка дизайна web-сайта, реализация клиентской части проекта).

$$Т_2 = 10 + 66 = 76 \text{ чел.×ч.}$$

Общая сумма затрат на оплату труда ($З_{ТР}$) определяется по формуле:

$$З_{ТР} = \sum_{i=1}^{n} ЧС_i \times Т_i \qquad (5.4.3)$$

где $ЧС_i$– часовая ставка i-го работника, тг;

$Т_i$– трудоемкость разработки ПП, чел.×ч;

i – категория работника;

n – количество работников, занятых разработкой ПП.

Web-разработчик:

$$З_{тр} = 1488{,}09×178 = 264880{,}02 \text{ тг.}$$

Программист:

$$З_{тр} = 1190{,}48×156 = 185714{,}88 \text{ тг.}$$

Web-дизайнер:

$$З_{тр} = 892{,}85×76 = 67856{,}6 \text{ тг.}$$

Общая сумма:

$$З_{тр} = 264880{,}02 + 185714{,}88 + 67856{,}6 = 518451{,}5 \text{ тг.}$$

Таблица 5.4.1 – Затраты на оплату труда

Квалификация	Трудоемкость разработки ПП, чел.×ч	Часовая ставка, тг/ч	Сумма, тг
Web-разработчик	178	1488,09	264880,02
Программист	156	1190,48	185714,88
Web-дизайнер	76	892,85	67856,6
ИТОГО			518451,5

Дополнительная заработная плата:

$$З_{доп} = З_{тр} \times 10\%$$
$$З_{доп} = 518451,5 \times 0,1 = 51845,15 \text{ тг.}$$

Фонд заработной платы:

$$Ф_{зп} = З_{тр.о} + З_{доп} \quad (5.4.4)$$

$$Ф_{зп} = 518451,5 + 51845,15 = 570296,65 \text{ тг.}$$

Расчет социального налога:

Налоги от заработной платы = 570296,65 * 10,46% = 59 650,94 (5.4.5)

Расчет амортизационных основных фондов

Общая сумма амортизационных отчислений определяется по формуле:

$$З_{АМ} = \sum_{i=1}^{n} \frac{Ф_i \times Н_{Ai} \times Т_{НИРi}}{100 \times Т_{Эфi}} \quad (5.4.6)$$

где $Ф_i$– стоимость i-го ОФ, тг;

$Н_{Ai}$– годовая норма амортизации i-го ОФ, %;

$Т_{НИРi}$– время работы i-го ОФ за весь период разработки ПП, ч;

$Т_{Эфi}$– эффективный фонд времени работы i-го ОФ за год, ч/год (по информации egov.kz за 2020 год принимается $Т_{Эфi}$=1968);

i – вид ОФ;

n – количество ОФ.

Расчет годовой нормы амортизации ОФ:

$$H_{Ai} = \frac{100}{T_{Ni}} \quad (5.4.7)$$

$$H_{Ai} = \frac{100}{3} \approx 33$$

где T_{Ni}– возможный срок использования i-го ОФ, год.

Для определения времени работы ПО для разработки ПП используются данные из таблицы 5.2.1.

Время работы ПО ARLAN SI для разработки ПП составляет 202 часов (реализация клиентской части проекта, реализация администраторской части проекта, тестирование программы).

$$T_i = 118 + 66 + 18 = 202 \text{ ч.}$$

Оборудование (Ноутбук Acer E5-573G-52 HQ):

$$З_{АМ} = \frac{180000*33*306}{100*1968} = 9235,97$$

Оборудование (Мышь проводная A4Tech Bloody V8M):

$$З_{АМ} = \frac{9200*33*306}{100*1968} = 472,06$$

Программное обеспечение ARLAN SI:

$$З_{АМ} = \frac{80000*33*202}{100*1968} = 2709,76$$

Таблица 5.4.2 – Амортизация основных фондов (ОФ)

Наименование оборудования и ПО	Стоимость оборудования и ПО, тг	Годовая норма амортизации, %	Эффективный фонд времени работы оборудования и ПО, ч/год	Время работы оборудования и ПО для разработки ПП, ч	Сумма, тг
Ноутбук Acer E5-573G-52 HQ	180000	33	1968	306	9235,97

Продолжение таблицы 5.4.2

Наименование оборудования и ПО	Стоимость оборудования и ПО, тг	Годовая норма амортизации, %	Эффективный фонд времени работы оборудования и ПО, ч/год	Время работы оборудования и ПО для разработки ПП, ч	Сумма, тг
Мышь проводная A4Tech Bloody V8M	9200	33	1968	306	472,06
ПО ARLAN SI	65000	33	1968	202	2709,76
ИТОГО					12417,79

Таблица 5.4.3 – Смета затрат на разработку ПП

Статьи затрат	Сумма, тг	%
1. Затраты на материальные ресурсы	274150	32
2. Затраты на электроэнергию	6159,33	1
3. Затраты на оплату труда	518451,5	60
4. Отчисления на социальные нужды	59 650,94	6
5. Амортизация основных фондов	12417,79	1
ИТОГО	870829,56	100

Смета затрат на разработку программного продукта

Рисунок 5.4.1 Смета на разработку ПО

Определение возможной (договорной) цены программного продукта.

Договорная цена ($Ц_Д$) для прикладных ПП рассчитывается по формуле:

$$Ц_Д = З_{НИР} \cdot \left(1 + \frac{Р}{100}\right), \quad (5.4.8)$$

где $З_{НИР}$– затраты на разработку ПП (из таблицы 1.6), тг;

Р– средний уровень рентабельности ПП – 25%.

$$Ц_Д = 870829{,}56 \cdot (1 + 0{,}25) = 870829{,}56 + 217707{,}39 = 1\ 088\ 536{,}95 \text{ тг}$$

Далее определяется цена реализации с учетом налога на добавленную стоимость (НДС), ставка НДС устанавливается законодательно Налоговым Кодексом РК. На 2020 год ставка НДС установлена в размере 12%.

Цена реализации с учетом НДС рассчитывается по формуле:

$$Ц_Р = Ц_Д \cdot (1 + НДС)$$

$$Ц_Р = 1088536{,}95 \cdot (1 + 0{,}12) = 1088536{,}95 + 130624{,}434 = 1\ 219\ 161{,}384 \text{ тг}$$

Срок окупаемости разработанного программного продукта рассчитывается исходя из затрат на заработную плату работников, участвующих в процессе. Например, средняя заработная плата web-разработчика в 2020 году составляет 250 000 тг, программиста 200 000 тг, web-дизайнера 150000 тг, общая сумма заработных плат составляет 600 000, налоговые отчисления 62 760 тенге, совокупные издержки в месяц – 662 760 тенге, в годовом исчислении - 7 953 120 тенге.

Определим условно-годовую экономию, экономическую эффективность и срок окупаемости продукта.

Экономическая эффективность проекта будет:

$$E_р = \frac{Э_{уг}}{К}$$

где $Э_{уг}$- условно-годовая экономия - 7 953 120 тенге.

К – капиталовложения - 1 219 161,384 тг.

Тогда

$$E_р = \frac{Э_{уг}}{К} = \frac{7\ 953\ 120}{1\ 219\ 161{,}384} = 6{,}52 \text{ или } 652\%.$$

С каждого вложенного тенге будет получена прибыл 6,52 тенге.

Расчетный срок окупаемости капитальных вложений составляет:

$$T_р = \frac{1}{E_р}$$

где $E_р$ - коэффициент экономической эффективности капитальных вложений.

$$T_р = \frac{1}{E_р} = \frac{1}{6{,}52} = 0{,}153 \text{ года или } 1{,}836 \text{ месяц.}$$

5.5 Вывод по технико-экономической части

В данной части дипломного проекта была проведена экономическая оценка проекта по разработке web-приложения (сайт) для ТОО «ARLAN SI». Данное web-приложение помогает компании находить новых клиентов, партнеров, тем самым повышая свою прибыль и узнаваемость в глобальной мировой сети интернет.

Стоимость реализации продукта с учетом НДС составила 1 219 161,384 тг, себестоимость – 870 829,56 тг и прибыль – 217 707,39 тг.

Основную часть затрат составляют затраты на оплату труда (60%).

Экономическая эффективность проекта составила 6,52 тг. Следовательно, срок окупаемости капитальных вложения равен 1,836 месяца. Меньше чем за 2 месяц капитальные вложения окупятся и это говорит о большой выгоде компании.

Заключение

В ходе выполнения дипломного проекта было создано web-приложение предоставляющее размещение и рекламирование услуг в сфере телекоммуникаций для компании «ARLAN SI» являющейся системным интегратором и поставщиком телекоммуникационного оборудования.

Для достижения поставленной цели были выполнены следующие задачи:

- произведен анализ структуры компании «ARLAN SI»;
- определены этапы проектирования сайта, а также были выбраны инструменты для разработки web-приложеня;
- был реализован пользовательский интерфейс, то есть front-end часть с применением HTML, CSS, JavaScript и Bootstrap 4 технологий;
- спроектирована база данных MongoDB, хранящая данные пользователей, услуг и комментариев;
- добавлены функции изменения данных об услугах, а также комментариев;
- реализована авторизация, аутентификация на сайте «ARLAN SI»;
- реализована страница обратной связи с применением технологии SendGrid позволяющая отправлять сообщения на почту;
- произведено технико-экономическое обоснование целесообразности разрабатываемого продукта, входе чего было выяснено, что экономическая эффективность проекта составила 6,52 тг. Меньше чем за 2 месяца данный продукт окупится для компании. При этом прибыль составила 217 707,39 тг;
- была произведена санитарно-гигиеническая оценка условий труда, удовлетворяющая всем требованиям. Также был произведен выбор кондиционера для улучшения микроклимота в помещении.

Данный разработанный продукт увелит конкуретноспособность компании и поможет выйти на новый уровень на рынке телекоммуникаций.

Список литературы

1 Фрейен Бен HTML5 и CSS3.Разработка сайтов для любых браузеров и устройств; Питер - Москва, 2014. - 304 с.

2 Гудман, Д. JavaScript и DHTML. Сборник рецептов. Для профессионалов / Д. Гудман. - М.: Питер, 2015. - 523 с.

3 URL: https://getbootstrap.com/

4 Кантелон, М. Node.js в действии / М. Кантелон. - М.: Питер, 2015. – 810 с.

5 URL: https://caolanmcmahon.com/posts/nodejs_style_and_structure/

6 Бэнкер, К. MongoDB в действии / К. Бэнкер. - Москва: Высшая школа, 2016. - 287 с.

7 Мерсер Drupal 6. Создание надежных и полнофункциональных веб-сайтов,блогов, форумов, порталов и сайтов-сообществ / Мерсер, Дэвид. - М.: Вильямс, 2016. - 272 с.

8 Jon Loeliger Version Control with Git; Л.: Общество охраны памятников -Москва, 2009. - 328 с.

9 Уэлдон, Дж.-Л. Администрирование баз данных; Гостехиздат - М., 2015. - 207 с.

10 Прамодкумар Дж. Садаладж, Фаулер Мартин NoSQL. Новая методология разработки нереляционных баз данных; Вильямс - М., 2015. - 192 с.

11 Герасимова, С.А. Культурология и теория телекоммуникации (для бакалавров) / С.А. Герасимова. - М.: КноРус, 2016. - 112 с.

12 Бен Лин. «Волшебство Git» /– 2016. / URL: http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/ru/ch01.html

13 СЭТ ПК РК 21-01-2015 Условие работы с источниками физических факторов

14 Санатова Т.С. Безопасность жизнедеятельности. – Учебное пособие для всех специальностей, 2017. - 407 с.

15 Г.Ш. Боканова – Методические указания по выполнению экономической части выпускной работы, 2020 г.

16 URL: https://api.2gis.ru/

Приложение А

Техническое задание

1 Основные требования:

- название разрабатываемой программы: разработка web-приложения для ТОО «ARLAN SI»;
- цель разработки: сделать компанию более конкурентно способной и вывести на новый уровень рынка телекоммуникаций.

Рекомендуемые платформы для разработки приложения (на выбор разработчика):

- JavaScript;
- Python;
- Node.js;
- Vue.js;
- React.js;
- MongoDB.

2 Технические требования:

- для просмотра сайта, можно использовать любой компьютер с выходом в интернет и установленным браузером Internet Explorer 9 и выше, или Firefox 33 и выше, или Opera 22 и выше, или Safari 8.0 и выше, или Google Chrome 38 и выше.

3 Экономические требования.

Расчет сметы разработки программного продукта

- стоимость готового продукта 1 219 161,384 тг;
- стоимость разработки 870 829,56 тг.

Приложение Б
Листинг программы

```
require("dotenv").config();
const express = require("express"),
  exphbs = require("express-handlebars"),
  app = express(),
  bodyParser = require("body-parser"),
  mongoose  = require("mongoose"),
  flash = require("connect-flash"),
  passport = require("passport"),
  LocalStrategy = require("passport-local"),
  methodOverride  = require("method-override"),
  Service = require("./models/service"),
  Comment = require("./models/comment"),
  User = require("./models/user"),
  seedDB = require("./seeds"),
  expressSanitizer  = require('express-sanitizer');

// requiring  routes
const commentRoutes  = require("./routes/comments"),
  serviceRoutes = require("./routes/services"),
  indexRoutes  = require("./routes/index");

mongoose.set("useNewUrlParser",  true);
mongoose.set("useFindAndModify",  false);
mongoose.set("useCreateIndex",  true);
mongoose.set("useUnifiedTopology",  true);

mongoose.connect("mongodb://localhost/arlan_si_v12");

app.use(bodyParser.urlencoded({  extended: true }));
app.use(bodyParser.json());
app.use(express.static(__dirname  + "/public"));
app.use(methodOverride("_method"));
app.use(flash());
app.set("view  engine",  "ejs");
// Mount express-sanitizer middleware  here
app.use(expressSanitizer());

//seedDB(); //seed the database

// PASSPORT CONFIGURATION
app.use(
  require("express-session")({
    secret: "Where is detonator?",
    resave: false,
    saveUninitialized:  false,
  })

);
```

```
app.use(passport.initialize());

app.use(passport.session());
app.locals.moment = require("moment");
passport.use(new LocalStrategy(User.authenticate()));
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());

app.use(function (req, res, next) {
  res.locals.currentUser = req.user;
  res.locals.error = req.flash("error");
  res.locals.success = req.flash("success");
  next();
});

app.use("/", indexRoutes);
app.use("/services", serviceRoutes);
app.use("/services/:id/comments", commentRoutes);

app.listen(3000, function () {
  console.log("Server listening on 3000 port");
});

var express = require("express");
var router = express.Router();
var Service = require("../models/service");
var middleware = require("../middleware");
var multer = require('multer');
var storage = multer.diskStorage({
  filename: function(req, file, callback) {
    callback(null, Date.now() + file.originalname);
  }
});
var imageFilter = function (req, file, cb) {
    // accept image files only
    if (!file.originalname.match(/\.(jpg|jpeg|png|gif)$/i)) {
        return cb(new Error('Only image files are allowed!'), false);
    }
    cb(null, true);
}
var upload = multer({ storage: storage, fileFilter: imageFilter})

var cloudinary = require('cloudinary');
cloudinary.config({
  cloud_name: 'dbjgz4lro',
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET
});

//INDEX - show all services
```

```
router.get("/", function(req, res) {
    //Get all services from DB

 Service.find({}, function(err, allServices) {
      if (err) {
        console.log(err);
      } else {
        res.render("servicess/index", { services: allServices, currentUser: req.user });
      }
    });
  });

  //CREATE - add new service to DB
  router.post("/", middleware.isLoggedIn, upload.single('image'), function(req, res) {
    //get data from form and to services array
    cloudinary.v2.uploader.upload(req.file.path, function(err, result) {
      if(err) {
        req.flash('error', err.message);
        return res.redirect('back');
      }
      // add cloudinary url for the image to the service object under image property
      req.body.service.image = result.secure_url;
      // add image's public_id to campground object
      req.body.service.imageId = result.public_id;
      // add author to campground
    req.body.service.author = {
      id: req.user._id,
      username: req.user.username
    }
    //Create a new service and save to DB
    Service.create(req.body.service, function(err, service) {
      if (err) {
        req.flash('error', err.message);
        return res.redirect('back');
      } else {
        //redirect back to service page
        console.log(service);
        res.redirect('/services/' + service.id);
      }
    });
  });
});

  //NEW - show form to create a new service
  router.get("/new", middleware.isLoggedIn, function(req, res) {
    res.render("servicess/new");
  });

  //SHOW- shows more info about one service
  router.get("/:id", function(req, res) {
```

```
    //find the service with provided ID
    Service.findById(req.params.id)
      .populate("comments")

  .exec(function(err, foundService) {
        if (err) {
          console.log(err);
        } else {
          console.log(foundService);
          //render show template with that ID
          res.render("servicess/show", { service: foundService });
        }
      });
  });

  // EDIT SERVICE ROUTE
router.get("/:id/edit", middleware.checkServiceOwnership, function(req, res) {
  Service.findById(req.params.id, function(err, foundService) {
      res.render("servicess/edit", {service: foundService});
  });
});

// UPDATE SERVICE ROUTE
router.put("/:id", middleware.checkServiceOwnership, upload.single('image'), function(req, res)
{
  //find and update the correct service
  Service.findById(req.params.id, async function(err, service) {
    if(err) {
      req.flash("error", err.message);
      res.redirect("back");
    } else {
      if (req.file) {
        try {
           await cloudinary.v2.uploader.destroy(service.imageId);
           var result = await cloudinary.v2.uploader.upload(req.file.path);
            service.imageId = result.public_id;
            service.image = result.secure_url;
        } catch(err) {
          req.flash("error", err.message);
          return res.redirect("back");
        }
      }
        service.name = req.body.service.name;
        service.description = req.body.service.description;
        service.save();
      req.flash("success", "Успешно обновлено!");
      res.redirect("/services/" + req.params.id);
    }
  });
});
```

```
// DELETE SERVICE ROUTE
router.delete("/:id", middleware.checkServiceOwnership, function(req, res) {
  Service.findById(req.params.id, async function(err, service) {

 if(err) {
      req.flash("error", err.message);
      return res.redirect("back");
    }
    try {
      await cloudinary.v2.uploader.destroy(service.imageId);
      service.remove();
      req.flash("success", "Успешно удалено!");
      res.redirect("/services");
    } catch(err) {
      if(err) {
        req.flash("error", err.message);
        return res.redirect("back");
      }
    }
  });
});

  module.exports = router;

var Service = require("../models/service");
var Comment = require("../models/comment");
var middlewareObj = {};

middlewareObj.checkServiceOwnership = function(req, res, next) {
    if(req.isAuthenticated()) {
      Service.findById(req.params.id, function(err, foundService) {
        if(err) {
          req.flash("error", "Услуга не найдена!");
          res.redirect("back");
        } else {
          if(/*foundService.author.id.equals(req.user._id)||*/ req.user.isAdmin) {
            next();
          } else {
            req.flash("error", "У вас нет доступа!")
            res.redirect("back");
          }
        }
      });
    } else {
      req.flash("error", "Вы не вошли в аккаунт!");
      res.redirect("back");
    }
  }

  middlewareObj.checkCommentOwnership = function(req, res, next) {
```

```
if (req.isAuthenticated()) {
    Comment.findById(req.params.comment_id, function (err, foundComment) {
      if (err) {
        req.flash("error", "Комментарий не найден!")

  res.redirect("back");
      } else {
       // does user own the comment?
       if (foundComment.author.id.equals(req.user._id) || req.user.isAdmin) {
         next();
       } else {
         req.flash("error", "У вас нет доступа!")
         res.redirect("back");
       }
      }
    });
  } else {
    req.flash("error", "Вы не вошли в аккаунт!");
    res.redirect("back");
  }
 }

middlewareObj.isLoggedIn = function(req, res, next) {
   if(req.isAuthenticated()){
     return next();
   }
   req.flash("error", "Вы не вошли в аккаунт!");
   res.redirect("/login");
 }

module.exports = middlewareObj;
```

Приложение В

Акт внедрения

Акт внедрения

Настоящий Акт свидетельствует о том, что информационный сайт, разработанный студентом Алматинского Университета Энергетики и Связи Курбановым Расимом Анваровичем, внедрен в ТОО «ARLAN SI».

Процесс внедрения проходил с 10 по 12 марта 2020 г.

Заявленные характеристики системы предполагали наличие следующих основных возможностей:

- Лэндинг страница;
- Слайдер с фотографиями;
- Права администратора на сайте;
- Возможность оставлять комментарии;
- Функция показа всех услуг компании.

В ходе опытной эксплуатации подтверждено, что сайт обладает всеми заявленными возможностями.

На момент подписания настоящего Акта система установлена и эксплуатируется сотрудниками данной компании.

Финансовый директор _____ Ли А. А.