МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»
Кафедра IT-инжиниринг

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой
PhD, доцент
_____ Т.С. Картбаев
«____» _____2019 г.

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Разработка информационной системы для ТОО «Fin-apps»

Специальность:5В070400 – «Вычислительная техника и программное обеспечение»

Выполнил:Әшімхан Ә.Ғ. Группа:ВТу-16-4
Научный руководитель: ст.преп., Мусатаева Г.Т.

Консультанты:
по экономической части:к.э.н., профессор_____Ж.Г. Аренбаева
«20» мая 2019 г.

по безопасности
жизнедеятельности:д.т.н., ст.преп._____Ш.Ш. Бекбасаров
«20» мая 2019 г.

по применению
вычислительной техники:ст. преп._____М.Н. Майкотов
«20» мая 2019 г.

Нормоконтролер:ассист._____ Ш.П. Жумагулова
«24» 05 2019 г.

Рецензент: к.т.н., зав. каф._____В.В. Сербин
«____» _____ 2019 г.

Алматы 2019

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

Институт систем управления и информационных технологий

Кафедра IT-инжиниринг

Специальность 5В070400 – «Вычислительная техника и программное обеспечение»

ЗАДАНИЕ
на выполнение дипломногопроекта

Студенту Әшімхан Әлішер Ғалымжанұлы
Тема проекта: Разработка информационной системы для ТОО «Fin-apps»

Утверждена приказом по университету №124 от «26» октября 2018 г.

Срок сдачи законченного проекта «24» мая 2019 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта):Руководство системы менеджмента качества на предприятии; международные стандарты ИСО-9001, данные преддипломной практики.

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

- аналитическая часть;
- проектная часть;
- экспериментальная часть;
- экономическая часть;
- безопасность жизнедеятельности;
- приложение А. Техническое задание;
- приложение Б. Листинг программы;
- приложение В. Акт внедрения.

Перечень графического материала (с точным указанием обязательных чертежей): представлены 15 таблиц, 20 иллюстрации.

Основная рекомендуемая литература:

1 Эрик Фриман, Элизабет Фримен Head First HTML with CSS & XHTML// СУБД. -2005 . - № 4. - С. 31-63.

2 Изучаем паттерны проектирования JavaScript/ Эдди Османи O'Reilly, 2004. – 145 с.

3 Герд Вагнер. Создание клиентских веб-приложений– СПб.: БХВ – Санкт-Петербург, 2005 – 1152с.

4 Гофман В.Э. NoSQL базы данных. – СПб.: БХВ – Петербург, 2012. – 656 с.

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
Экономическая часть	Аренбаева Ж.Г.	04.03. 2019 – 20.05. 2019	
Безопасность жизнедеятельности	Бекбасаров Ш.Ш.	04.03 – 10.05. 2019	
Программное обеспечение	МайкотовМ.Н.	02.05. - 14.05. 2019	
Нормоконтролер	Жумагулова Ш.П	03.04.2019 – 24.05.2019	

ГРАФИК
подготовки дипломной проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Аналитическая часть	05.11.18 г. – 22.12.18 г.	
Проектирование сайта	07.01.2019 г. - 30.01.2019 г.	
Экспериментальная часть	04.02.2019 г. - 13.04.2019 г.	

Дата выдачи задания «14» января 2019 г.

Заведующий кафедрой_____Т.С. Картбаев

Научный руководительпроекта_____Г.Т. Мусатаева

Задание принял к
исполнению студент_____Ә.Ғ. Әшімхан

Аңдатпа

Дипломдық жобаның тақырыбы ««Fin-apps» ЖШС үшін ақпараттық жүйе құру» .

Дипломдық жобаның мақсаты– Қазақстандағы барлық әдемі және қызықты орындарды ашқысы келетін адамдарға арналған бағдарламалық жүйені әзірлеу.

Алға қойған мақсатқа жету үшін келесідей технологиялар қолданылды:

– клиенттік интерфейсті құру үшін: HTML, CSS, JavaScript;

– мәліметтер базасын жобалау үшін: СУБД MongoDB;

– серверлік логиканы құру үшін: Node.js, Express.js.

Дипломдық жоба кіріспеден, 5 бөлімнен және қорытындыдан тұрады.

Кіріспеде таңдалған тақырыптың өзектілігі ашылады, өңдеу мақсаты қойылады және орындауға қажетті міндеттер, өңдеуді қолдану аймағы анықталады.

Бірінші бөлімде есепті шешуге арналған технологияларға талдау жасау мен таңдау келтірілген.

Екінші бөлімде қосымшаны жобалау сипатталады.

Үшінші бөлім қосымшаны құруды сипаттаудан, қойылған мақсатты практикалық шешудің қадамдық сипатталуынан тұрады.

Төртінші бөлімде өңделген жобаның экономикалық мақсаттылығын негіздеу қарастырылған.

Бесінші бөлімде жүзеге асырылатын жоба аясында еңбек шарттарын жақсарту бойынша шаралар қарастырылған.

Қорытындыда клиент-серверлік веб-қосымшаларды құрудың қазіргі заманғы құралдары меңгерілген және қойылған мақсатқа сай өңдеу орындалған, ары қарай қосымшаны толыққанды өнімге дейін жақсарту стратегиялары құрылған.

Аннотация

Тема дипломного проекта: «Разработка информационной системы для «Fin-apps»».

Цель дипломного проекта – разработать программный продукт с детальной информацией для всех тех, кто хочет открыть для себя все красивые и интересные места Казахстана.

Для достижения поставленной цели использовались технологии:

– для разработки клиентского интерфейса: HTML, CSS, JavaScript;

– для проектирования базы данных: СУБД MongoDB;

– для разработкисерверной логики: Node.js, Express.js.

В дипломный проект входит введение, 5 глав и итоговое заключение.

Во введении раскрывается актуальность выбранной темы, ставится цель разработки и задачи, которые необходимо выполнить, определяется область применения разработки.

В первой главе производится анализ и выбор технологий для решения задачи.

Во второй главе описывается проектирование приложения.

Третья глава представляет собой описание разработки приложения, содержится пошаговое описание практического решения поставленной задачи.

В четвертой главе производится обоснование экономической целесообразности разрабатываемого проекта.

В пятой главе рассматриваются мероприятия по улучшению условий труда в рамках реализуемого проекта.

Заключение освещает результаты исследования и разработки:

– изучены современные средства разработки клиент-серверных веб-приложений и выполнена разработка согласно поставленной цели;

– разработана дальнейшая стратегия улучшения приложения до полноценного продукта.

Annotation

Theme of the graduation project: «Development of information systems for «Fin-apps» LLP».

The purpose of the study: analyze the workflow of office staff and develop software that allows increasing the productivity of specialists.

Technologies used to achieve the goal:

– development of the client user interface: HTML, CSS, JavaScript;

– creation of the database: MongoDB;

– developmentof the server logic: Node.js, Express.js.

The graduation project includes an introduction, 5 chapters and a conclusion.

The introduction reveals the relevance of the chosen topic, sets the development goal and the tasks that need to be performed, and determines the scope of the development.

The first chapter analyzes and selects technologies for solving the problem.

The second chapter describes the design of the application.

The third chapter is a description of the development stage of the application and contains a step-by-step description of the practical solution of the task.

The fourth chapter examines measures to improve working conditions in the framework of the ongoing project.

The conclusion highlights research and development results such as:

– studied modern development tools for client-server web applications and developed according to the set goal;

– further strategies to improve the application in to a full product.

Содержание

Введение

Положение в сфере рекламы в наше время высоких технологий меняется с каждым днем. На смену телерекламы и рекламы в газетах пришел новый вид рекламы - через Интернет. Веб-дизайнеры быстро поняли эффективность нового средства массовой информации. Сегодня наличие собственного сайта считается критерием современного предприятия или фирмы. Значительную часть Интернета составляют сайты, всецело посвященные рекламе.

Целью данной дипломной работы является создание сайта , соответствующего современному положению вещей в сфере рекламы с учетом особенностей Интернета. Для этого необходимо было создать сайт, отвечающий следующим требованиям :

- современный дизайн;
- дружественный интерфейс;
- удобство загрузки.

Результатом дипломной работы стал сайт ТОО "Fin-apps", который представляет собой множество логически взаимосвязанных между собой страниц. Основным принципом дизайна, реализованного в работе, стал принцип простоты и доступности.

Данный сайт является своего рода каркасом, который можно с успехом модифицировать. Так, например, если бы сайт распространял свои услуги по заказу, то можно было бы создать подобную систему заказа товара через Интернет посредством данного сайта. На данный же момент сайт отвечает всем требованиям заказчика.

Основные средства, использованные при создании сайта:

- SublimeText – редактор веб-страниц;;
- AdobePhotoshop - редактор изображений;

1 Этапы построения сайта. Сбор информации

Самый первый шаг в создании любого сайта – это сбор необходимой информации, чтобы точно знать, что ты хочешь или не хочешь для своего сайта. Это можно сделать, исследуя сайты, на которые ты уже заходишь, собирая пользовательские отзывы.

Самым главным является сбор информации о технологиях, которые будут использоваться при разработке информационной системы. От выбранной технологии будет зависеть сложность дальнейшей поддержки сайта. Ведь мало просто разработать сайт, нужно ещё и поддерживать этот сайт впоследствии.

Назначение

Необходимо задаться вопросом: Какова цель данного сайта? Сайт будет поставлять услуги, товар, информацию или будет площадкой для хранения больших данных.

Цели

Какие действия я хочу получить от посетителей во время посещения сайта?

Целевая аудитория

Какова будет целевая аудитория моего сайта?

Информация

Какого рода информацию будет искать целевая аудитория данного сайта? Они будут искать необходимую им информацию, конкретный продукт, или же онлайн покупки?

1.1 Планировка

Собранная информация из первого этапа очень сильно поможет вам в планировании сайта. Задачи, которые необходимо решить на данном этапе:

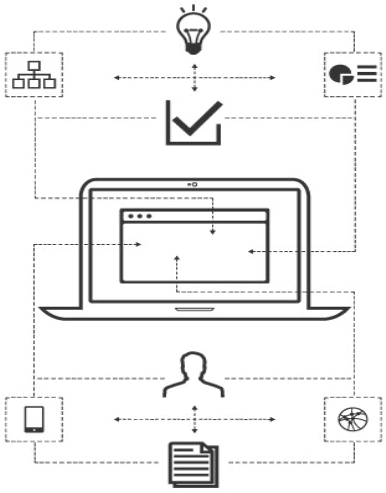


Рисунок 1.2 – Планировка

- Создание карты сайта. Список всех главных компонентов сайта, таких как темы, чтобы разработать легкое в понимании для пользователей навигационную систему;
- Решить какие технологии нужны. Определение того, что вам нужно использовать. Например, интерактивные формы, флеш-приложения или системы управления контентом;
- Определить какие разрешения экрана приспосабливать. Так как количество мобильных устройств растет с каждым днем, необходимо продумать реализацию на экранах мобильных телефонов, чтобы блоки сайта красиво располагались не только на ПК. Это позволит пользоваться сайтом на любом устройстве, независимо компьютер это или смартфон;
- Создание макетов. Это позволит вам визуализировать дизайн и создать основу для вашего сайта;
- Планирование информации. Знание располагаемой информации позволит продуманно подойти к разработке и дизайну сайта.

1.2 Дизайн

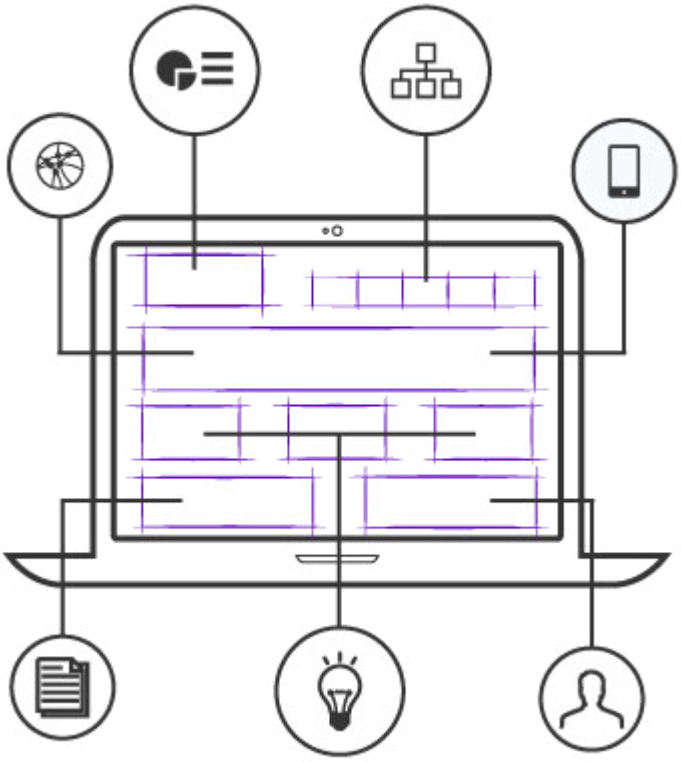


Рисунок 1.3 – Проектирование дизайна

Если подойти к деталям разработки в этапах 1 и 2 с умом, тогда создание главной страницы и всех последующих страниц будет простой. К

примеру, знание того, что основной группой вашей целевой аудитории будут студенты в возрасте от 18 до 24 лет, учащиеся в университетах, определенно будет влиять на то, какие элементы дизайна и приложения нужно использовать.

Просмотр макетов страниц с основной информацией и фотографиями, а также с логотипом действительно оживят ваше видение сайта. Используйте это время, чтобы внести необходимые изменения. Очень важно предвидеть все будущие нововведения и с самого начала думать о реализации, так как внесение больших и непредвиденных изменений в более поздних этапах является дорогостоящим и более трудным для реализации.

1.3 Разработка

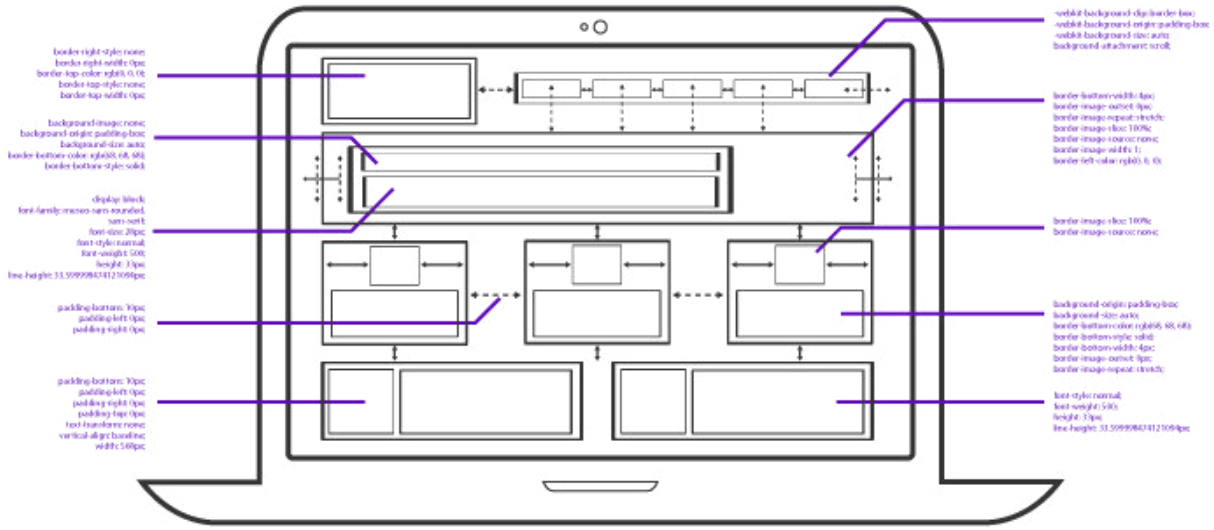


Рисунок 1.4 – Разработка

Когда уже дизайн разработан и продуман, можно уже приступать к разработке. На данном этапе используя языки программирование можно приступать к реализации функционального сайта на базе созданных ранее макетов. Все интерактивные элементы сайта, такие как контактные формы, анимации и карты для покупок будут реализованы на этом этапе.

Зачастую все небольшие изменения и корректировки вносятся именно во время данного этапа.

1.4 Тестирование и запуск

Рисунок 1.5 Тестирование

На данном этапе проводятся все тесты и проверки сайта. От полной функциональности до проблем совместимости. Кроме того, нужно убедиться, что весь код, написанный для веб-сайта, проверяется на соответствие текущим веб-стандартам.

После окончательной проверки файлы сайта будут загружены на серверы, а затем сайт будет открыт для публики.

1.5 Обслуживание

Рисунок 1.6 – Обслуживание

Когда наконец всё готово и разработано, очень важно поддерживать сайт. Во время этапа планирования вы уже должны знать где будет проводиться поддержка и обслуживание сайта – дома или довериться третьим лицам(услуги других компании по обслуживанию). Если вы решили обладать полным контролем над сайтом, вы можете сами добавлять новую информацию, когда это необходимо. Это всё реализовано благодаря системе управления контентом.

Что очень важно – разработка сайта не прекращается с запуском сайта. Как и в любом строении, нужно будет принимать меры, чтобы сайт был безопасным и работал наилучшим образом. К тому же, стандарты разработки меняются каждый год, поэтому очень важно своевременно вносить изменения, чтобы быть в ногу со временем.

2 Архитектура Web программирования

Существует много языков, которые применяются в Web программировании. Каждый из языков имеет свои нюансы и обладает своим спектром особенностей для решения задач. Web технологии принято подразделять на два вида: Front-End и Back-End.

2.1 Клиентская часть(Front-End)

Front-End – это всё, что связано с тем, что видит пользователь на сайте, включая дизайн и языки как HTML и CSS. Для того, чтобы весь код, написанный на этих языках отображался правильно необходимы браузеры.

«Web Браузер» - это программа, установленная на компьютере, которая транслирует и отображает код, который был написан на web языках программирования.

HTML - язык гипертекстовой разметки, который был разработан для представления информации (такие как текст, изображение и медиа форматы как аудио и видео).

CSS может взаимодействовать с HTML для того, чтобы можно было применять стили и визуальные улучшения.

JavaScript служит для того, чтобы улучшить пользовательский опыт(User Experience), обеспечивая интерактивность блоков информации на стороне клиента.

Сайты можно создать без подключения back-end части. Такие сайты называются статичными. Статичный сайт – это сайт, в котором нет необходимости подключаться к базе данных. К примеру, это может быть сайт-реклама ресторана или салона красоты. Всё, что видит пользователь – это перечень услуг и адрес, поэтому нет необходимости в базе данных.

2.2 Серверная часть(Back-End)

Серверная часть работает рука об руку с клиентской частью, задавая внешним элементам веб-приложения серверную логику. Для того, чтобы реализовать эту логику, back-end технология зачастую использует серверные скриптовые языки, такие как Ruby и PHP. Помимо реализации сервисной логики в планы серверной части разработки входит также и оптимизация приложения (улучшение его скорости работы и эффективности). Зачастую разработчики сталкиваются с задачей создания решения для хранения больших объемов информации в базах данных приложения. Именно здесь и приходит на помощь back-end часть. База данных является важнейшим компонентом для всех веб-приложений, так как предназначена для хранения информации о пользователях, комментариях, постах и т. д. Общие базы данных включают MySQL/SQLite, Mongo DB и PostgreSQL.

Back-end – это то, как сайт работает, обновляется и меняется. Это относится ко всему, что пользователь не видит в браузере, как базы данных и серверы. Языки серверной части могут использоваться для разных целей, таких как:

- Хранение информации в базе данных как MySQL, SQL или MongoDB;
- Получение, реструктурирование и организация информации;
- Произведение расчетов;
- Отправление писем с сайта;
- Системы авторизации пользователей;
- Системы управления контентом.

Эта технология также нужна для того, чтобы строить динамические сайты. В динамических сайтах изменения вносятся в режиме реального времени. На сегодняшний день сайты всех известных компании как Facebook и Google являются динамическими. Динамический сайт требует наличия и подключения к базе данных. Вся информация вроде пользовательских профилей, загруженных пользователями изображений и комментариев, хранятся в базе данных.

Back-end обычно состоит из трех частей: сервер, приложение и база данных. Если, к примеру, пользователь хочет купить билет на тур, то он заходит на сайт. Когда он заходит на сайт, сначала он взаимодействует с клиентской частью кода, но как только пользователь выбирает себе необходимый тур, браузер посылает запрос на сервер, и серверная часть обрабатывает этот запрос и отправляет обратно HTTP ответ с нужной информацией из базы данных.

2.3 Язык гипертекстовой разметки HTML

Язык гипертекстовой разметки, являющийся стандартизированным языком во всемирной паутине. Преимущественное большинство веб-сервисов, ресурсов и страниц содержат в себе описание разметки именно на языке HTML, в редких случаях может быть и XHTML. Этот язык интерпретируется самими браузерами и полученный в процессе всех действий и интерпретации форматированный текст отображается надисплее как мобильных устройств и планшетов, так и на экранах персонального компьютера, ноутбука либо ультрабука. Язык HTML по своей структуре является приложением SGML что в понимании: «стандартного обобщённого языка разметки» и соответствует общепринятому международному стандарту ISO 8879. Язык XHTML является наиболее строгим вариантом привычного всем HTML, и он следует всем документированным ограничениям расширяемого языка разметки XML и, фактически, XHTML можно воспринимать как приложение языка к области разметки гипертекста.

Во глобальной сети интернет, HTML-страницы, передаются браузерам от сервера после отправки запроса на получения текстовых и графических данных по протоколам HTTP или HTTPS, в виде обычного текста или с использованием шифрования предполагаемым протоколом HTTPS в основном использующимся на сайтах и страницах с передачей ключей, электронных цифровых подписей, важных данных на сервер, для исключения перехвата пакетов с данными и их использования сторонними лицами.

Язык HTML был разработан Тимом Бернерсом-Ликоторый в свою очередь на тот момент был британским ученым, в периодс 1986 по1991 гг. в стенах Европейской организации по ядерным исследованиям в Женеве в Швейцарии. Изначально язык гипертекстовой разметки HTML создавался как унифицированный язык для быстрого обмена. Научной и технической документацией, который в свою очередь должен был быть пригодный для использования людьми. Не являющимися специалистами в области вёрстки веб-страниц и плохо разбирающихся в программировании. HTML для тех времен, когда был только разработан, успешно справлялся с проблемой сложности существующего на тот момент SGML путём разрешения определения небольшого набора структурных и семантических элементов, названных дескрипторами. Дескрипторы как их еще часто называют и есть те самые «теги». С помощью гипертекстовой разметки HTML можно быстро и легко создать красиво оформленный документ без кардинальных корректировок самого документа с минимальным увеличением его исходного веса при учете фактора того, что полученные документы могут быть открыты, прочтены и отредактированы в обычном текстовом редакторе, поддерживающем HTML форматы, которых на данный момент подавляющее большинство.Помимо значительного переосмысления и упрощенияструктуры документа, в HTML внесена поддержка гипертекста, а конкретней информации содержащей в себе совокупность документов и мультимедийных файлов, связанных между собой взаимными ссылками в единый текст.

Стоит отметить, что первично язык HTML был задуман и реализован как совокупность средств структурирования, форматирования и корректной обработки документов без их конкретной привязки к средствам воспроизведения либо отображения. Как заверяет сам автор, текст с разметкой HTML предполагался без стилистических искажений отражаться на устройствах с различной аппаратной начинкой: жидкокристаллический экран современного компьютера, либо монохромный экран электронной книги, или экраны малого разрешения, к которым относятся первые мобильные устройства начала 2000-х годов.

Однако современное применение языка гипертекстовой разметки HTML ушло далеко от его изначальной задумки. К примеру, всем известный в HTML тег <table> предназначен для создания и реализации преимущественно таблиц, но примерно до 2007-2009 годов часто использовался для оформления и размещения, как отдельных элементов, так и модулей на странице в целом. С течением долгого времени идея платформ независимости которая когда-то

была основой языка HTML исчерпала свои ресурсы в угоду современным потребностям графического и мультимедийного контент наполнения.

На рисунке 2.2 показан пример гипертекстовой разметки HTMLс предположением вывода текстовой информации в окне браузера, без использования каскадных таблиц стилей CSS.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5  </head>
6  <body>
7   Hello World!
8  </body>
9  </html>
```

Рисунок 2.2 – Структура HTMLдокумента

Все текстовые документы, содержащие в себе исключительно разметку на языке HTML с учетом наполнения и гипервставок, применяющие расширение .html или .htm, в свою очередь обрабатываются специализированными приложениями, которые в свою очередь отображают исходный документ в его отформатированном виде. Такие приложения, имеют название «браузер» или раньше использовалось такое название как «интернет-обозреватель». Стоит отметить, что данное программное обеспечение в своем понимании предоставляет пользователю приятный и интуитивно понятный интерфейс для последующего запроса веб-страниц с возможностью интеграции сервисов обмена данными с сервером, такими как формы обратной связи, подписка на почтовые рассылки и другие. Популярными на данный момент браузерами являются Mozilla Firefox, Google Chrome, MicroftEdge и соответственно Safari browser.

Официальной документированной версии HTML 1.0 не существует по причине множества неофициальных стандартов разработанных. До 1995 года существовало несколько разработок HTML от разных как коммерческих, так и некоммерческих организаций под версией 1.0 которые предположительно должны были быть негласным стандартом. Чтобы официальная версия и все разработки отличалась от них, ей сразу присвоили второй номер.

Третья версия гипертекстовой разметки была предложена Консорциумом Всемирной паутины W3C в начале марта 1995 года и собственно обеспечивала множество документированных, новых возможностей с улучшенным логическим структурированием данных, таких как удобное и простое создание таблиц, функция «обтекания» изображений текстовыми блоками и данными и отображение на конечной странице сложных математических формул, также немаловажным прорывом было введение функционала предполагающего поддержкуgif формата. Даже при

учете того фактора, что данный стандарт был полностью совместим со второй версией гипертекстовой разметки HTML,его реализация быланамного сложней для браузеров тех времен, по причини несовершенства браузерных движком и ограниченности в ресурсах допустимых технологически для электронно-вычислительных машин того времени. ВерсияHTMLпод номером 3.1 официально никогда не предлагалась, но была доступна в репозиториях и имела множество недокументированных особенностей, распространяясь исключено для первооткрывателей и энтузиастов в этой сфере, и следующей версией данного стандарта HTML стала именно версия 3.2 ставшая намного популярней своей предшественницы по причине того, что многие ненужные функции и методы были исключены, но добавлены нестандартные элементы на тот момент времени, поддерживаемые исключительно браузерами Netscape Navigator и Mosaic, которые в свою очередь на тот момент времени пользовались широкой популярностью, по причине отлаженной работы, и быстрой соответственно тем вычислительным мощностям, обработкой информации и с учетом факторов сотрудничества со многими компаниями в данной сфере имели большую популярность нежели сторонние решения предлагаемые компаниями, как решение по умолчанию для операционных систем.

Далее немногим временем спустя была выпушена версия HTML 4.0 и в свою очередь произошла некоторая «очистка», как говорили многие специалисты и верстальщики, стандарта. Многие элементы, использованные в предыдущих версиях, начали считаться устаревшими и не рекомендованными для использования в дальнейшем. К примеру, частным случаем может являться, тег <font>, используемый при верстке сайтов и веб-сервисов для изменения свойств шрифта, отображаемого на странице. Данный тег был отмечен как устаревший и рекомендовалось использовать CSSстили для реализации задуманного. Это было одним из первых серьезных шагов для повсеместной интеграции CSSстилей и выноской данных в отдельные файлы с возможной реализацией подгрузки этих самых файлов перед отображением запрашиваемого контента собственно в браузере персонального компьютера.

Во втором квартале 1998 года Консорциум Всемирной паутины начал первичную работу над новым языком разметки, кардинально отличающимся по своей логике, но схожим по структуре и собственно основанным на HTML версии 4, но соответствующим общепринятому синтаксису XML. Новый язык получил название путем коллаборации используемых названий, а именно получил название XHTML. Первая тестовая версия XHTML 1.0 была одобрена в представлении Рекомендации консорциума Всемирной паутины от 26 января 2000 года. Планируемая версия XHTML 2.0 разрабатываемая на протяжении девяти лет должна была разорвать всякую совместимость со старыми версиями HTML и XHTML, но с учетом многих факторов и совокупностей 2 июля 2009 года Консорциум Всемирной паутины официально объявил, что полномочия группы, работающей над версиейXHTML2, истекают к концу 2009 года и дальнейшая работа над этим

проектом приостанавливается на неопределенное время. С учетом вышесказанного была остановлена вся дальнейшая разработка стандарта XHTML 2.0, без возможности ее возобновления по причине прошедшего времени и устаревания технологии. Так-как HTML является теговым языке разметки документов, то любой документ, файл, блок кода верстки на языке HTML представляет собой набор отдельных в то же время связанных между собой элементов, при условии использования отдельных свойств позиционирования и зависимости блочных элементов друг от друга, стоит учитывать, что начало и конец каждого элемента обозначается специальными пометками, как уже было сказано ранее - тегами. Предположительно все имеющиеся элементы могут быть пустыми, то есть не содержащими никакого текста, контента, либо других данных. Таким тегом можно назвать тег перевода строки
. В данном случае обычно не указывается закрывающий тег по причине его ненадобности дабы не засорять страницу лишними элементами. Кроме того, эти элементы могут иметь собственный, либо общие атрибуты, определяющие какие-либо их свойства и указываются в открывающем теге.

2.2 Каскадные таблицы стилей CSS

Каскадные таблицы стилей CSS– это формальный язык, применяемый к внешнему виду документа, который в свою очередь должен быть написан с использованием языка разметки. В основном используется как средство описания и оформления внешнего вида страниц глобальной сети интернет, написанных с помощью языков разметки XHTML и преимущественно HTML, но аналогично может применяться XML-документам любого рода и структуры, например, к векторным SVG файлам или XUL документам.

CSS активно используется создателями веб-страниц и сервисов, находящихся во всемирной паутине для задания вариативной цветовой гаммы, шрифтов, позиционирования отдельных блоков, а также других аспектов представления внешнего вида тех самых веб-страниц. Основной целью разработки CSS послужило разделение описания логической структуры веб-страницы, которое производится предпочтительно с помощью HTML или других языков разметки, в зависимости от описания внешнего вида веб-страницы, которое в свою очередь производится с помощью языка CSS. Такое разделение практически может увеличить доступность документа, предоставить качественный инструментал разработчикам, а также большую гибкость и возможность управления его абстрактным представлением, а также значительно уменьшить сложность и повторяемость в структурные содержимые веб-страницы и отдельных сервисов использующих гипертекстовую разметку HTML. Кроме того, каскадные таблицы стилей CSS позволяют представлять один и тот же документ с одинаковым внутренним наполнением в различных стилях или моделях вывода, таких как экранное

представление, либо печатное представление, а также чтение голосом, или при выводе устройствами, использующими к примеру шрифт Брайля.

На рисунке 2.3 продемонстрирован исходный код файла main.cssявляющегося частью проекта и отвечающего за корректное отображение и правильное позиционирование блоков и модулей на сайте с учетом всех размеров и вариативной цветовой гаммы.

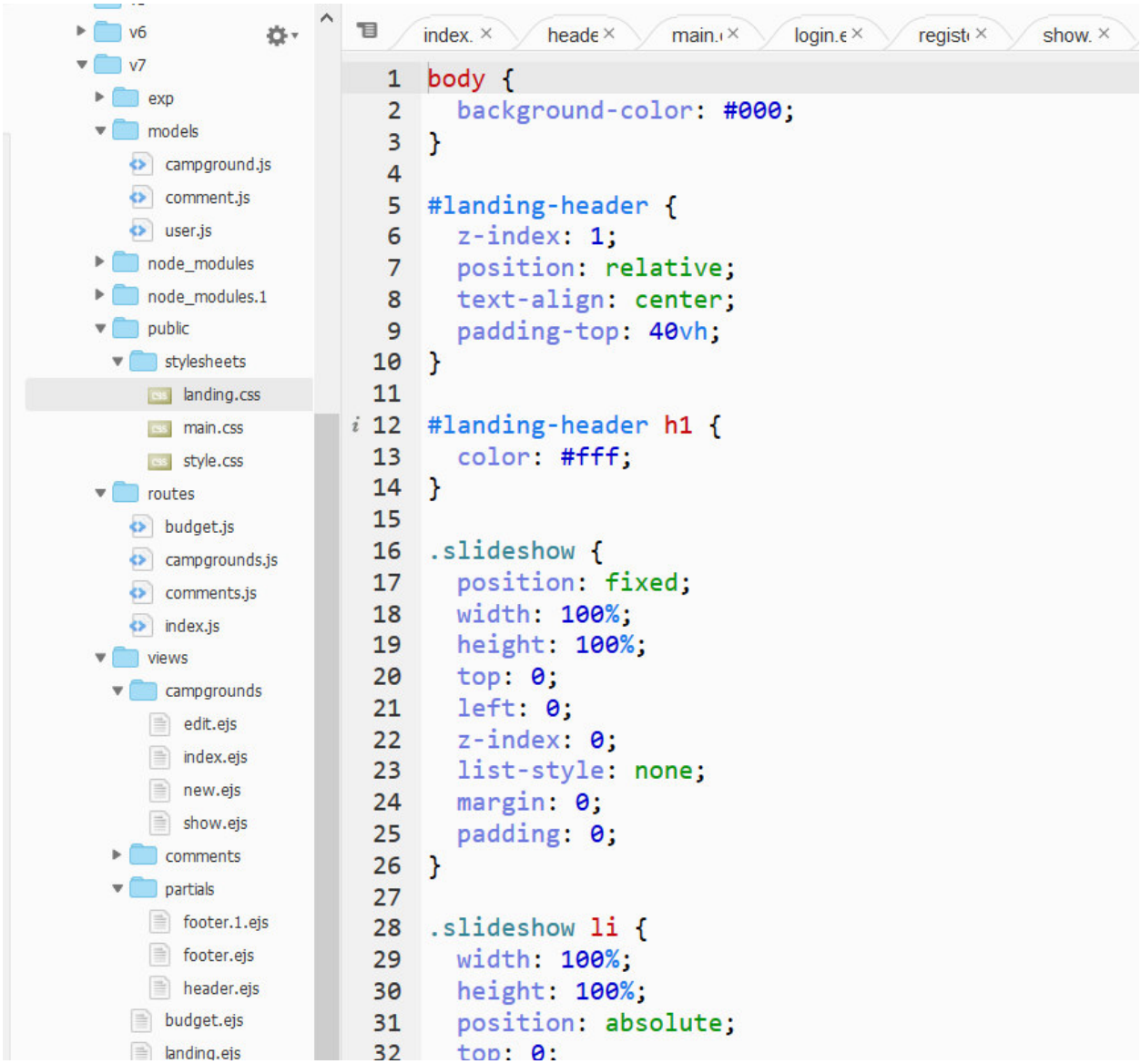


Рисунок 2.3 – Структура файла, содержащего стили CSS

Правила CSS соответственно пишутся на формальном языке CSS и располагаются собственно в таблицах стилей, то есть таблицы стилей содержат в себе правила CSS и ничего более. Эти таблицы стилей могут располагаться как в самом веб-документе помимо контента в нем содержащегося, внешний вид которого они описывают, так и в отдельных файлах, имеющих формат CSS.

Как показано на рисунке 2.4, практика интеграции CSS стилей напрямую в HTMLтеги очень удобно и зачастую применяется для создания уникальных правил, применяемых к одному блоку с информацией, в тег которой они заключены.

```
15
16  <div class="container">
17  <p style="
18      text-align: center;
19      font-size: 18px;
20      margin-top: -14px;
21      color: darkgray;
22  ">
23   Некий текст
24  </p>
25  </div>
26
```

Рисунок 2.4 – Интеграция CSS стилей в HTML тег

Когда таблица стилей находится в отдельном файле, что является хорошим тоном для общих стилей CSS, она может быть подключена к веб-документу, посредством специального тега <link>, располагающегося в этом же документе между тегами <head> и </head> совместно с остальными правилами подобного рода, а именно название страницы, подключение скриптов и стилей.

2.3 Сценарный язык программирования JavaScript

В веб-верстке помимо HTMLи CCS так же активно используются скрипты для реализации поставленных задач разной сложности. JavaScript-это прототипно-ориентированный сценарный язык программирования. Является реализацией языка ECMAScript, но со временем, учитывая постоянные обновления и дополнения перерос в независимую экосистему со своими независимыми форумами и библиотеками для реализации проектных задач. JavaScript как правило используется на практике в качестве встраиваемого в разметку языка для программного и управляемого доступа к объектам искомых приложений. На данный момент наиболее широкое применение находит в современных браузерах как язык сценариев для придания интерактивности и адаптивности веб-страницам и сервисам, использующим веб-технологии в решении собственных задач. Основные архитектурные черты присущие данному сценарному языку предпочтительно являются следующие:

- слабая и динамическая типизация;
- автоматическое управление памятью;
- прототипное программирование;

– функции как объекты первого класса.

На JavaScript большое влияние оказывали на протяжении долгого временного промежутка многие языки, при разработке была поставлена цель сделать язык синтаксисом и парадигмой похожим на Java, но в свою очередь при этом он должен быть лёгким для использования непрограммистами, как это реализовано в объектно-ориентированном языке программирования python. Языком JavaScript не владеет какая-либо компания или коммерческая организация, он использует принципы свободно программного обеспечения, распространяемого и позиционирующегося как бесплатный продукт с открытым кодом, что отличает его от рядапроприетарных языков программирования, использующих методы закрытого кода, либо закрытых библиотек, описанных лишь методами взаимодействия с данными.Название «JavaScript» является зарегистрированным товарным знаком компании Oracle Corporation, по причине коллаборации их товарного знака и наименования Java, вышеупомянутая компания выкупила права на владение и данным названием. Было множество предпосылок, означающих о выходе этого скриптового языка.

Еще в 1992 году компания Nombas, впоследствии приобретённая Openwave начала собственную разработку встраиваемого скриптового языка Cmm который, по сути и замыслу разработчиков, должен был стать достаточно абстрактным и мощным, чтобы заменить макросы и частное применение модулей, сохраняя при этом схожесть с объектно-ориентированным языком программирования Си, чтобы разработчикам в свою очередь не составляло труда изучить данный язык и с легкостью применять его при решений задач разного уровня абстрактности, логического структурирования и сложности. Главным отличием от Си была корректно реализованная работа с памятью в рамках обработки событий браузером. В новом языке всё управление памятью осуществлялось автоматически что дает ему огромное преимущество в гибкости и простоте перед вышеупомянутым аналогом что избавляло от следующих необходимостей и проблем, связанных с простотой в работе:

– необходимостьсоздание буфера;
– объявлять переменных;
– осуществлять преобразование типов.

С учетом вышеперечисленных особенностей языки сильно походили друг на друга. А именно: Cmm поддерживал стандартные функции и операторы Си, которые успешно реализованы в JavaScript, Cmm был переименован впоследствии и мел названиеScriptEase, поскольку исходное название звучало слишком негативно по мнению самих разработчиков, а упоминание в нём объектно-ориентированного языка программирования Си так сказать «отпугивало» людей по причине сложности данного языка в освоении и применении на практике. Также на основе упомянутого языка был создан проприетарный продукт CEnvi распространяющийся под собственной лицензией. В середине четвертого квартала, а именно в конце ноября 1995

года Nombas разработала высокоуровневую и специализированную версию CEnvi, встраиваемую в веб-страницы.

Страницы, которые предположительно можно было изменять с помощью сказанного скриптового языка, получили уникальное название Espresso Pages, иони в свою очередь демонстрировали использование этого скриптового языка для создания некой игры, а именно проверки пользовательского ввода в формы и создания после считывания и обработки введенных данных соответствующей анимации. Espresso Pages позиционировались как демоверсия и не являлись серьезным коммерческим решением, лишь призванная помочь представить себе, что получится, если в любой браузер будет внедрён язык Cmm с последующей реализацией документированных функций и максимальным коэффициентом полезного действия. Работали упомянутые страницы исключительно в 16-битовом Netscape Navigator того времени и только под управлением WindowsNTсовместимых системах того времени.

JavaScript по своему логическому структурированию является объектно-ориентированным языком программирования, но используемое в данном языке прототипирование обуславливает кардинальные отличия в работе с объектами по сравнению с традиционными класс-ориентированными языками программирования и методами в них практикующимися на постоянной основе. Кроме того, JavaScript имеет ряд уникальных и отличительных свойств, присущих именно функциональным языкам, что в понимании данного контекста говорит о следующих особенностях:

- функции как объекты первого класса;
- объекты как списки;
- карринг;
- анонимные функции;
- замыкания.

Если детально рассмотреть данные особенности позаимствованные и реализованные из других функциональных в свою очередь языков, то можно сделать вывод, что данные факторы придают языку дополнительную гибкость неприсущую аналогичным решениям от сторонних разработчиков.

Так же отдельно стоит упомянуть о главной особенности этого языка, благодаря которой он получил особую популярность. А именно это синтаксис, который очень благоприятно повлиял на становление и развитие JavaScriptделая его популярней с каждым годом с момента его запуска. Несмотря на схожий с Си синтаксис, JavaScript по сравнению со своим самым близким языком Си имеет коренные отличия, к которым относятся:

- объекты, с возможностью интроспекции;
- функции как объекты первого класса;
- автоматическое приведение типов;
- автоматическая сборка мусора;
- анонимные функции.

Все же JavaScript помимо строгой регистрозависимости идентификаторов и семантической схожести с языком программирования Лисп существуют недокументированные недостатки, а именно в данном языке отсутствуют такие полезные вещи как:

– модульная система: JavaScript по своей структуре и логике не предоставляет возможности управлять зависимостями и изоляцией областей видимости, что является одним из самых главных недостатков данного языка по мнению неофициального сообщества;

– стандартная библиотека: в частности, отсутствует единый, стандартизированный интерфейс программирования приложений по работе с файловой системой и каждая команда разработчиков используя средства back-endязыков программирования по-своему реализуют данную задачу, а также управлению потоками ввода-вывода данных и базовых типов для бинарных данных;

– стандартные интерфейсы к веб-серверам и базам данных, что привело к созданию собственных интерфейсов все с тем же использованием дополнительных возможностей back-endязыков;

– система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

Собственно, стоит отметить тот факт, что JavaScript используется в клиентской части веб-приложений, а именно клиент-серверных программах, в которых клиентом является сам браузер, установленный на персональном компьютере пользователя, а сервером является веб-сервер, имеющий распределённую специальным образоммежду сервером и клиентом логическую взаимосвязь.

Обмен информацией и всеми необходимыми данными в веб-приложениях и веб-сервисах происходит исключительно по сети. Одним из основных преимуществ данного подхода является тот факт, что клиенты, пользующиеся веб-браузерами, не зависят от конкретной операционной системы, поэтому все веб-приложения являются кроссплатформенными сервисами и данные решения с последующей их реализацией зависят исключительно от функциональных особенностей браузеров, работающих под этими самыми операционными системами. Также стоит упомянуть то, что JavaScript используется в AJAX, являющимся на данный момент времени популярным подходом к построению интерактивных пользовательских интерфейсов веб-приложений и веб-сервисов разного уровня и набора базовых функций и состоит в заключающемся, либо «фоновом» асинхронном обмене персональными данными браузера пользователя с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, то есть перезагружаются отдельные модули внутри страницы без перезагрузки всей страницы целиком и интерфейс веб-приложения становится быстрее, чем это происходит при традиционном подходе без применения AJAX инструментария.Так же стоит отметить что технология AJAXшироко применяется такими компаниями как Google, которая успешно реализует

мгновенный поиск информации и прогружает поисковую выдачу на той же странице на которой и был сделан запрос.

На рисунке 2.6 показан исходный код и скриншот программы, выполняемой в среде Seedи демонстрирующий базовый набор функциональных возможностей для построения логики небольшого приложения.

```
for (var i = 0; i < square.length; i++) {
    square[i].style.backgroundColor = colors[i];

    square[i].addEventListener("click", function () {
        var clickedColor = this.style.backgroundColor;

        if (clickedColor === pickedColor) {
            message.textContent = "Correct!";
            changeColorToCorrect(clickedColor);
            btnNewGame.textContent = "Play Again?";
        }
        else {
            this.style.backgroundColor = "#232323";
            message.textContent = "Try again";
        }
    })
}

function changeColorToCorrect(correctColor) {
    headerTop.style.backgroundColor = correctColor;
    for (var i = 0; i < square.length; i++) {
        square[i].style.backgroundColor = correctColor;
    }
}

function pickColor() {
    var random = Math.floor(Math.random() * colors.length);
    return colors[random];
}

function colorsRandomizer(num) {
    var arr = [];
    for (var i = 0; i < num; i++) {
        arr[i] = doRandomColors();
    }
    return arr;
}

function doRandomColors() {
    var r = Math.floor(Math.random() * 256);
    var g = Math.floor(Math.random() * 256);
    var b = Math.floor(Math.random() * 256);
    return "rgb(" + r + ", " + g + ", " + b + ")";
}
```

Рисунок 2.5 – JavaScript-программа, выполняемая в среде Seed

3 Проектирование сайта

Для проектирования сайта будет использоваться Node или Node.js. Это программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

В 1996 году в компании Netscape были попытки создания серверного JavaScript (Server-side JavaScript — SSJS), однако технология не получила распространения.

Node.js разработал Райан Даль (англ. Ryan Dahl) в 2009 году после двух лет экспериментирования над созданием серверных веб-компонентов. В ходе своих исследований он пришёл к выводу, что вместо традиционной модели параллелизма на основе потоков следует обратиться к событийно-ориентированным системам. Эта модель была выбрана из-за простоты, низких накладных расходов (по сравнению с идеологией «один поток на каждое соединение») и быстродействия. Целью Node является предложить «простой способ построения масштабируемых сетевых серверов». Для того, чтобы подключить к нашему проекту другие библиотеки, нужно воспользоваться npm.

npm - это менеджер пакетов для Node. Две основных обязанностей менеджера пакетов - это установка пакетов и управление зависимостями. Это быстрый, надежный и легкий менеджер пакетов. Именно благодаря установщику пакетов npm экосистема Node так быстро развивается. Также плюсом является то, что npm входит в состав Node, поэтому нет необходимости устанавливать отдельно. Ознакомившись с npm мы приступаем к установке библиотеки Express.

Express – это минимализированная и гибкая библиотека для веб-приложений на node.js, которая предоставляет надежный набор функций для создания одностраничных или многостраничных гибридных веб-приложений. Главные плюсы использования Express:

Минимализм . Это один из самых привлекательных аспектов Express. Часто разработчики фреймворков забывают, что обычно «чем меньше, тем лучше». Философия Express заключается в том, чтобы обеспечить минимальный слой между вашим кодом и сервером. Это означает, что время,

которое затрачивается на написание кода, уменьшится, позволяя вам полностью выразить свои идеи, и в то же время, обходясь без огромного кода.

Гибкость. Еще один ключевой аспект фреймворка Express заключается в том, что Express является расширяемым. Изначально Express предоставляет вам минимальную структуру, и вы, по мере необходимости, можете добавлять различные части Express-функций, заменяя все, что не соответствует вашим потребностям. Это позволяет максимально оптимизировать проект. Очень часто первая задача, с которой сталкиваются при разработке проекта - это трата драгоценного времени на удаление ненужных или неиспользуемых функций, которые не соответствуют требованиям заказчика. В Express же используется противоположный подход, позволяющий добавлять то, что вам нужно, когда вам это нужно.

Одностраничные веб-приложения. Это относительно новая идея и тренд. Вместо полноразмерного веб-сайта, требующего сетевого запроса каждый раз, когда пользователь переходит на другую вкладку внутри сайта, одностраничное веб-приложение сразу загружает весь сайт в браузер клиента. После этого навигация происходит гораздо быстрее, потому что с сервером практически отсутствует связь, так как не приходится обращаться к серверу при каждом новом запросе. Разработка одностраничных сайтов облегчается благодаря использованию популярных библиотек, таких как Angular или Ember.

Многостраничные и гибридные веб-приложения. Многостраничные веб-приложения – это более традиционный подход к веб-сайтам. Каждая страница на сайте представляет собой отдельный запрос к серверу и после каждого запроса загружается новая вкладка на сайте. Это значит, что у разработчиков теперь есть больше опций - вы можете сами решить какие части вашего контента должны быть отправлены в виде одностраничного приложения, и какие части должны быть отправлены в виде многостраничного веб-приложения. Такой подход называется гибридным.

3.1 Разработка функционала проекта

Первым делом мы устанавливаем Express с помощью установщика пакетов npm. Для этого нужно открыть консольный терминал node.js. Самое первое, что нам необходимо сделать – это создать рабочий каталог, где мы и будем размещать наш проект. Создаем каталог для своего приложения и делаем его своим рабочим каталогом.

```
$ mkdir myapp
$ cd myapp
```

Рисунок 3.1 – Создание директории

Файл package.json содержит в себе информацию о вашем приложении: название, версия, зависимости и тому подобное. Любая директория, в которой есть этот файл, интерпретируется как Node.js-пакет. С помощью команды npm init создаем файл package.json для своего приложения.

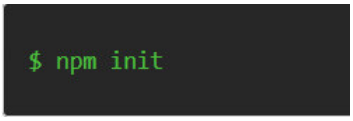


Рисунок 3.1.2 – Создание директории

Эта команда выдает целый ряд приглашений, например, приглашение указать имя и версию вашего приложения. На данный момент, достаточно просто нажать клавишу ВВОД, чтобы принять предлагаемые значения по умолчанию для большинства пунктов, кроме следующего:

Рисунок 3.1.3 – Создание директории

Это значит какой файл будет входной точкой в наше приложение. Можно ввести app.js или любое другое имя главного файла по своему желанию. Так как нам необходимо назвать главный файл app.js, переименовываем с index.js на app.js, и нажимаем клавишу ВВОД, чтобы применить имя файла.
Теперь устанавливаем Express в каталоге app.

Рисунок 3.1.4 – Установка Экспресс

После установки Express нам нужно сохранить это в каталоге зависимостей, так как если этого не делать, то нужно будет каждый раз проделывать эти пункты, когда мы закрываем и открываем проект.

Рисунок 3.1.5 – Сохранение Экспресс в зависимость

После установки всех зависимостей мы приступаем к разработке самого проекта. Первым делом необходимо внедрить Express в наш код.

```
var express = require("express");
var app = express();
```

Рисунок 3.1.6 – Добавление Экспресс в код

Когда уже Express установлен, нам теперь нужно использовать bodyParser. Это нужно для того, чтобы обрабатывать POST запросы, так как технология node.js из коробки умеет обрабатывать только GET запросы. Это очень важно, потому что если в вашем проекте будут формы, которые будут вносить что-то в базу данных или менять уже существующую информацию в базе данных, то нам нужна возможность как-то извлекать эти данные с форм. Подключаем bodyParser и говорим приложению использовать его.

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

Рисунок 3.1.7 – Добавление функции body-parser

В самом низу нашего проекта создаем app.listen. Здесь создается web-сервер, «слушающий» порт 3000. Если всё прошло без ошибок, и сервер успешно запустился, то программа в терминале выдаст: «Сервер успешно запущен!!!». Это значит, что у нас есть выделенный сервер, где мы и будем размещать наш проект.

```
app.listen(3000, process.env.IP, function() {
    console.log("Сервер успешно запущен!!!");
});
```

Рисунок 3.1.8 – Создание сервера

После настройки сервера, нам нужно разбить наш проект на представления. Это является основой шаблона проектирования MVC.

```
app.set("view engine", "ejs");
```

Рисунок 3.1.9 – Объявляем расширение ejs в качестве стандартного

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Под компонентом следует понимать отдельную часть кода. Каждая часть из которых играет одну из ролей контроллера, модели или представления. Модель служит для извлечения и манипуляций данными приложения в базе данных, Представление отвечает за видимое пользователю отображение этих данных (то есть, в применении к вебу, формирует отдаваемый сервером браузеру пользователя HTML/CSS), а Контроллер управляет всем этим.

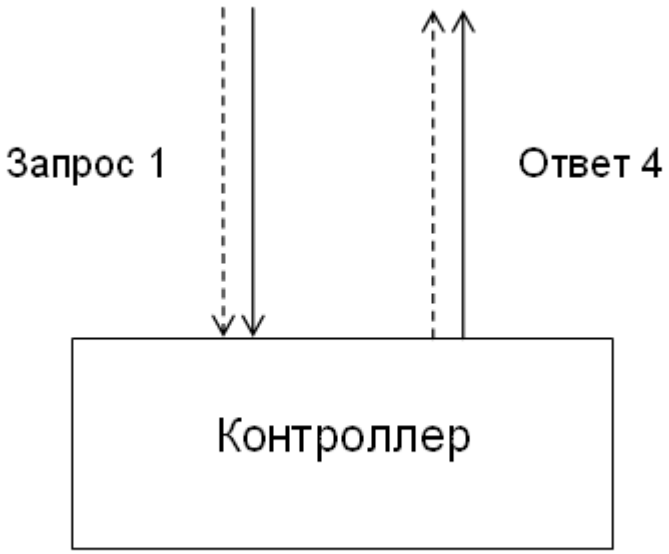


Рисунок 3.1.10 – Контроллер

На этом и последующем рисунках пунктирными линиями показана управляющая информация (такая, например, как ID запрашиваемой записи или товара в магазине), а сплошными – данные приложения (которые могут храниться в БД, или в виде файлов на диске). Запрос и ответ используют HTTP, поэтому можно считать, что на этом рисунке пунктиром обозначены заголовки HTTP-запроса и ответа, а сплошными линиями – их тела.

Получив Запрос 1, Контроллер его анализирует, и в зависимости от результатов обработки может выдать следующие варианты ответа (почему ответ имеет номер 4, станет понятно из следующих рисунков):

1. Сразу выдать ответ об ошибке (например, при запросе несуществующей страницы отдать только HTTP-заголовок «404 Not found»)

2. Если поступивший Запрос 1 признан корректным, то, в зависимости от того, является он запросом на просмотр или на модификацию данных, Контроллер вызывает соответствующий метод Модели, такой как **Save** или **Load** (Запрос 2 на Рис. 3.1.11).

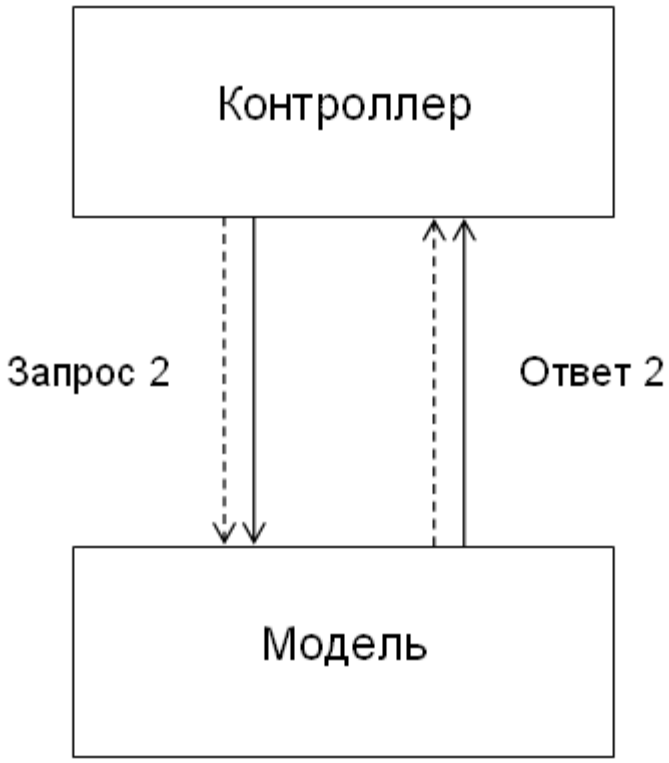


Рисунок 3.1.11 – Взаимодействие модели с контроллером

Важное замечание: концепция MVC не только не привязана к какому-то конкретному языку программирования, она также не привязана и к используемой парадигме программирования.

В зависимости от полученного от Модели «Ответа 2» Контроллер решает, какое из Представлений вызвать для формирования итогового ответа на изначальный «Запрос 1»:

3.1. В случае неудачи – Представление для сообщения об ошибке

3.2. В случае успеха – Представление для отображения запрашиваемых данных либо сообщения об их успешном сохранении (если Запрос 1 был на изменение данных).

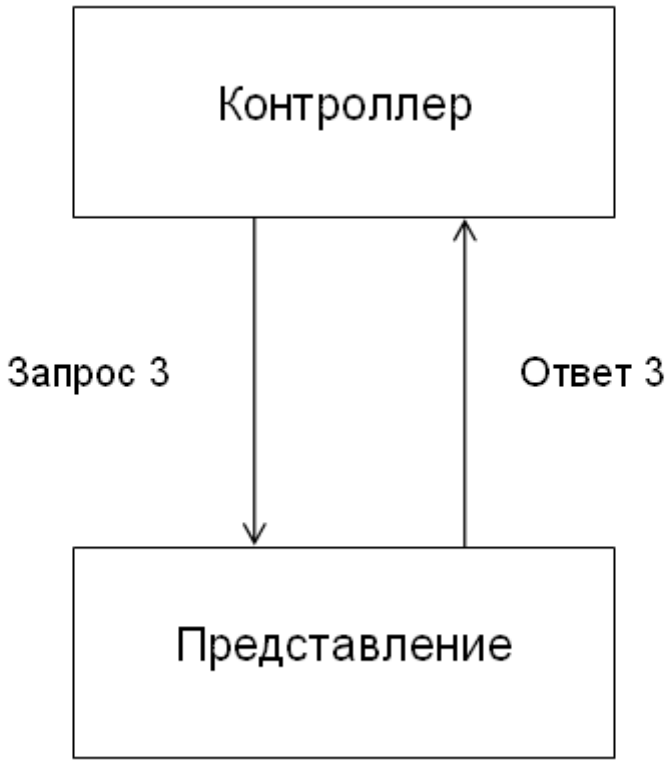


Рисунок 3.1.12 Представление

Контроллер проверяет входные данные на предмет «общей» (т.е. независящей от конкретного запроса) корректности. Соответствие требованиям Модели к валидности сохраняемых данных проверяет соответствующий метод Модели, а права доступа – метод «Access» отдельного класса «User».

Для вызова Представления в node.js иногда проектируется специальный класс (а то и несколько классов), например, «View» (это часто встречается в описаниях MVC в реализации того или иного фреймворка), однако это не является требованием MVC.

Файлы Представлений часто называют шаблонами, а при использовании так называемых шаблонизаторов роль Представления играет сам шаблонизатор, а шаблоны, то есть файлы, содержащие непосредственно HTML-разметку часто называют «layouts».

Займемся разработкой шапки сайта или хедером. Хедер выступает одним из ключевых элементов оформления сайта. Он оказывает прямое влияние на внешнюю привлекательность ресурса. Шапка играет важную роль при оптимизации сайта, поскольку именно в хедере размещены ключевые сведения о ресурсе.

Задача хедера давать пользователям ответы на основные вопросы: что за бренд представлен, какие товары и услуги предлагаются пользователям, как

связаться с сотрудниками компании, есть ли актуальные акции, какие предоставляются гарантии.

```
<!DOCTYPE html>
<html>
    <head>
         <link rel="stylesheet"
         href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
        <title>Fin-apps</title>
    </head>
    <body>

<nav class="navbar navbar-default">
    <div class="container-fluid">
        <div class="navbar-header">
            <a class="navbar-brand" href="/">Fin-apps</a>
        </div>
        <div class="collapse navbar-collapse">
            <ul class="nav navbar-nav navbar-right">
                <li><a href="/">Вход</a></li>
                <li><a href="/">Регистрация</a></li>
                <li><a href="/">Выход</a></li>
            </ul>
        </div>
    </div>
</nav>
```

Рисунок 3.1.13 – Код навигационного бара

Для создания фронт-энд части будем использовать библиотеку Bootstrap. Это открытый HTML, CSS и JS фреймворк, который используется веб-разработчиками для быстрого создания адаптивных дизайнов сайтов.

Фреймворк Bootstrap используется не только независимыми разработчиками, но и целыми компаниями. Основная область его применения – это разработка фронтенд составляющих сайтов и интерфейсов. Среди аналогичных систем (Foundation, UIkit, Semantic UI, InK и др.) фреймворк Bootstrap является самым популярным.

В сущности, Bootstrap - это набор файлов (CSS и JavaScript). После подключения этих файлов к странице вам станут доступны для верстки дизайна большое количество классов и готовых компонентов. Используя их можно очень быстро и качественно создать современный адаптивный дизайн сайта.

Классы Bootstrap можно разбить на 3 большие группы:

- Классы для создания сетки (адаптивного макета страницы);
- Классы для стилизации контента (текста, кода, изображений, таблиц и другой информации);

– Служебные классы (для решения наиболее часто встречающихся вспомогательных задач, таких как выравнивание, управление отображением, добавление границ и др.).

Кроме классов во фреймворке Bootstrap имеются ещё и компоненты (готовые объекты интерфейса). Это кнопки, хлебные крошки, формы, навигационные меню, выпадающие списки, всплывающие панели и др.

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

Рисунок 3.1.14 – Создание видов

Создаем директорию views. Внутри этой папки создаём ещё одну директорию partials. В этой папке мы будем хранить наш созданный выше файл шапки Header.ejs.

```
<% include partials/header %>
```

Рисунок 3.1.14 – Подключение шапки

Когда мы будем создавать новую страницу в проекте, нам уже не будет необходимости каждый раз повторять и заново вставлять код с шапкой. Всё что нам нужно будет сделать – это при помощи команды include добавить шапку в нашу создаваемую страницу.

```
    </body>
</html>
```

Рисунок 3.1.15 – Закрывающие скопки

В файле Footer.ejs не будет много кода, потому что весь код мы разместили в файле Header.ejs. А футер часть будет хранить только закрывающие теги, так как без них каркас шапки не будет работать. Этот код мы размещаем в самом низу страницы.

```
<% include partials/footer %>
```

3.2 Создание моделей в базе данных

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

Со времен динозавров было обычным делом хранить все данные в реляционных базах данных (MS SQL, MySQL, Oracle, PostgresSQL). При этом было не столь важно, а подходят ли реляционные базы данных для хранения данного типа данных или нет.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

3.2.1 Создание каталога для БД и запуск MongoDB

После установки надо создать на жестком диске каталог, в котором будут находиться базы данных MongoDB.

В ОС Windows по умолчанию MongoDB хранит базы данных по пути «C:\data\db», поэтому, если вы используете Windows, вам надо создать соответствующий каталог.

Итак, после создания каталога для хранения БД можно запустить сервер MongoDB. Сервер представляет приложение mongod, которое находится в папке bin. Для этого запустим командную строку в Windows и там введем соответствующие команды. Для ОС Windows это будет выглядеть так:

:

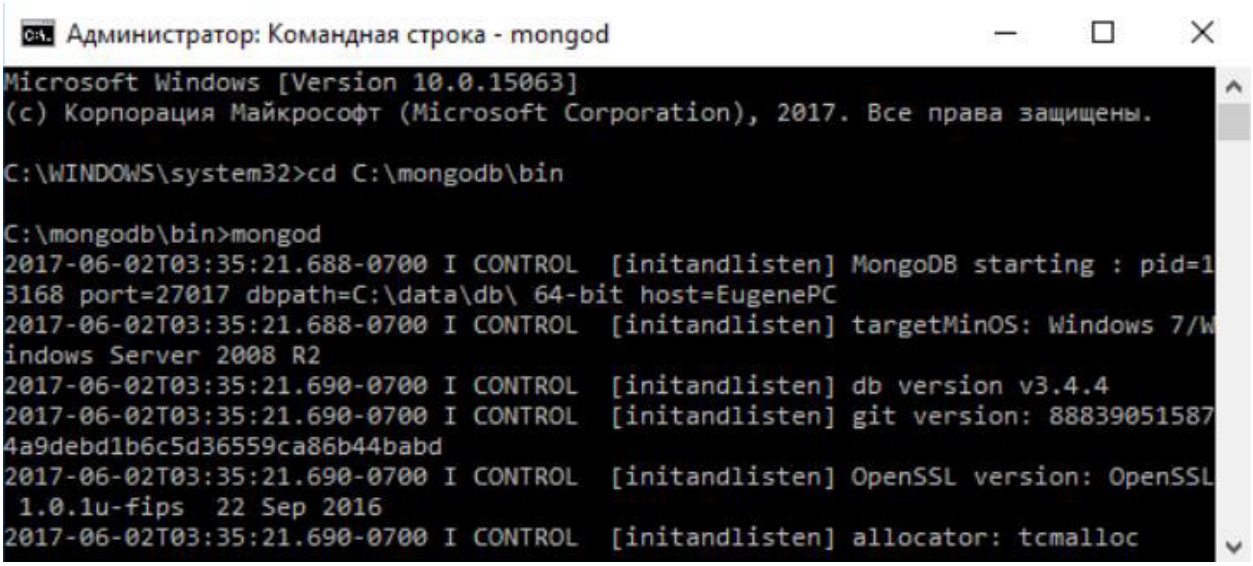


Рисунок 3.2 – Командная строка

Командная строка отобразит нам ряд служебной информации, например, что сервер запускается на localhost на порту 27017(это стандартный порт на котором работает MongoDB).

И после удачного запуска сервера мы сможем производить операции с бд через оболочку mongo. Эта оболочка представляет файл mongo.exe, который располагается в выше рассмотренной папке установки. Запустим этот файл:

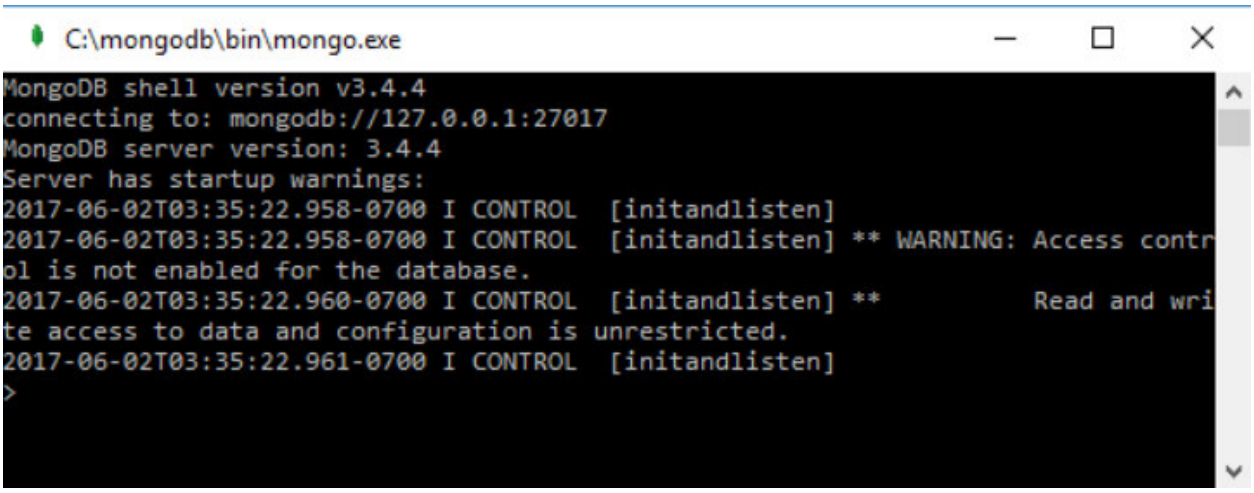


Рисунок 3.2.1 – Запуск базы данных

Это консольная оболочка для взаимодействия с сервером, через которую можно управлять данными. Второй строкой эта оболочка говорит о подключении к серверу mongod.

Теперь поизведем первые действия в создании и проверке БД. Введем в mongo последовательно следующие команды и после каждой команды нажмем на Enter:

```
1 use myapp
2 db.users.save( { name: "Alisher" } )
3 db.users.find()
```

Рисунок 3.2.2 – Работа в БД

Первая команда use myapp устанавливает в качестве используемой базу данных myapp. Даже если такой бд нет, то она создается автоматически. И далее db будет представлять текущую базу данных - то есть базу данных myapp. После db идет users - это коллекция, в которую затем мы добавляем новый объект. Если в SQL нам надо создавать таблицы заранее, то коллекции MongoDB создает самостоятельно при их отсутствии.

С помощью метода db.users.save() в коллекцию users базы данных myapp добавляется объект { name: "Alisher" }. Описание добавляемого объекта определяется в формате JSON. То есть в данном случае у объекта определен один ключ "name", которому сопоставляется значение "Alisher". То есть мы добавляем пользователя с именем Alisher.

Если объект был успешно добавлен, то консоль выведет результа в виде выражения WriteResult({ "nInserted" : 1 }).

```
> use myapp
switched to db myapp
> db.users.save( { name: "Alisher" } )
WriteResult({ "nInserted" : 1 })
> db.users.find()
{ "_id" : ObjectId("5cdbde928f236ba118f18110"), "name" : "Alisher" }
```

Рисунок 3.1.14 – Просмотр результатов в БД

А третья команда db.users.find() выводит на экран все объекты из бд myapp.

3.3 Создание Модели

Первым делом нам необходимо создать папку models, где будут храниться все наши модели.

```
$ mkdir models
```

Рисунок 3.3 – Создание директории Моделей

Модель определяет логическую структуру базы данных и в корне определяет, каким образом данные могут храниться, организовываться и обрабатываться.

Мы только, что создали модель. Так как наш проект связан с туристическими местами, то наша первая модель будет хранить информацию об имени этого места, изображения, описание, расположение, координаты на карте, и стоимость поездки.

```
module.exports = mongoose.model("Campground", campgroundSchema);
```

```
var campgroundSchema = new mongoose.Schema({
    name: String,
    image: String,
    description: String,
    location: String,
    lat: Number,
    lng: Number,
    cost: Number,
    author: {
        id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        },
        username: String
    },
    comments: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Comment"
        }
    ]
});
```

Рисунок 3.3.1 – Схема главной таблицы

В конце экспортируем созданную модель в модуль. Основная концепция модуля заключается в том, что он инкапсулирует логически связанный между собой код в единый блок. Создание модуля заключается в помещении всех взаимосвязанных между собой функций в один единый файл.

Далее мы создаём ещё одну модель. Модель пользователя.

```
var userSchema = new mongoose.Schema({
    username: String,
    password: String
});
```

Рисунок 3.3.2 – Схема пользователя в БД

Модель пользователя будет хранить информацию об имени пользователя и пароле. Но стоит отметить одну очень важную вещь – пароль,

если всё оставить как есть, будет в храниться в базе данных в точно таком же виде, как и было написано при вводе, то есть в незашифрованном виде.

```
> db.User.insert({name: "Alisher", password: "adminadmin"})
WriteResult({ "nInserted" : 1 })
> db.User.find()
{ "_id" : ObjectId("5cdbf75cfdcfdf9d88969a91"), "name" : "Alisher", "password" : "adminadmin" }
>
```

Рисунок 3.3.3 – Просмотр результатов в БД

Это не соответствует стандартом безопасности. Так как MongoDB является относительно новым языком базы данных, то он предоставляет методы зашифровки как раз для таких случаев. Для этого нужно всего лишь добавить в наш код метод passportLocalMongoose.

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

После этого каждая запись с паролем в базе данных будет иметь зашифрованный вид.

```
> db.users.find()
{ "_id" : ObjectId("5cc310f7d01272142044fba7"), "user
name" : "Alisher", "salt" : "0daf7e31fc3b58dcabb824b7
c1693c37f00cd712b59db0a952f248a0c0d84fbe", "hash" : "
2f04a14e9adb29e6ff08412e7833fb5cb1405faa0bfbd18452c08
9389ce4dc9f5b0e5915a1a3f242d76b4d794db0c39ff7d04d3e3a
048c6012138c96c88f99108077dae090fcc5677a2e5eecbab7d19
bcf8300d94ab3c915d4673adea19ce69d8699703d7b21171fd5a4
bf7639f75039eee323982b44019733457c3079f314b39bde1c705
7baa7e0c55ac0c94cd90fbe00f02e0363b2159bd069dfe2e35333
dcb965cea85fe9299fe6e18e35dd8b94d073ba33fadbeeafc3948
65e70fb357702a000bc6de79b9dd4931f024b251b2d270837074e
59bb9b5eb95b294570f3622de3638569a098ed738c22461f76288
e094d068d08674121daebceb82226bb2c3c27d46863a9355d976d
298f17de2a7a0b425a7e80c7f8567318edc9c420eb515922305bd
27788b6b3fe5b3c6b6a0295c3b2c5195f5dcbbb5944cf79a643f9
597a8a9bc308b7c08e954d19f8f916fa33f18f2926cb9ce258332
a540164250cd80abc", "__v" : 0 }
```

Рисунок 3.3.4 – Шифровка

Создав модели мест и пользователей, теперь нам нужно создать модель комментария под новостью.

```
var commentSchema = new mongoose.Schema({
    text: String,
    author: {
        id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        },
        username: String
    }

});
```

Рисунок 3.3.4 – Создание модели комментариев

В модели комментария помимо самого текста есть поле author, который по ссылке User связан с моделью пользователя, которую мы создавали ранее.

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

Рисунок 3.3.3 – Экспортирование

В самом конце мы экспортируем созданную только что модель в модуль, чтобы можно было обращаться к схеме пользователя с любого места внутри нашего проекта.

3.3.1 Создание Контроллера.

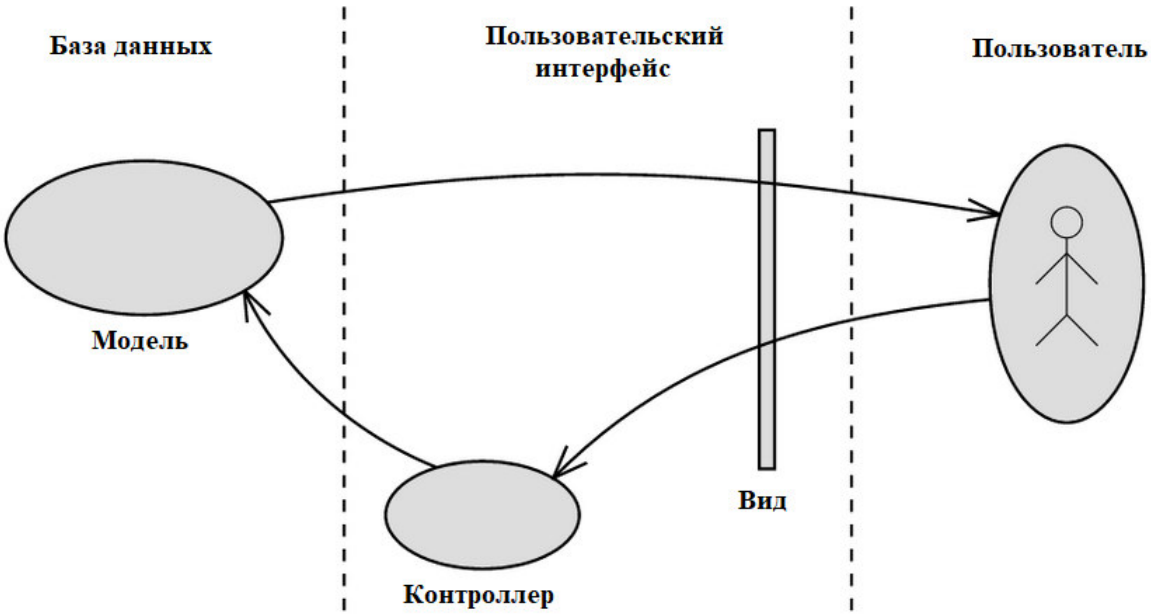


Рисунок 3.4 – Взаимодействие ПО

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

```
var express          = require("express"),
    router           = express.Router(),
    Campground       = require("../models/campground"),
    NodeGeocoder     = require('node-geocoder');

var options = {
  provider: 'google',
  httpAdapter: 'https',
  apiKey: process.env.GEOCODER_API_KEY,
  formatter: null
};

var geocoder = NodeGeocoder(options);
```

Рисунок 3.4.1 – Подключение контроллера

Здесь мы создаем обработчик GET запроса на страницу, где отображаются все места. Функция Campground.find обращается к базе данных.

```
router.get("/", function(req, res) {
    Campground.find({}, function(err, campgrounds) {
        if (err) {
            console.log(err);
        }
        else {
            res.render("campgrounds/index", { campgrounds: campgrounds,
            currentUser: req.user })
        }
    });

});
```

Рисунок 3.4.2 – Обработчик

Если записи в базе данных были найдены, то этот контроллер передаёт все данные нашему представлению(view).

```
<div data-aos="fade-up"  data-aos-duration="3000" class="row text-center" style="display: flex; flex-wrap:wrap;">
    <% campgrounds.forEach(function(campgrounds){ %>
        <div class="col-md-3 col-sm-6">
                <div  class="thumbnail">
                    <img id="thumba" src="<%= campgrounds.image %>">
                    <div class="caption">
                        <%= campgrounds.name %></h4>
                    </div>
                    <p>
                        <a href="campgrounds/<%=campgrounds._id %>" class="btn btn-primary">More info</a>
                    </p>
                </div>
        </div>
    <% }) %>
```

Рисунок 3.4.3 – Отображение найденных результатов

Через цикл forEach мы отображаем на странице все записи, которые были переданы на представление через контроллер.

Переменную campgrounds мы определили в нашем контроллере. «{ campgrounds: campgrounds }». Благодаря этому, теперь мы можем в нашем представлении напрямую обращаться к нашей модели туристических мест в базе данных. В рисунке(**!!!!!!**) мы написали код, который выводит на главную страницу имя и изображения всех наших туристических мест из базы в виде маленькой превьюшки.

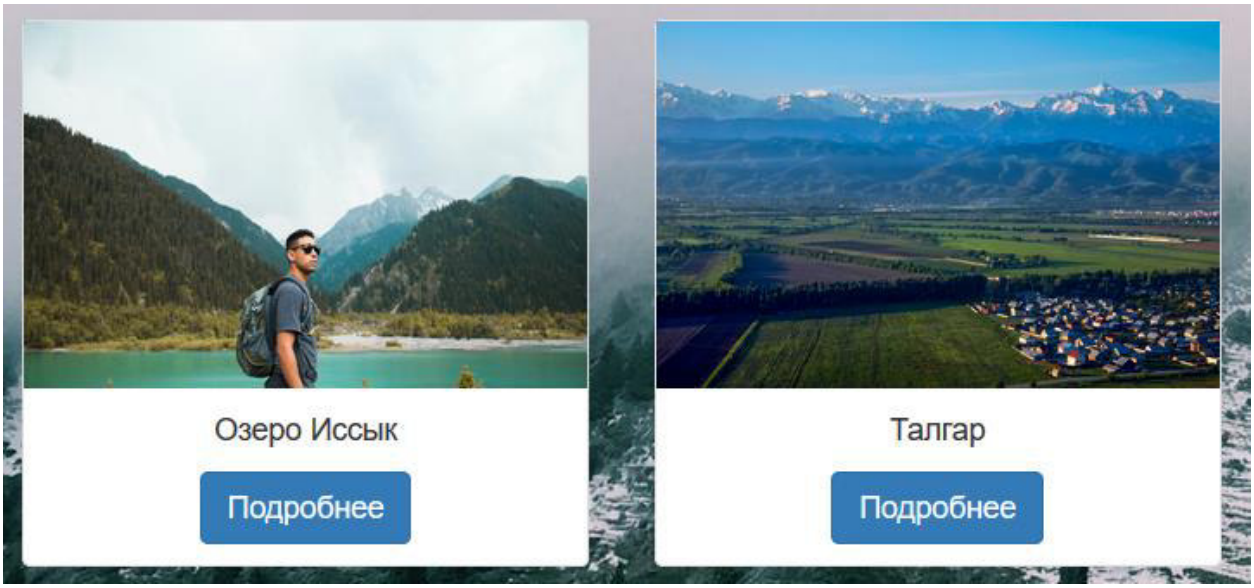


Рисунок 3.4.4 – Выведение на главной странице

После корректного выведения на главной странице, нам необходимо реализовать функционал редактирования. Так как в ходе эксплуатации не раз возникнет ситуация, когда необходимо отредактировать или обновить уже имеющуюся запись в базе данных.

Первым делом мы создаем GET запрос на страницу, где будет размещаться форма для редактирования.

```
router.get("/:id/edit", checkCampgroundOwnership, function(req, res) {
        Campground.findById(req.params.id, function(err, foundCampground) {
                    res.render("campgrounds/edit", { campground: foundCampground });
                });
});
```

Рисунок 3.4.5 – Создание страницы для редактирования

В качестве параметра метод findById передаёт id той записи, которую мы собираемся отредактировать. И функция будет искать в базе данных только ту запись, id которой соответствует той, что соответствует id в параметре.

Создаём файл в директории views, где будет форма для редактирования записи.

```
$ touch views/edit.ejs
```

Рисунок 3.4.6 – Создание файла, гду будет храниться стр. для редактирования

Внутри этого файла мы сначала добавляем наш хэдер файл, который отвечает за размещение шапки. И также выводим название места.

```
<% include ../partials/header %>
<div class="container">
<h1 style="text-align: center;">Edit "<%=campground.name%>"</h1>
```

Рисунок 3.4.7 – Подключение хэдер файла

Создаём теперь саму форму. Для того, чтобы обновить запись в базе данных нам нужен метод PUT. Но HTML не принимает PUT запросы из коробки, поэтому нужно дописать ?_method=PUT

```
<form action="/campgrounds/<%= campground._id %>?_method=PUT" method="POST" style="width: 30%; margin: 25px auto;">
    <div class="form-group">
        <input class="form-control" type="text" name="campground[name]" value="<%= campground.name %>">
    </div>
    <div class="form-group">
        <input class="form-control"  type="text" name="campground[image]" value="<%=campground.image%>">
    </div>
    <div class="form-group">
        <label for="description">Description</label>
        <textarea class="form-control" id="description" name="campground[description]" ><%=campground.description%>
    </div>
    <div class="form-group">
        <label for="price">Цена</label>
        <input id="price" class="form-control" type="number" name="campground[cost]" value="<%= campground.cost %>"
    </div>
    <div class="form-group">
        <label for="location">Location</label>
        <input class="form-control" type="text" name="location" id="location" value="<%= campground.location %>">
    </div>
    <div class="form-group">
        <button class="btn btn-lg btn-primary btn-block">Submit</button>
    </div>
    <a href="/campgrounds">Вурнуться назад</a>
</form>
```

Рисунок 3.4.8 – Создание формы

Согласно шаблону REST каждый HTTP метод должен иметь только тот метод, который соответствует его прямому назначению. REST - это общие принципы организации взаимодействия приложения или сайта с сервером через протокол HTTP. Главные принципы REST:

– Независимость от состояния (Statelessness). Сервер не должен запоминать состояние пользователя между запросами — в каждом запросе передаётся информация, идентифицирующая пользователя (например, token, полученный через OAuth-авторизацию) и все параметры, необходимые для выполнения операции. Независимость от состояния означает, что данные, возвращаемые определенным вызовом API, не должны зависеть от вызовов, сделанных ранее;

– Многоуровневая архитектура (Layered System). Многоуровневая архитектура означает, что клиент не знает, является ли сервер, который отвечает, на самом деле конечным сервером, который обслуживает ресурс. Это является отличным принципом для обеспечения балансировки нагрузки и предоставления общих КЭШей;

– Единый унифицированный программный интерфейс. К примеру, для получения списка блогов мы используем URL вида: */videos.com/blogs*, а для получения информации о конкретном фильме ваш URL будет: */videos.com/blogs/1*);

– Кэшируемая архитектура (Cacheable)**.** Ответ сервера может быть кэширован на определенный период времени и использоваться повторно без новых запросов к серверу;

– Удобное представление данных. В качестве представления данных объекта передаются данные в формате JSON или XML.

4 основных типа HTTP запросов REST-архитектуры:

– GET — для получения (чтение);
– POST — для создания;
– PUT — для изменения;
– DELETE — для удаления.

HTTP статус коды указывают на результат HTTP запроса:

– 1XX — информационный;
– 2XX — успешное выполнение;
– 3XX — перенаправление;
– 4XX — ошибка клиента;
– 5XX — ошибка сервера.

Итоговое представление страницы редактирования выглядит следующим образом.

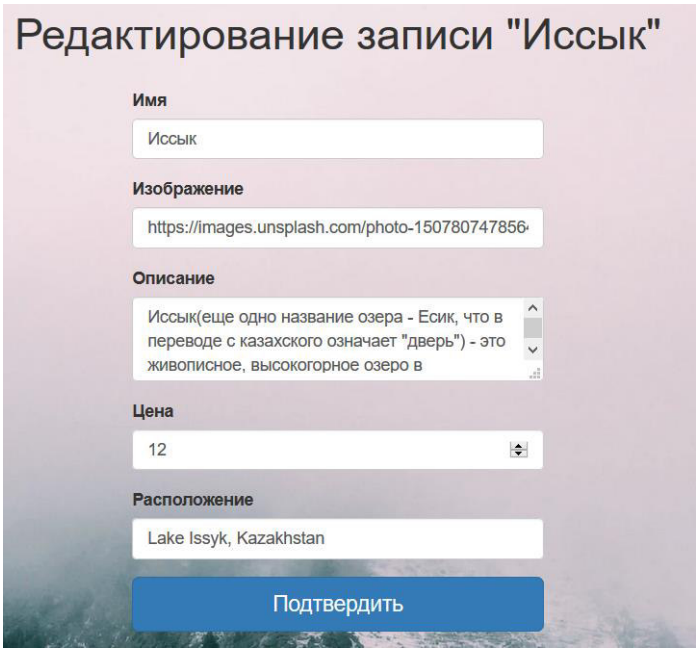


Рисунок 3.4.9 – Созданная страница редактирования

Реализовав форму в странице редактирования записи, теперь нам необходимо в нашем контроллере написать код, который и будет выполнять всю работу, а именно – находить нужную нам запись и обновлять все измененные данные в ней. Для реализации данного контроллера согласно принципам REST нам нужно использовать метод PUT, так как именно HTTP метод PUT отвечает за изменение записей в базе данных.

```
105 router.put("/:id", checkCampgroundOwnership, function(req, res){
106   geocoder.geocode(req.body.location, function (err, data) {
107     if (err || !data.length) {
108       req.flash('error', 'Invalid address');
109       return res.redirect('back');
110     }
111     req.body.campground.lat = data[0].latitude;
112     req.body.campground.lng = data[0].longitude;
113     req.body.campground.location = data[0].formattedAddress;
114 
115     Campground.findByIdAndUpdate(req.params.id, req.body.campground,
116     function(err, campground){
117         if(err){
118             req.flash("error", err.message);
119             res.redirect("back");
120         } else {
121             req.flash("success","Successfully Updated!");
122             res.redirect("/campgrounds/" + campground._id);
123         }
124     });
125   });
126 });
```

Рисунок 3.4.10 – Контроллер, отвечающий за измение записи в БД

Метод findByIdAndUpdate – это метод MongoDB. Этот метод в качестве параметров берёт id и тело запроса(req.params.id). В теле запроса хранятся все данные, которые были изменены. В случае ошибки, контроллер выдаст ошибку и перенаправит нас обратно, а если всё прошло успешно, то команда res.redirect перенаправит на страницу, где размещена только что обновленная запись.

```
128 router.delete("/:id", checkCampgroundOwnership, function(req, res) {
129     Campground.findByIdAndRemove(req.params.id, req.body.blog,
130     function(err, removedCampground) {
131         if (err) {
132             res.render("/campgrounds");
133         }
134         else {
135             res.redirect("/campgrounds");
136         }
137     });
138 });
```

Рисунок 3.4.11 – Удаление записи в БД

На рисунке мы создали контроллер, который даёт возможность удалять записи. Так как мы используем REST архитектуру, то для удаления нам

необходимо использовать HTTP метод DELETE. Здесь мы использовали метод findByIdAndRemove. Работа этого метода очень проста – находит в базе по id, и удаляет запись.

Создаем файл show.ejs в директории views. Это будет страницей просмотра туристического места.

```
$ touch views/show.ejs
```

Рисунок 3.4.12 – Создание страницы отображения всех мест

Далее в верхней части кода подключаем наш хэдер файл командой <% include /partials/header %>. Далее мы можем приступать к представлению нашей страницы.

```
<div class="col-md-9" data-aos="fade-left"
     data-aos-duration="2000">
    <div class="thumbnail">
        <img class="img-responsive" src="<%= campgrounds.image %>">
        <div class="caption-full">
            <h4 class="pull-right">$<%= campgrounds.cost %></h4>
            <h1><a><%= campgrounds.name %></a></h1>
            <p> <%= campgrounds.description %> </p>
            <p>
            <em> Submited by <%= campgrounds.author.username %> </em>
            </p>
            <% if(currentUser && campgrounds.author.id.equals(currentUser._id)){ %>
            <a class="btn btn-xs btn-warning" href="/campgrounds/<%=campgrounds._id%>/edit">Edit</a>
            <form id="deleteForm" action="/campgrounds/<%=campgrounds._id %>?_method=DELETE" method="POST">
            <button class="btn btn-xs btn-danger">Delete</button>
            </form>
            <% } %>
        </div>
    </div>
```

Рисунок 3.4.13 – Реализация отображения на стороне сервера

Здесь мы создали страницу используя колонки Bootstrap. Колонка нужна для разметки страницы, в частности, для создания адаптивных макетов. Сетка состоит из групп рядов и колонок, расположенных внутри одного или нескольких контейнеров.

Существует несколько типов колонок и способов их размещения внутри сетки в Bootstrap.

Колонки нужны для разделения области просмотра по горизонтали, при этом в одном ряду могут быть столбцы разной ширины. Их размер может изменяться в зависимости от некоторых факторов. Пространство между колонками называется «желоб» (gutter).

Рисунок 3.4.14 – Флексбокс

Обратите внимание на то, что по умолчанию у столбцов нет фона. Он добавлен дополнительно, чтобы четко обозначить границы.

В большинстве случаев, не требуется использование всех, их можно объединять по мере надобности. Представьте, что вся область просмотра разделена на 12 равных частей – единиц ширины. В одной колонке может быть от 1 до 12 таких единиц.

Классическая Bootstrap-сетка состоит из 12 колонок:

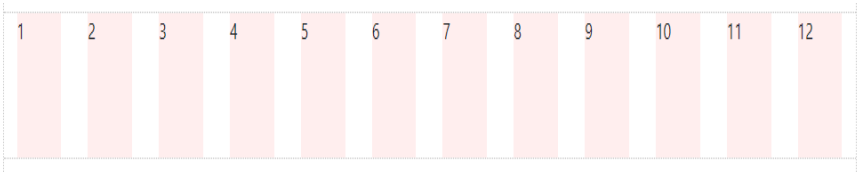


Рисунок 3.4.15 – Сетка из 12 частей

```
<div class="col-md-9"
```

Я в своём проекте использовал значение col-md-9. Это значит, что информация будет занимать 9/12 часть страницы. Для удобства, мы добавим снизу кнопку удаления и редактирования.

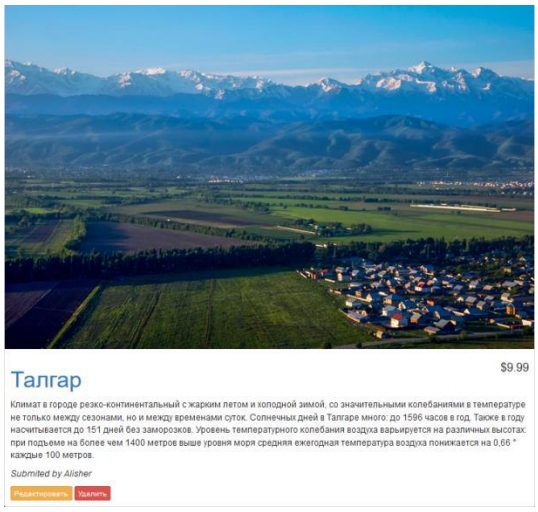


Рисунок 3.4.16 – Добавление кнопок удаления и редактирования

Теперь, когда у нас есть кнопки удаления и редактирования перед нами встаёт одна проблема. Любой пользователь при желании может удалить запись. Это является очень большой проблемой безопасности. И чтобы решить это нам нужно реализовать middleware метод проверки.

Функции промежуточной обработки (middleware) - 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.

Функции промежуточной обработки могут выполнять следующие задачи:

- Выполнение любого кода;
- Внесение изменений в объекты запросов и ответов;
- Завершение цикла “запрос-ответ”;
- Вызов следующей функции промежуточной обработки из стека.

Если текущая функция промежуточной обработки не завершает цикл “запрос-ответ”, она должна вызвать next() для передачи управления следующей функции промежуточной обработки. В противном случае запрос зависнет.

```
154 function checkCampgroundOwnership(req, res, next) {
155     if (req.isAuthenticated()) {
156         Campground.findById(req.params.id, function(err, foundCampground) {
157             if (err) {
158                 req.flash("error", "Campground not found");
159                 res.redirect("back");
160             }
161             else {
162                 if (foundCampground.author.id.equals(req.user._id)) {
163                     next();
164                 }
165                 else {
166                     req.flash("error", "You don't have permission to do that");
167                     res.redirect("back");
168                 }
169             }
170         });
171     }
172     else {
173         req.flash("error",  "you need to be logged in to do that");
174         res.redirect("/login");
175     }
176 }
```

Рисунок 3.4.17 – Middleware функция

Сперва наша функция промежуточной обработки проверяет авторизован ли пользователь, то есть, зашёл ли он на свой аккаунт или сидит на сайте как гость. Если проверка прошла успешно, то, чтобы иметь возможность изменять

эти записи или удалять, нужно пройти ещё одно условие. Как мы знаем, у каждого пользователя есть свой уникальный id в базе данных. И условие «if (foundCampground.author.id.equals(req.user._id))» проверяет совпадают ли id пользователя с тем id, который был у пользователя, создавшего эту запись. В случае несовпадения сервер выдаст ошибку.

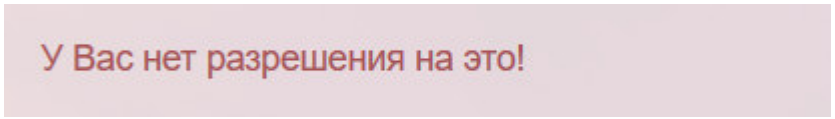


Рисунок 3.4.18 – Флэш сообщение

3.3 Создание авторизации

```
var express               =   require("express"),
    app                   =   express(),
    mongoose              =   require("mongoose"),
    passport              =   require("passport"),
    User                  =   require("./models/user"),
    bodyParser            =   require("body-parser"),
    LocalStrategy         =   require("passport-local"),
    passportLocalMongoose =   require("passport-local-
                                        mongoose");
```

Рисунок 3.3 – Объявление всех нужных библиотек.

Подключаемся к базе данных используя метод connect. Прописываем порт по умолчанию для mongoDB.

```
mongoose.connect("mongodb://localhost:27017/app",
{useNewUrlParser: true})
```

Рисунок 3.3.1 – Подключение к БД

Стандартный порт на котором работает mongoDB – это порт 27017.

```
app.use(require("express-session")({
    secret: "AUES is one of the best universities in KZ",
    resave: false,
    saveUninitialized: false
}));
```

Рисунок 3.3.1 – Сессия приложения

Здесь мы сказали нашему приложению использовать сессию. То есть каждый раз, когда пользователь будет входить в свой аккаунт, будет

начинаться новая сессия. Поле secret используется для энкодирования и декодирования сессии. Оно может быть любым.

```
passport.use(new LocalStrategy(User.authenticate()));
app.use(passport.initialize());
app.use(passport.session());
```

Рисунок 3.3.2 – Использование функций для сессии

Инициализация, необходима всегда, если в приложении будет password.

```
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());
```

Рисунок 3.3.3 – Сериализация и десериализация

Эти два метода ответственны за чтение и получение данных из сессии. Второй метод в самом конце полностью шифрует сессию(для безопасности).

3.3.1 Создание комментариев

Создаём контроллер, который будет выводить страницу, где будет выводиться форма.

```
router.get("/campgrounds/:id/comments/new", isLoggedIn ,function(req, res) {
   Campground.findById(req.params.id, function(err, foundCampground){
       if(err){
           console.log(err);
       }
       else{
           res.render("comments/new", {campground: foundCampground});
       }
   })
});
```

Рисунок 3.3.1 – Контроллер комментариев

Создаём файл new.ejs

```
$ touch views/newComment.ejs
```

Рисунок 3.3.2 – Создание файла комментариев

В этом файле создаём форму, которая в качестве действия будет отправлять POST метод в базу данных. В этой форме будет только одно поле, где и будет храниться текст комментария.

```
<form action="/campgrounds/<%=campground._id%>/comments" method="POST" style="width: 30%;
    <div class="form-group">
        <input class="form-control"  type="text" placeholder="text" name="comment[text]">
        <!--<textarea class="form-control" name="comment[text]"></textarea>-->
    </div>
    <div class="form-group">
        <button class="btn btn-lg btn-primary btn-block">Submit</button>
    </div>
    <a href="/campgrounds">Back to The Campgrounds page</a>
</form>
```

Рисунок 3.3.3 Создание формы для записи комментариев

Создаём контроллер с методом POST, которая будет брать значения из созданной ранее формы и отправлять всё это в базу данных. В базе данных мы уже прописали поле комментариев, когда создавали Модель туристического места в базе данных. Это значит, что каждая запись места будет хранить в базе данных свой массив комментариев.

```
router.post("/campgrounds/:id/comments",isLoggedIn,  function(req, res){
    Campground.findById(req.params.id, function(err, campground) {
        if(err){
            console.log(err);
            res.redirect("/campgrounds");
        }
        else{
            console.log(req.body.comment);
            Comment.create(req.body.comment, function(err, comment){
                if(err){
                    console.log(err);
                }
                else{
                    //Добавляем имя пользователя и айди в коммент
                    comment.author.id = req.user._id;
                    comment.author.username = req.user.username;
                     console.log("THE AUTHOR'S NAME IS " + req.user.username);
                    //save comment
                    comment.save();
                    campground.comments.push(comment);
                    console.log(comment);
                    campground.save();
                    req.flash("success", "Successfully added comment");
                    res.redirect("/campgrounds/" + campground._id);
                }
            });
```

Рисунок 3.3.3 – Пост запрос для отправки комментариев

Добавляем возможность редактировать комментарий. Для этого мы используем команду «Comment.findById», а в качестве параметра передаём id комментария.

```
router.get("/campgrounds/:id/comments/:comment_id/edit", checkCommentOwnership,
function(req, res) {
    Comment.findById(req.params.comment_id, function(err, comment) {
        if(err){
            console.log(err);
        }
        else{
            res.render("comments/edit",
            {comment: comment, campground_id: req.params.id});
        }
    })
});
```

Рисунок 3.3.4 – Редактирование комментариев

Далее создаём страницу editComment.ejs.

```
<form action="/campgrounds/<%=campground_id%>/comments/<%=comment._id%>?_method=PUT" method="POST" style="width: 30%;
    <div class="form-group">
        <input class="form-control"  type="text" placeholder="text" name="comment[text]" value="<%=comment.text%>">
        <!--<textarea class="form-control" name="comment[text]"></textarea>-->
    </div>
    <div class="form-group">
        <button class="btn btn-lg btn-primary btn-block">Submit</button>
    </div>
    <a href="/campgrounds">Back to The Campgrounds page</a>
</form>
```

Рисунок 3.3.5 – Создание страницы редактирования

Так как мы в контроллере вместе с id также передали и само тело комментария, то в форме мы можем взять текст данного комментария. Выглядеть это будет так:

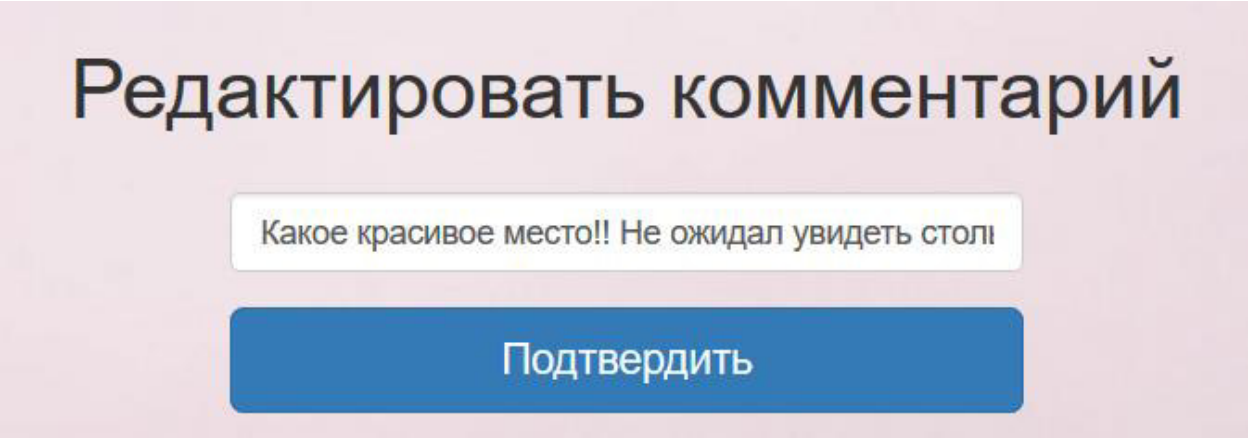


Рисунок 3.3.6 – Итоговый вариант страницы редактирования

Теперь осталось только реализовать возможность удаления комментария. Всё, что нам нужно сделать для удаления из базы данных – это с помощью метода findByIdAndRemove указать в параметрах id этого комментария. И контроллер передаст всю необходимую информацию в Модель, которая, в свою очередь, и удалит эту запись в базе данных.

```
router.delete("/campgrounds/:id/comments/:comment_id", checkCommentOwnership, function(req, res){
    Comment.findByIdAndRemove(req.params.comment_id, function(err, removedComment){
        if(err){
            console.log("back");
        } else {
            req.flash("success",  "Comment deleted");
            res.redirect("/campgrounds/" + req.params.id);
        }
    });
});
```

Наш комментарий в конце выглядит вот таким образом.

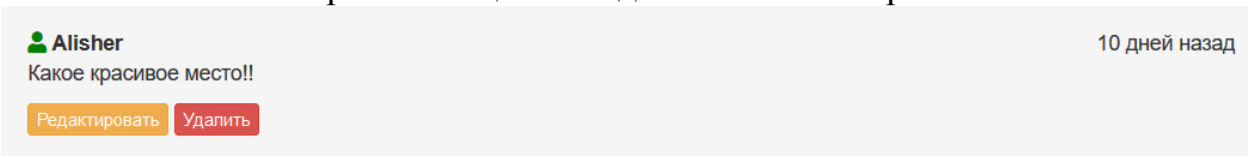


Рисунок 3.3.7 – Итоговый вид добавленного комментария

3.3.2 Добавление карты

Благодаря размещению карты, удастся создать интерфейс, который будет визуально насыщен и больше понравится пользователям. Такой интерфейс будет стимулировать ваших пользователей на взаимодействие.

```
var options = {
  provider: 'google',
  httpAdapter: 'https',
  apiKey: process.env.GEOCODER_API_KEY,
  formatter: null
};

var geocoder = NodeGeocoder(options);
```

Рисунок 3.3.2 – Добавление карты

Это создаст мобильную карту, которая растягивает ширину, но имеет фиксированную высоту. Теперь, когда у нас есть некоторые базовые стили, мы можем создать представление, которое отображает красивые места на карте:

Теперь мы создаём ключ по которому будем обращаться на сервер, чтобы использовать карту.

```
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCAhLuZI
X12gpK4pVwivVgKZPtFL-9XG74&callback=initMap">
</script>
```

Рисунок 3.3.22 – Подключение ключа

И вот, теперь в левом углу у нас есть карта с детальным расположением места.

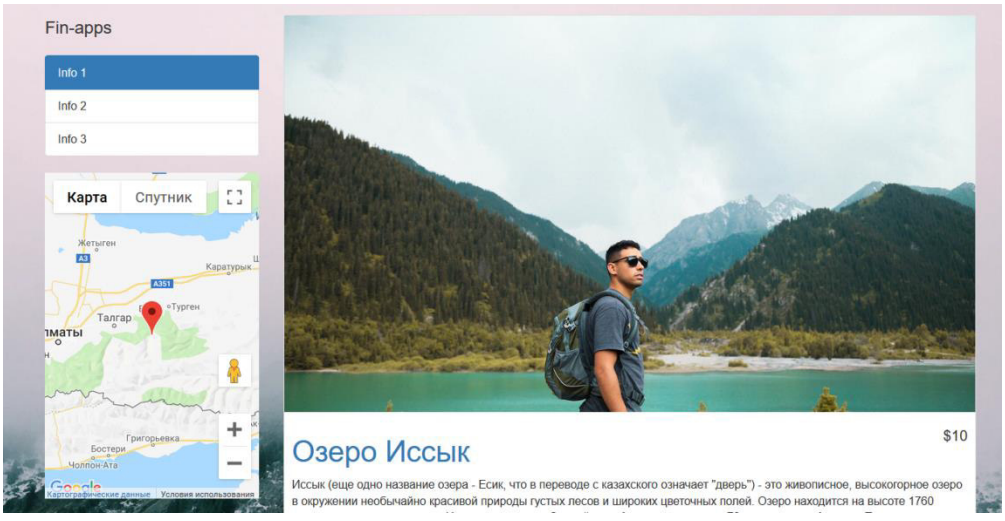


Рисунок 3.3.22 – Итоговый вывод информации

```
<script>
  function initMap() {
    var lat = <%= campgrounds.lat %>;
    var lng = <%= campgrounds.lng %>;
    var center = {lat: lat, lng: lng };
    var map = new google.maps.Map(document.getElementById('map'), {
        zoom: 8,
        center: center,
        scrollwheel: false
    });
    var contentString = `
      <strong><%= campgrounds.name %><br />
      <%= campgrounds.location %></strong>
      <p><%= campgrounds.description %></p>
    `
    var infowindow = new google.maps.InfoWindow({
      content: contentString
    });
    var marker = new google.maps.Marker({
        position: center,
        map: map
    });
    marker.addListener('click', function() {
      infowindow.open(map, marker);
    });
  }
</script>
```

Рисунок 3.3.21 – Созданный скрипт для выведения карты

4 Безопасность жизнедеятельности

Темой дипломного проекта было выбрано WEB-приложение, состоящее из двух частей, первая часть которого является этапом разработки технического задания, а ко второму этапу относится программная разработка. При разработке были соблюдены все правила техники безопасности, в связи с тем, что работа является сидячей, а именно разработка сайта, то основными пунктами были: освещение, вентиляция, а также работа с электрическими приборами и правила их эксплуатации.

Данная система основана на базе компании «Fin-apps». Описание рабочего места:

- Расположено на 3 этаже здания из 9;
- Размеры комнаты 10×5×3,2 (ширина-длина-высота);
- 3 компьютера и 2 принтера;
- Два окна размерностью (2×4);
- Лампа мощностью 60 Вт;
- Для борьбы с избыточной яркостью используются жалюзи;
- В помещении находится 3 человека, работая с 9:00 до 18:00.

Данные компьютеры не создают большого шумового давления в пределах нормативов. В связи с тем, что вентиляция проведена не самым лучшим образом, я решил, что в данном разделе Безопасности Жизнедеятельности я буду рассчитывать вентиляцию.

Перед предстоящим расчетом нужно получить исходные данные исходя из того, какое мы используем помещение, количество персонала, а также оборудование, которое участвует непосредственно в разработке программного продукта, а именно сайта.

Город: Алматы;
Параметры помещения (Ш х Д х В), м: 10×5×3,2;
Данные по оборудованию: кол-во 5 шт.;
Данные по ист. света: мощность N ос. уст., Вт/м^2 = 60;
Вид источника света: люминесцентные лампы;
Число сотрудников, из них: мужчины = 2, женщина = 1;
Окна: кол-во 2;
Площадь 1 окна, м^2 = 0.5;
Расположение: СВ;
Расчетное время суток, ч.: 12-13;
Температура в помещении, ^{0}С: летом 23, зимой 21;
Вид положения работы: сидячая работа.

В данной части дипломного проекты была рассмотрена тема вентиляции помещения. Вентиляция является основным параметром безопасности жизнедеятельности во время труда и является обязательным параметром для расчета. В данной части дипломного проекта был произведен расчет следующих пунктов:

– Расчет тепловых нагрузок внутри помещения;

– Расчет наружных тепловых нагрузок помещения;

– Расчет кол-во воздуха необходимого тому или иному помещению для подачи в данное помещение;

– По полученному расчету выбрать нужный кондиционер и показать все его технические характеристики в таблице;

– После выбора кондиционера показать расположение его блоков внешнего и внутреннего.

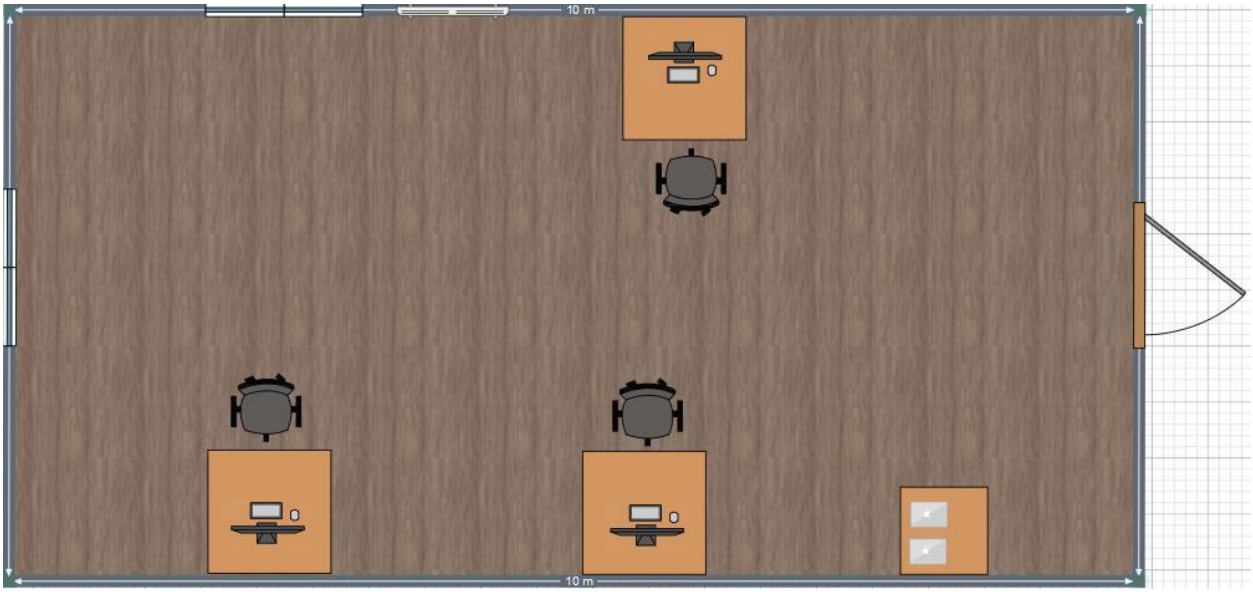


Рисунок 5.1 Схема помещения

Результаты проделанной работы:

4.1 Рассчитать тепловые нагрузки в помещении: внутренние и наружные.

Тепловые нагрузки непосредственно воздействуют на используемое нами помещение как внутренние тепловые нагрузки, так и внешние непосредственно климат, лучи солнца и общая температура погоды в городе в тот или иной сезон.

Наружные тепловые нагрузки.

Данные нагрузки представлены следующими составляющими:

– теплопоступления или теплопотери в результате разности температур снаружи и внутри здания через стены, потолки, полы, окна и двери;

– разность температур снаружи здания и внутри него летом является положительной, в результате чего имеет место приток тепла снаружи во внутрь помещения; и наоборот – зимой эта разность отрицательна и направление потока тепла меняется;

– теплопоступления от солнечного излучения через застекленные площади; данная нагрузка проявляется в форме ощущаемого тепла;

– теплопоступления от инфильтрации.

В зависимости от времени года и времени суток наружные тепловые нагрузки могут быть положительными. Теплопоступления и теплопотери в результате разности температур определяются по формуле 1.1:

$$Q_{огр} = V_{пом} \cdot X_o \cdot (t_{Нрасч} - t_{Врасч}), \quad Вт \tag{1.1}$$

, где:

$V_{пом.}$ – объем помещения, $м^3$:

$V_{пом}$= 10 х 5 х 3,2=160 $м^3$;

X_o– удельная тепловая характеристика, Вт/$м^3{}^0C$:

$$X_o = 0.42 \quad Вт / м^3{}^0C;$$

$t_{Нрасч}$ – наружная температура (параметр А). Для холодного периода – средняя температура самого холодного месяца в 13 часов, для теплого периода – средней температуре самого жаркого месяца в 13 часов.

$t_{Врасч}$ – внутренняя температура, выбирается с учетом комфортных условий или технологических требований, предъявляемых к производственным процессам.

Для теплого времени года:

$t_{Нрасч}$ = 29,4 ^{0}C

$t_{Врасч}$ = 26 ^{0}C

$Q_{огр.}$=160 х 0,42 х 3,4=228,48 Вт

Для холодного времени года

$t_{Нрасч}$ = -9 ^{0}C

$t_{Врасч}$ = 19 ^{0}C

$Q_{огр.}$=160 х 0,42 х |-28|=1881,6 Вт

Избыточная теплота солнечного излучения в зависимости от типа стекла почти до 90% поглощается средой помещения, остальная часть отражается. Максимальная тепловая нагрузка достигается при максимальном уровне излучения, которое имеет прямую и рассеянную составляющие. Интенсивность излучения зависит от ширины местности, времени года и времени суток.

Теплопоступление от солнечного излучения через остекление определяется по формуле 1.2:

$$Q_p = (q^I F_0^I + q^{II} F_0^{II}) \cdot \beta_{с.з.} \tag{1.2}$$

, где:

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

F^I_o, F^{II}_o – площади светового проема, облучаемые и не облучаемые прямой солнечной радиацией, м^3;

βс. з. – коэффициент теплопропускания:

βс. з. = 0.15

При отсутствии наружных затеняющих козырьков, ребер и т. д. для периода облучения остекления солнцем, когда его лучи проникают через окно в помещение $F^I_o=F_o$; $F^{II}_o=0$, (1.3):

$$Q_p = q^I F_0 \cdot \beta_{c.з.} = (q_{вп} + q_{вр}) \cdot K_1^c \cdot K_2 \cdot \beta_{c.з.} \cdot n \cdot S_o, \ Вт \quad (1.3),$$

, где:

$Q_{вп}$;$q_{вр}$ − тепловые потоки от прямой рассеянной радиации, Вт/м^2. Для широты в 440 СШ до полудня в 11-12 ч. при расположении З:

$Q_{вп}$ = 73 Вт/м^2 ;

$q_{вр}$ = 77 Вт/м^2 ;

$F_o = nS_o$ = 2x0.5=1 м^2 – площадь светового проема (n – число окон; S_o – площадь 1 окна);

K_1 – коэффициент затемнения остекления переплетами (K^C_1 – для облученных проемов):

K^C_1 = 0.72;

K_2 – коэффициент загрязнения остекления:

K_2 = 0.9.

Тогда:

Q_p= (73+77) x 0,72 x 0,9 x 0,15 x 1=14,58 Вт.

Для широты в 440 СШ до полудня в 11-12 ч. при расположении В:

$Q_{вп}$ = 214 Вт/м^2; $q_{вр}$ = 79 Вт/м^2 ;

$F_o = nS_o$ = 3·0.5=1.5 м^2– площадь светового проема (n – число окон; S_o – площадь 1 окна);

Тогда:

Q_p= (214+79)*0,72*0,9*0,15*1=28,47 Вт.

Тогда общее теплопоступление солнечного излучения с обеих окон равно:

Q_p=14,58 +28,47 =43,05 Вт.

4.1.2 Внутренние тепловые нагрузки.

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

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

В производственных и технологических помещениях различного назначения дополнительными источниками тепловыделений могут быть: нагретое производственное оборудование, горячие материалы, в том числе жидкости и различного рода полуфабрикаты, продукты сгорания и химических реакций.

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

Летом при 24^0С один мужчина выделяет явного тепла 61 Вт, а общего – 102 Вт. Женщина выделяет 85% от нормы тепловыделений взрослого мужчины

Тогда выделение явного тепла в помещении составит:

$$Q_л^я = 61 \text{ x } 2 + 61 \text{ x } 0{,}85 = 173{,}85 \text{ Вт.}$$

А выделение общего тепла:

$$Q_л^о = 102x2+102x0{,}85 = 290{,}7 \text{ Вт.}$$

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

$$Q_л^я = 82 \text{ x } 2 + 82 \text{ x } 0{,}85 = 233{,}7 \text{ Вт.}$$

А выделение общего тепла:

$$Q_л^о = 103 \text{ x } 2 + 103 \text{ x } 0{,}85 = 293{,}55 \text{ Вт.}$$

Теплопоступление от осветительных приборов, оргтехники и оборудования рассчитывается следующим образом. Теплопоступление от ламп определяется по формуле(1.4):

$$Q_{осв} = \eta \cdot N_{осв} \cdot F_{пол}, \quad Вт \tag{1.4}$$

, где:

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

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

$F_{пол}$ – площадь пола:

$F_{пол}$=10x5=50

Тогда:

$$Q_{осв} = 0{,}5 \text{ x } 60 \text{ x } 50 = 1500 \text{ Вт}$$

Тепло, выделяемое Персональным компьютером, определяется по формуле:

$$Q_{об} = N_{уст} \cdot K \quad (1.5)$$

$Q_{об}$=1,8*10^3*3*0,95= 5130 Вт.

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

$Q_{орг}$ = 1,8*10^3*3*0,3= 1539 Вт.

4.2 Рассчитать необходимое количество воздуха для вентиляции

На основании выполненных расчетов составим баланс теплопоступлений в помещении:

Лето: $Q_{изб}$=43,05 +173,85 +1500+5130 +1539 +228,48 = 8614,38 Вт

Зима:$Q_{изб}$=43,05 +233,7 +1500+5130 +1539 + 1881,6 = 10327,35 Вт

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

$$Q_н = \frac{Q_{изб.лето} \times 860}{V_{пом}} \quad (1.6)$$

$$Q_н = \frac{10{,}327 * 860}{160} = 55{,}50 \text{ ккал/м3}$$

При $Q_н$ >20ккал/м3 , Δt = 8 °С.

Определение количества воздуха, необходимое для поступления в помещение:

$$L = \frac{Q_{изб} \times 860}{C \times \Delta t \times \gamma} \quad (1.7)$$

$$L = \frac{10{,}327 * 860}{0{,}24 \cdot 8 \cdot 1{,}206 \cdot 10^4} = 357{,}02 \frac{\text{м3}}{\text{час}}$$

, где:

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

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

Определение кратности воздухообмена:

$$N = \frac{357.02}{160} = 2.23 \text{ час}^{-1}$$

По найденному значению количества воздуха подобрать соответствующую модель кондиционера.

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

Привести основные характеристики выбранного кондиционера.

Таблица 4.4 Основные технические характеристики настенного кондиционера серии Elenberg CSH-12OB

Эл. питаниеВ /Гн	Произв. по холоду, кВт	Потр. эл мощн, кВт	Потребл ток, А	Произв. по теплу, кВт	Размер (внешн. блок) мм	Расход воздуха, м3 /ч	Размер (внутр. блок) мм
380/3/50	10,65	5,376	9,2	12,24	L 1167 H 900 B 340	1000	L 945 H 660 B 205

Привести схему расположения кондиционера в помещении и схему подачи воздуха.

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

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

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

Так как количества воздуха, необходимое для поступления в помещение равно 357,02 м3/час, то будет использованодин кондиционер ElenbergCSH-12OB, который выдает необходимый нам расход воздуха.

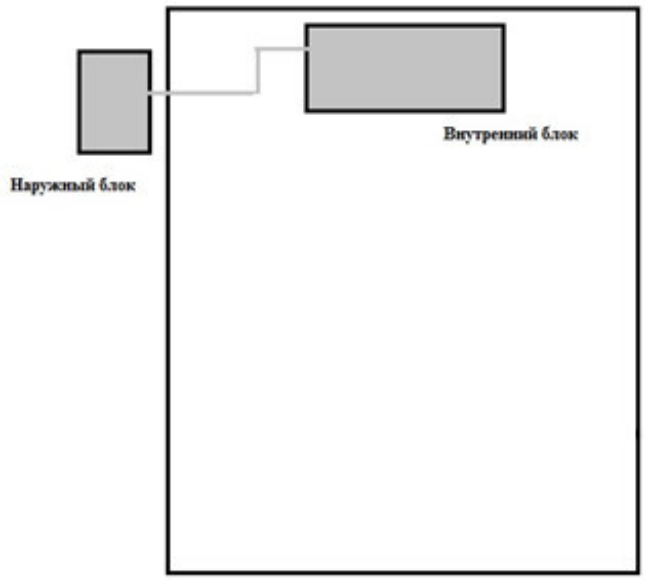


Рисунок 4.3 – Схема расположения кондиционеров в производственном помещении

Вывод

В этой части дипломного проекта мы расчитали возмонжные тепловые нагрузки как и внутренние так и внешние для выбранного нами помещения. Расчитали необходимое количество подаваемого воздуха, после того как узнали количество воздуха выбрали подходящий кондиционер по его характеристикам и указали место расположения его блоков внешнего вентиляционного и внутреннего в помещении.

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, а так же все пункты с овещением. Также установлено, что тепловые нагрузки зависят от размерности помещения и от количества людей находящимся в нем, а так же от таких внешних тепловых нагрузок как солнечные лучи и количество окон в помещении через которые они проходят. Большое влияние оказали погодные условия в городе проведения разработки.

5 Технико-экономическое обоснование

5.1 Описание работы и обоснование необходимости

Темой дипломного проекта является «разработка информационных систем для Fin-apps»

В данной дипломной работе описан проект по разработке сайта для тех, с детальной информацией для всех тех, кто хочет открыть для себя все красивые и интересные места Казахстана. Благодаря данному приложению можно найти нужную информацию в области туризма Казахстана.

Целью данного раздела является расчет затрат на создание программного продукта и расчет себестоимости прикладной программы.

Чтобы определить себестоимость программы, необходимо также принять:

- трудоемкость разработки программного продукта;
- материальные затраты;
- затраты на оплату труда;
- социальный налог;
- амортизация основных фондов.

6.2 Расчет трудоемкости разработки программного продукта

Таблица 6.1 – Распределение работ по этапам и видам и оценка их трудоемкости

Этап разработки	Вид работы	Трудоемкость разработки, чел.×ч.
1	Составление задачи	20
2	Создание алгоритмов и блок схемы	10
3	Создание клиентской части программного проекта	50
4	Создание серверной части проекта	100
5	Тестирование	20
6	Документация	40
ИТОГО трудоемкость выполнения дипломной работы		240

Чтобы определить сколько это будет в днях, нам нужно разделить итоговое число на 8(рабочий день): 240 / 8 = 30

6.3 Расчет затрат на разработку программного продукта

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

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

Таблица 6.2 – Затраты на материальные ресурсы

Наименование материального ресурса	Единица измерения	Количество	Цена за единицу, тг	Сумма, тг
Ноутбук ASUS ROG V502	шт.	1	150000	150000
ПО Fin-apps	шт.	1	70000	70000
Картридж	шт.	1	1500	1500
Бумага А4	шт.	1	2000	2000
ИТОГО				223500

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

$$З_M = \sum_{i=1}^{n} P_i \times Ц_i , \qquad (6.1)$$

,где:

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

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

i – вид материального ресурса;

n – количество видов материальных ресурсов.

$$З_{ноутбук} = 1 \times 150000 = 150000 \text{ тг}$$

$$З_{ПО} = 1 \times 70000 = 70000 \text{ тг}$$

$$З_{бумага} = 1 \times 2000 = 2000 \text{ тг}$$

$$З_{картридж} = 1 \times 1500 = 70000 \text{ тг}$$

$$З_{общие} = 150000 + 70000 + 2000 + 1500 = 223500 \text{тг}$$

Если для разработки ПП используется электрооборудование, то необходимо рассчитать затраты на электроэнергию по форме, приведенной в таблице 6.3.

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

$$З_Э = \sum_{i=1}^{n} M_i \times K_i \times T_i \times Ц\,, \qquad (6.2)$$

, где:

M_i– паспортная мощность i-го электрооборудования, кВт;

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

T_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 2019 году составляет 18,32 тенге за 1 кВт/ч с учетом НДС (согласно данным представленным на официальном сайте ТОО «АлматыЭнергоСбыт»).

$З_э$=0,9·0,9·240·18,32=3561,40 тг

$З_э$=0,3·0,7·240·18,32= 923.32 тг

Таблица 6.3 – 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	240	18,32	3561,40
Освещение	0,3	0,7	240	18,32	923.32
ИТОГО					4484.73

6.4 Расчет расходов на оплату труда

В разработке программного продукта заняты 2 сотрудника: инженер-разработчик и программист. Средняя заработная плата инженер-разработчика в 2019 году составляет 200 000 тг, а программиста 180 000 тг (для города Алматы).

Рабочие часы сотрудника за месяц определяются по формуле:

$$Ч_м = N_м \cdot Ч_{рд}, \quad (6.4)$$

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

$$Ч_м = 21 \times 8 = 168 \text{ ч.}$$

Часовая ставка работника может быть рассчитана по формуле:

$$ЧС_i = \frac{ЗП_i}{ФРВ_i} \quad (6.5)$$

Инженер-разработчик:

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

Программист:

$$ЧС_i = \frac{180000}{168} = 892,86 \ тг, \quad (6.6)$$

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 6.1.

Трудоемкость разработки ПП инженер-разработчика равна 450чел.×ч(составление задачи, разработка алгоритмов и блок схемы, реализация клиентской части проекта, тестирование, документация).

$$T_2 = 20+10+50+20+40 = 140 \text{ чел.×ч.}$$

Трудоемкость разработки ПП программиста равна 420чел.×ч.(разработка блок-схемы алгоритма, разработка администраторской части проекта, тестирование программы).

$$T_2 = 10 + 100 + 20 = 130 \text{ чел.×ч.}$$

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

$$З_{ТР} = \sum_{i=1}^{n} ЧС_i \times T_i \,, \qquad (6.7)$$

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

T_i– трудоемкость разработки ПП, чел.×ч;

i – категория работника;

n – количество работников, занятых разработкой ПП.

Инженер-разработчик:

$$З_{тр} = 1190{,}48 \times 140 = 166667.19 \text{ тг.}$$

Программист:

$$З_{тр} = 892{,}86 \times 130 = 116071.78 \text{ тг.}$$

Общая сумма:

$$З_{тр} = 166667.19 + 116071.78 = 282739{,}97 \text{ тг.}$$

Таблица 6.4 – Затраты на оплату труда

Квалификация	Трудоемкость разработки ПП, чел.×ч	Часовая ставка, тг/ч	Сумма, тг
Инженер-разработчик	140	1190,48	166667.19
Программист	130	892,86	116071.78
ИТОГО			282739,97

Дополнительная заработная плата:

$$З_{доп} = З_{тр} \times 10\%$$

$$З_{доп} = 282739{,}97 \times 0{,}1 = 28273.997 \text{ тг.}$$

Фонд заработной платы:

$$Ф_{зп} = З_{тр.о} + З_{доп} \qquad (6.8)$$

$$Ф_{зп} = 282739.97 + 28273.99 = 311013.95 \text{ тг.}$$

Расчет социального налога:

$$Н_с = (Ф_{зп} - ОПВ) \times 11\%, \qquad (6.9)$$

где ОПВ – обязательные пенсионные взносы–10% от $Ф_{зп}$.

$$Н_с = (311013.95 - (311013.95 \times 0{,}1)) \times 0{,}11 = 30790.38 \text{ тг.}$$

Расчет амортизационных основных фондов

Общая сумма амортизационных отчислений определяется по формуле:

$$З_{АМ} = \sum_{i=1}^{n} \frac{Ф_i \times Н_{Ai} \times Т_{НИРi}}{100 \times Т_{Эфi}}, \quad (6.10)$$

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 2019 год принимается $Т_{Эфi}$=1968);

i – вид ОФ;

n – количество ОФ.

Расчет годовой нормы амортизации ОФ:

$$Н_{Ai} = \frac{100}{4} = 25$$

$$Н_{Ai} = \frac{100}{Т_{Ni}}, \quad (6.11)$$

где $Т_{Ni}$– возможный срок использования i-го ОФ, год;

Для определения времени работы ПО для разработки ПП используются данные из таблицы 6.1.

Время работы ПО Fin-apps для разработки ПП составляет 170 часов (реализация клиентской части проекта, реализация администраторской части проекта, тестирование программы).

$$Т_i = 100 + 50 + 20 = 170 \text{ ч.}$$

Оборудование:

$$З_{АМ} = \frac{150000*25*240}{100*1968} = 4573.17$$

Программное обеспечение Fin-apps:

$$З_{АМ} = \frac{70000*25*170}{100*1968} = 1511.68$$

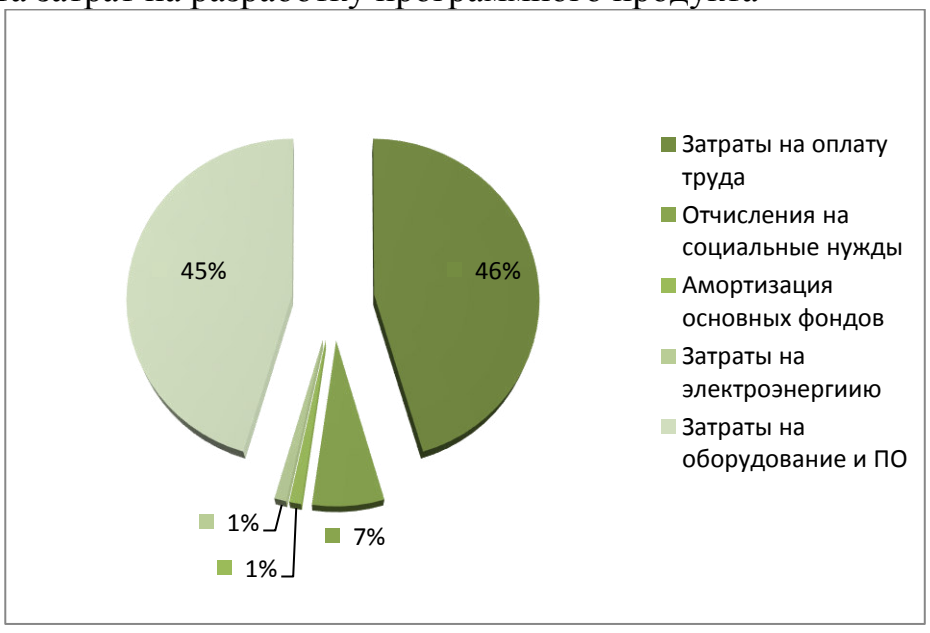
Таблица 6.5 – Амортизация основных фондов (ОФ)

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 Toshiba SATELLITE U840	70000	25	1968	240	4573.17
ПО GrantInfo	65000	25	1968	170	1511.68
ИТОГО					6084.85

Таблица 6.6 – Смета затрат на разработку ПП

Статьи затрат	Сумма, тг	%
1. Затраты на материальные ресурсы	223500,00	40
2. Затраты на электроэнергию	4484.73	1
3. Затраты на оплату труда	311013,95	52
4. Отчисления на социальные нужды	30790.38	6
5. Амортизация основных фондов	6084.85	1
ИТОГО	572373.92	100

Смета затрат на разработку программного продукта



Рисуное 6.3 Смета на разработку ПО

Определение возможной (договорной) цены программного продукта

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

$$Ц_Д = З_{НИР} \cdot \left(1 + \frac{Р}{100}\right), \quad (6.11)$$

где $З_{НИР}$– затраты на разработку ПП (из таблицы 6.6), тг;
Р– средний уровень рентабельности ПП – 25%.

$Ц_Д$ = 572373,92· (1 + 0,25) =572373,92+ 143093.47=715467.90 тг

Далее определяется цена реализации с учетом налога на добавленную стоимость (НДС), ставка НДС устанавливается законодательно Налоговым Кодексом РК. На 2019 год ставка НДС установлена в размере 12%.

Цена реализации с учетом НДС рассчитывается по формуле:

$$Ц_Р = Ц_Д \cdot (1 + НДС)$$

$Ц_Р$ = 715467.90 · (1 + 0,12) = 715467.90 + 85856.08= 801323.98 тг

6.6 Вывод по технически-экономической части

Таким образом, цена реализации с учетом НДС равна 801323.98 тг, себестоимость – 572373,92 тг и прибыль – 143093.47 тг.

Основную часть затрат составляют затраты на оплату труда(46%).

Заключение

В ходе выполнения дипломной проекта был разработан программный продукт с детальной информацией для всех тех, кто хочет открыть для себя все красивые и интересные места Казахстана.

Для успешного выполнения проекта были выполнены следующие задачи:

– проведено изучение предметной области на примере компании «Fin-apps»;

– определены основные требования к программному продукту;

– проведен анализ существующих технологий для решения поставленных задач и составлен собственных стек технологий для последующей разработки;

– при помощи таких технологий, как HTML, CSS, JavaScript и Node.jsбыла проведена разработка пользовательского интерфейса;

– спроектирована база данных, состоящая из 3 таблиц, предназначенная для хранения данных пользователей;

– добавлена функциональность на стороне сервера: регистрация, аутентификация, комментарии;

– произведено экономическое обоснование целесообразности разрабатываемого продукта, в ходе которого сделано следующее заключение: цена реализации окупает все затраты на разработку программного продукта. Прибыль от реализации программного продукта приблизительно равна 143 тысячам тенге.

– также были исследованы условия труда в предлагаемом помещении и предложены мероприятия по улучшению освещения в помещении в пасмурную погоду или темное время суток.

Таким образом, была проведена разработка приложения, которое полноценно выполняет все возложенные на него функции и предоставляет своим пользователям использование всего разработанного функционала.

Список литературы

1 Э. Фримен, Э. Фримен. Изучаем HTML, XHTML и CSS = Head First HTML with CSS & XHTML/ URL: https://ieeexplore.ieee.org/document/9785498071138 (датаобращения: 02.02.2019)

2 Кевин Кончес. HTML 5.3 Editors Draft / URL: https://w3c.github.io/html/ (датаобращения: 02.02.2019)

3 Activating Browser Modes with Doctype / URL: http://hsivonen.iki.fi/doctype/ (датаобращения: 05.02.2019)

4 The Unicode Standard: A Technical Introduction / URL: https://en.wikipedia.org/wiki/Parking_meter(дата обращения: 05.02.2019)

5 "Naming Files, Paths, and Namespaces" / URL: https://msdn.microsoft.com/en-us/library/windows/desktop/2147217396 (дата обращения: 05.02.2019)

6 Стивен Шафер. HTML, XHTML и CSS. Библия пользователя, 5-е издание = HTML, XHTML, and CSS Bible, 5th Edition / URL: http://www.dialektika.com/books/978-5-8459-1676-1.html (дата обращения: 08.02.2019)

7 ТОО «Hi-TecSecuritySystems», Автоматизированные платные парковки / URL: http://www.htss.kz/ (дата обращения: 08.02.2019)

8 Дэвид Сойер Макфарланд. Новая большая книга CSS = CSS: The Missing Manual / URL: https://orkhanalyshov.com/media/HTMLCSS/DavidSawyerMcFarlandCSStheMissingManual.pdf (дата обращения: 08.02.2019)

9 Web-based Mobile Apps of the Future Using HTML 5, CSS and JavaScript/ URL: http://www.htmlgoodies.com/beyond/article.php/3893911/Web-based-Mobile-Apps-of-the-Future-Using-HTML-5-CSS-and-JavaScript.htm (датаобращения: 08.02.2019)

10 "Selectors Level 3" / URL: http://www.w3.org/TR/selectors/ (дата обращения: 08.02.2019)

11 Press release announcing JavaScript/ URL: https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html (дата обращения: 08.02.2019)

12 Ohloh. – Анализ исходного кода/ URL: http://www.ohloh.net/p/eyeos/analyses/latest (дата обращения: 08.02.2019)

13

14 Бен Лин. «Волшебство Git» /– 2016. / URL: http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/ru/ch01.html (дата обращения: 08.02.2019)

15 «SQL или NoSQL – вот в чём вопрос» / URL: https://m.habr.com/ru/company/ruvds/blog/324936/ (дата обращения: 08.02.2019)

16 Скринкаст Node.JS от А до Я / URL: http://learn.javascript.ru/nodejs-screencast (дата обращения: 12.02.2019)

17 "Large volume data analysis on the Typesafe Reactive Platform" / URL: http://www.slideshare.net/MartinZapletal/zapletal-martinlargevolumedataanalytics (дата обращения: 12.02.2019)

18 Оффициальный сайт MongoDB/ URL: https://www.mongodb.com/ (дата обращения: 12.02.2019)

19 Development of a neural interface for the control/ URL: URL: https://7c4745ab-a-cdf32725-s-sites.googlegroups.com/a/neurostat.mit.edu/lciti/ (дата обращения: 12.02.2019)

20 MongoDB (статья Википедии) / https://en.wikipedia.org/wiki/MongoDB (дата обращения: 12.02.2019)

Приложение А
(обязательное)

Техническое задание

– Разработка сайта;

– Разработка прототипа. Разработать структуру сайта на основании предварительной структуры сайта с интерактивными элементами и формами, с которыми можно взаимодействовать;

– Верстка и наполнение сайта информационными и графическими материалами. Разработать сайт на основе адаптивной верстки, обеспечивающий корректную работу сайта на всех компьютерных и мобильных устройствах;

– Для просмотра сайта, можно использовать любой компьютер с выходом в интернет и установленным браузером InternetExplorer 9 и выше, или Firefox 33 и выше, или Opera 22 и выше, или Safari 8.0 и выше, или GoogleChrome 38 и выше;

– Включить поддержку JavaScript и cookies;

– В качестве базы данных для сайта использовать MongoDB;

– Заказчику должны быть предоставлены права админа;

– Запрещается использовать в дизайне сайта Flash-элементы. Необходимо использовать анимационные эффекты на основе технологии JS, JQ, HTML5 и CSS3.

Основные разделы сайта:

– Лэндинг страница
– Главная страница
– Слайдер с фотографиями
– В странице с местом разработать блок комментариев.

Приложение Б
(обязательное)

Листинг программы

```
require('dotenv').config();

var express             = require("express"),
    app                 = express(),
    bodyParser          = require("body-parser"),
    mongoose            = require("mongoose"),
    flash               = require("connect-flash"),
    passport            = require("passport"),
    methodOverride      = require("method-override"),
    User                = require("./models/user"),
    LocalStrategy       = require("passport-local"),
    passportLocalMongoose = require("passport-local-mongoose"),
    Campground          = require("./models/campground"),
    Comment             = require("./models/comment"),
    seedsDB             = require("./seeds.js");

var campgroundRoutes    = require("./routes/campgrounds"),
    commentRoutes       = require("./routes/comments"),
    indexRoutes         = require("./routes/index"),
    budgetRoute         = require("./routes/budget");


mongoose.connect("mongodb://localhost:27017/yelpcamp", {useNewUrlParser: true});
app.use(bodyParser.urlencoded({ extended: true }));
app.set("view engine", "ejs");
app.use(express.static(__dirname + "/public"));
app.use(methodOverride("_method"));
app.use(flash());
// seedsDB();  //////НЕ ТРОГАТЬ!!!!!

app.use(require("express-session")({
    secret: "Rusty is the best and cuttiest dog in the world",
    resave: false,
    saveUninitialized: false
}));

app.use(passport.initialize());

app.use(passport.session());

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("/campgrounds",campgroundRoutes);
app.use(commentRoutes);
app.use(indexRoutes);


app.listen(process.env.PORT, process.env.IP, function() {
   console.log("YelpCamp server has started");
})

   var mongoose = require("mongoose");
   var Campground = require("./models/campground");
   var Comment   = require("./models/comment");

   var data = [
      {
         name: "Cloud's Rest",
         image: "https://farm4.staticflickr.com/3795/10131087094_c1c0a1c859.jpg",
         description: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum"
      },
      {
         name: "Desert Mesa",


 image: "https://farm6.staticflickr.com/5487/11519019346_f66401b6c1.jpg",
         description: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum"
      },
      {
         name: "Canyon Floor",
         image: "https://farm1.staticflickr.com/189/493046463_841a18169e.jpg",
         description: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum"}
```

```
]

function seedDB(){
  //Remove all campgrounds
  Campground.remove({}, function(err){
      if(err){
         console.log(err);
      }
      console.log("removed campgrounds!");
      Comment.remove({}, function(err) {
         if(err){
            console.log(err);
         }
         console.log("removed comments!");
          //add a few campgrounds
         data.forEach(function(seed){
            Campground.create(seed, function(err, campground){
               if(err){
                  console.log(err)
               } else {
                  console.log("added a campground");
                  //create a comment
                  Comment.create(
                     {
                        author: "Homer",


                      text: "This place is great, but I wish there was internet"
                       }, function(err, comment){
                          if(err){
                             console.log(err);
                          } else {
                             campground.comments.push(comment);
                             campground.save();
                             console.log("Created new comment");
                          }
                       });
               }
            });
         });
      });
    });
    //add a few comments
}

module.exports = seedDB;
```

```
.env
/node_modules/node_modules.1
```

```
<% include partials/header %>
<div class="container">
<h1 style="text-align: center;">Регистрация</h1>

<form action="/register" method=POST style="width: 30%; margin: 25px auto;">
   <div class="form-group">
   <input class="form-control" type="text" placeholder="Имя пользователя" name="username">
   </div>
   <div class="form-group">
   <input class="form-control" type="password" placeholder="Пароль" name="password">
   </div>
   <div class="form-group">
   <button class="btn btn-lg btn-primary btn-block">Submit</button>
   </div>
</form>

<% include partials/footer %>

<% include partials/header %>
<div class="container">
<h1 style="text-align: center;">Вход в Аккаунт</h1>

<form action="/login" method=POST style="width: 30%; margin: 25px auto;">
   <div class="form-group">
      <input class="form-control" type="text" placeholder="Имя пользователя"
name="username">
   </div>
   <div class="form-group">
      <input class="form-control" type="password" placeholder="Пароль" name="password">
   </div>
   <div class="form-group">
      <button class="btn btn-lg btn-primary btn-block">Submit</button>
   </div>
</form>
</div>
<% include partials/footer %>

<!DOCTYPE html>
<html>
   <head>
      <title>YelpCamp</title>
      <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
      <link rel="stylesheet" href="/stylesheets/landing.css">
      <script src="https://cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.min.js"
type="text/javascript" async></script>
   </head>
   <body>
   <div class="container">
```

```
      <% if(error && error.length > 0){ %>
        <div class="alert alert-danger" role="alert">
          <%= error %>
        </div>
      <% } %>
      <% if(success && success.length > 0){ %>
        <div class="alert alert-success" role="alert">
          <%= success %>
        </div>

  <% } %>
   </div>

   <div id="landing-header">
                <h1>Добро пожаловать в Fin-apps!</h1>
                <a href="/campgrounds" class="btn btn-lg btn-success">Просмотреть все
места</a>
   </div>

   <ul class="slideshow">
     <li></li>
     <li></li>
     <li></li>
     <li></li>
     <li></li>
   </ul>

<% include partials/footer %>

<html lang="en">
   <head>
      <meta charset="UTF-8">
      <link href="https://fonts.googleapis.com/css?family=Open+Sans:100,300,400,600"
rel="stylesheet" type="text/css">
      <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css"
integrity="sha384-
50oBUHEmvpQ+1lW4y57PTFmhCaXp0ML5d60M1M7uH2+nqUivzIebhndOJK28anvf"
crossorigin="anonymous">
      <link type="text/css" rel="stylesheet" href="/stylesheets/style.css">
      <title>Budgety</title>
   </head>
   <body>

      <div class="top">
        <div class="budget">
          <div class="budget__title">
            Доступный в <span class="budget__title--month">%Month%</span>  бюджет:
          </div>
```

```
            <div class="budget__value">+ 2,345.64</div>

<div class="budget__income clearfix">
                <div class="budget__income--text">Имеющаяся сумма</div>
                <div class="right">
                    <div class="budget__income--value">+ 4,300.00</div>
                    <div class="budget__income--percentage"> </div>
                </div>
            </div>

            <div class="budget__expenses clearfix">
                <div class="budget__expenses--text">Расходы</div>
                <div class="right clearfix">
                    <div class="budget__expenses--value">- 1,954.36</div>
                    <div class="budget__expenses--percentage">45%</div>
                </div>
            </div>
        </div>
    </div>



    <div class="bottom">
        <div class="add">
            <div class="add__container">
                <select class="add__type">
                    <option value="inc" selected>+</option>
                    <option value="exp">-</option>
                </select>
                <input type="text" class="add__description" placeholder="Добавить описание">
                <input type="number" class="add__value" placeholder="Сумма">
                <button class="add__btn"><i class="fas fa-check"></i></button>
            </div>
        </div>

        <div class="container clearfix">
            <div class="income">
                <h2 class="icome__title">Доходы</h2>

                <div class="income__list">

                    <!--
                    <div class="item clearfix" id="income-0">
                        <div class="item__description">Salary</div>

 <div class="right clearfix">
                            <div class="item__value">+ 2,100.00</div>
                            <div class="item__delete">
                                <button class="item__delete--btn">
```

```
                        <i class="ion-ios-close-outline"></i>
                      </button>
                    </div>
                  </div>
                </div>

                <div class="item clearfix" id="income-1">
                  <div class="item__description">Sold car</div>
                  <div class="right clearfix">
                    <div class="item__value">+ 1,500.00</div>
                    <div class="item__delete">
                      <button class="item__delete--btn"><i class="ion-ios-close-
outline"></i></button>
                    </div>
                  </div>
                </div>
                -->

              </div>
            </div>



            <div class="expenses">
              <h2 class="expenses__title">Расходы</h2>

              <div class="expenses__list">

                <!--
                <div class="item clearfix" id="expense-0">
                  <div class="item__description">Apartment rent</div>
                  <div class="right clearfix">
                    <div class="item__value">- 900.00</div>
                    <div class="item__percentage">21%</div>
                    <div class="item__delete">
                      <button class="item__delete--btn"><i class="ion-ios-close-
outline"></i></button>
                    </div>
                  </div>
 </div>
                <div class="item clearfix" id="expense-1">
                  <div class="item__description">Grocery shopping</div>
                  <div class="right clearfix">
                    <div class="item__value">- 435.28</div>
                    <div class="item__percentage">10%</div>
                    <div class="item__delete">
                      <button class="item__delete--btn"><i class="ion-ios-close-
outline"></i></button>
                    </div>
```

```
                    </div>
                </div>
                -->

            </div>
        </div>
    </div>


</div>

<script>
    // BUDGET CONTROLLER
var budgetController = (function() {

    var Expense = function(id, description, value) {
        this.id = id;
        this.description = description;
        this.value = value;
        this.percentage = -1;
    };


    Expense.prototype.calcPercentage = function(totalIncome) {
        if (totalIncome > 0) {
            this.percentage = Math.round((this.value / totalIncome) * 100);
        } else {
            this.percentage = -1;
        }
    };



Expense.prototype.getPercentage = function() {
        return this.percentage;
    };


    var Income = function(id, description, value) {
        this.id = id;
        this.description = description;
        this.value = value;
    };

    var calculateTotal = function(type) {
        var sum = 0;
        data.allItems[type].forEach(function(cur) {
            sum += cur.value;
        });
        data.totals[type] = sum;
```

```
    };


    var data = {
        allItems: {
            exp: [],
            inc: []
        },
        totals: {
            exp: 0,
            inc: 0
        },
        budget: 0,
        percentage: -1
    };


    return {
        addItem: function(type, des, val) {
            var newItem, ID;

            //[1 2 3 4 5], next ID = 6
            //[1 2 4 6 8], next ID = 9
            // ID = last ID + 1

// Create new ID
            if (data.allItems[type].length > 0) {
                ID = data.allItems[type][data.allItems[type].length - 1].id + 1;
            } else {
                ID = 0;
            }

            // Create new item based on 'inc' or 'exp' type
            if (type === 'exp') {
                newItem = new Expense(ID, des, val);
            } else if (type === 'inc') {
                newItem = new Income(ID, des, val);
            }

            // Push it into our data structure
            data.allItems[type].push(newItem);

            // Return the new element
            return newItem;
        },


        deleteItem: function(type, id) {
            var ids, index;
```

```
        // id = 6
        //data.allItems[type][id];
        // ids = [1 2 4  8]
        //index = 3

        ids = data.allItems[type].map(function(current) {
            return current.id;
        });

        index = ids.indexOf(id);

        if (index !== -1) {
            data.allItems[type].splice(index, 1);
        }

    },



calculateBudget: function() {

        // calculate total income and expenses
        calculateTotal('exp');
        calculateTotal('inc');

        // Calculate the budget: income - expenses
        data.budget = data.totals.inc - data.totals.exp;

        // calculate the percentage of income that we spent
        if (data.totals.inc > 0) {
            data.percentage = Math.round((data.totals.exp / data.totals.inc) * 100);
        } else {
            data.percentage = -1;
        }

        // Expense = 100 and income 300, spent 33.333% = 100/300 = 0.3333 * 100
    },

    calculatePercentages: function() {
        /*
        a=20
        b=10
        c=40
        income = 100
        a=20/100=20%
        b=10/100=10%
        c=40/100=40%
        */
        data.allItems.exp.forEach(function(cur) {
```

```
            cur.calcPercentage(data.totals.inc);
        });
    },


    getPercentages: function() {
        var allPerc = data.allItems.exp.map(function(cur) {
            return cur.getPercentage();
        });
        return allPerc;

},


    getBudget: function() {
        return {
            budget: data.budget,
            totalInc: data.totals.inc,
            totalExp: data.totals.exp,
            percentage: data.percentage
        };
    },

    testing: function() {
        console.log(data);
    }
};

})();



// UI CONTROLLER
var UIController = (function() {
    var DOMstrings = {
        inputType: '.add__type',
        inputDescription: '.add__description',
        inputValue: '.add__value',
        inputBtn: '.add__btn',
        incomeContainer: '.income__list',
        expensesContainer: '.expenses__list',
        budgetLabel: '.budget__value',
        incomeLabel: '.budget__income--value',
        expensesLabel: '.budget__expenses--value',
        percentageLabel: '.budget__expenses--percentage',
        container: '.container',
        expensesPercLabel: '.item__percentage',
        dateLabel: '.budget__title--month'
```

```
    };


    var formatNumber = function(num, type) {

var numSplit, int, dec, type;
        /*
            + or - before number
            exactly 2 decimal points
            comma separating the thousands

            2310.4567 -> + 2,310.46
            2000 -> + 2,000.00
            */

        num = Math.abs(num);
        num = num.toFixed(2);

        numSplit = num.split('.');

        int = numSplit[0];
        if (int.length > 3) {
            int = int.substr(0, int.length - 3) + ',' + int.substr(int.length - 3, 3); //input 23510, output
23,510
        }

        dec = numSplit[1];

        return (type === 'exp' ? '-' : '+') + ' ' + int + '.' + dec;

    };


    var nodeListForEach = function(list, callback) {
        for (var i = 0; i < list.length; i++) {
            callback(list[i], i);
        }
    };


    return {
        getInput: function() {
            return {
                type: document.querySelector(DOMstrings.inputType).value, // Will be either inc or
exp
                description: document.querySelector(DOMstrings.inputDescription).value,


    value: parseFloat(document.querySelector(DOMstrings.inputValue).value)
            };
```

```
    },


    addListItem: function(obj, type) {
        var html, newHtml, element;
        // Create HTML string with placeholder text

        if (type === 'inc') {
            element = DOMstrings.incomeContainer;

            html = '<div class="item clearfix" id="inc-%id%"> <div
class="item__description">%description%</div><div class="right clearfix"><div
class="item__value">%value%</div><div class="item__delete"><button class="item__delete--
btn"><i class="ion-ios-close-outline"></i></button></div></div></div>';
        } else if (type === 'exp') {
            element = DOMstrings.expensesContainer;

            html = '<div class="item clearfix" id="exp-%id%"><div
class="item__description">%description%</div><div class="right clearfix"><div
class="item__value">%value%</div><div class="item__percentage">21%</div><div
class="item__delete"><button class="item__delete--btn"><i class="ion-ios-close-
outline"></i></button></div></div></div>';
        }

        // Replace the placeholder text with some actual data
        newHtml = html.replace('%id%', obj.id);
        newHtml = newHtml.replace('%description%', obj.description);
        newHtml = newHtml.replace('%value%', formatNumber(obj.value, type));

        // Insert the HTML into the DOM
        document.querySelector(element).insertAdjacentHTML('beforeend', newHtml);
    },


    deleteListItem: function(selectorID) {

        var el = document.getElementById(selectorID);
        el.parentNode.removeChild(el);


},
    clearFields: function() {
        var fields, fieldsArr;

        fields = document.querySelectorAll(DOMstrings.inputDescription + ', ' +
DOMstrings.inputValue);

        fieldsArr = Array.prototype.slice.call(fields);
```

```
            fieldsArr.forEach(function(current, index, array) {
                current.value = "";
            });

            fieldsArr[0].focus();
        },


        displayBudget: function(obj) {
            var type;
            obj.budget > 0 ? type = 'inc' : type = 'exp';

            document.querySelector(DOMstrings.budgetLabel).textContent =
formatNumber(obj.budget, type);
            document.querySelector(DOMstrings.incomeLabel).textContent =
formatNumber(obj.totalInc, 'inc');
            document.querySelector(DOMstrings.expensesLabel).textContent =
formatNumber(obj.totalExp, 'exp');

            if (obj.percentage > 0) {
                document.querySelector(DOMstrings.percentageLabel).textContent = obj.percentage +
'%';
            } else {
                document.querySelector(DOMstrings.percentageLabel).textContent = '---';
            }

        },


        displayPercentages: function(percentages) {


var fields = document.querySelectorAll(DOMstrings.expensesPercLabel);

            nodeListForEach(fields, function(current, index) {

                if (percentages[index] > 0) {
                    current.textContent = percentages[index] + '%';
                } else {
                    current.textContent = '---';
                }
            });

        },


        displayMonth: function() {
            var now, months, month, year;
            now = new Date();
```

```
            //var christmas = new Date(2016, 11, 25);

            months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September',
'October', 'November', 'December'];
            month = now.getMonth();

            year = now.getFullYear();
            document.querySelector(DOMstrings.dateLabel).textContent = months[month] + ' ' +
year;
        },


        changedType: function() {

            var fields = document.querySelectorAll(
                DOMstrings.inputType + ',' +
                DOMstrings.inputDescription + ',' +
                DOMstrings.inputValue);

            nodeListForEach(fields, function(cur) {
              cur.classList.toggle('red-focus');
            });

            document.querySelector(DOMstrings.inputBtn).classList.toggle('red');
    },


        getDOMstrings: function() {
            return DOMstrings;
        }
    };

})();



// GLOBAL APP CONTROLLER
var controller = (function(budgetCtrl, UICtrl) {

    var setupEventListeners = function() {
        var DOM = UICtrl.getDOMstrings();

        document.querySelector(DOM.inputBtn).addEventListener('click', ctrlAddItem);

        document.addEventListener('keypress', function(event) {
            if (event.keyCode === 13 || event.which === 13) {
                ctrlAddItem();
            }
        });
```

```
    document.querySelector(DOM.container).addEventListener('click', ctrlDeleteItem);

    document.querySelector(DOM.inputType).addEventListener('change', UICtrl.changedType);
  };


  var updateBudget = function() {

    // 1. Calculate the budget
    budgetCtrl.calculateBudget();

    // 2. Return the budget
    var budget = budgetCtrl.getBudget();


// 3. Display the budget on the UI
    UICtrl.displayBudget(budget);
  };


  var updatePercentages = function() {

    // 1. Calculate percentages
    budgetCtrl.calculatePercentages();

    // 2. Read percentages from the budget controller
    var percentages = budgetCtrl.getPercentages();

    // 3. Update the UI with the new percentages
    UICtrl.displayPercentages(percentages);
  };


  var ctrlAddItem = function() {
    var input, newItem;

    // 1. Get the field input data
    input = UICtrl.getInput();

    if (input.description !== "" && !isNaN(input.value) && input.value > 0) {
      // 2. Add the item to the budget controller
      newItem = budgetCtrl.addItem(input.type, input.description, input.value);

      // 3. Add the item to the UI
      UICtrl.addListItem(newItem, input.type);

      // 4. Clear the fields
      UICtrl.clearFields();
```