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 web-сервиса «Сеть гостеприимства»

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

Выполнил: Сагимбеков М.А. Группа: ВТ-15-2
Научный руководитель: доцент, к.т.н. Табултаев С.С.

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
«13» мая 2019 г.

по применению
вычислительной техники: ст. преп. _____ М.Н. Майкотов
«14» мая 2019 г.

Нормоконтролер: ст. преп. _____ А.А. Айтказина
«15» 05 2019 г.

Рецензент: к.т.н., асс. профессор _____ Н.А. Сейлова
«____» _____ 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 дипломного проекта

Студенту Сагимбеков Мирас Асылович

Тема проекта: Разработка web-сервиса «Сеть гостеприимства»

Утверждена приказом по университету № 33 от «01» марта 2019 г.

Срок сдачи законченного проекта «22» 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. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.
2. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
3. Chris Fidao. Implementing Laravel. Real-world implementation of testable and maintainable code. 2014. - 105c.
4. Терри Фельке-Моррис. Большая книга веб-дизайна. 2017.

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.	27.03.2019 – 8.05.2019	
Программное обеспечение	Майкотов М.Н.	04.04.2019 – 14.05.2019	
Нормоконтролер	Айтказина А.А.	02.04.19 – 15.05.19	

ГРАФИК
подготовки дипломной проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Аналитическая часть	05.11.2018 – 20.12.2018	
Проектная часть	07.01.2019 – 30.01.2019	
Экспериментальная часть	04.02.2019 – 13.04.2019	

Дата выдачи задания «25» октября 2018 г.

Заведующий кафедрой _____ Т.С. Картбаев

Научный руководитель проекта _____ С.С. Табултаев

Задание принял к
исполнению студент _____ М.А. Сагимбеков

Аңдатпа

Дипломдық жобаның тақырыбы «Қоңақжайлық желісі» web-сервисін әзірлеу».

Дипломдық жобаның мақсаты – елдегі туризмді дамытып Қазақстанды әлемдік деңгейде танымал ету

Алға қойған мақсатқа жету үшін келесідей технологиялар қолданылды: HTML, CSS, Python, Django;

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 web-сервиса «Сеть гостеприимства».

Цель дипломного проекта – развитие туризма в стране и популяризация Казахстана на мировой арене.

Для достижения поставленной цели использовались технологии: HTML, CSS, Python, Django.

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 web-service – “Hospitality Network”».

The purpose of the study: development of tourism in the country and popularization of Kazakhstan on the world stage.

Technologies used to achieve the goal: HTML, CSS, Python, Django.

The graduation project includes an introduction, five 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 systematic 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, 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

Природа нашей страны настолько уникальна, настолько красива, каждый край нашей страны, в силу разности климатических условий, уникален по-своему. Эта разность климатических условий, привела к тому что быт в разных частях нашей страны, исторически отличается друг от друга, а в древности отличий было еще больше.

И эта разность послужила основой для другой разности: разности археологических артефактов на территории Казахстана. Арии, саки, гунны, сарматы, древние тюрки, персы, согдийцы, кидани, монголы Чингизхана и др. Все эти народы в свое время проживали на территории нашей необъятной страны и все они оставили свои исторические памятники и археологические артефакты. И всё что они оставили после себя представляет собой не только культурную и историческую, но и туристическую ценность.

А со становлением Казахстана, как независимого государства, их ценность возросла еще больше, каждый регион получил своё уникальное развитие.

И для того, чтобы побывать в других городах, увидеть красоту природы нашей страны, люди пользуются услугами различных туристических агентств, гостиниц и т.д., в разы переплачивая за то чтобы просто насладиться достопримечательностями и уникальной природой нашей необъятной страны. Альтернативой всему этому и являются сети гостеприимства.

Сети гостеприимства - это сообщества людей с разных уголков земли, которые приглашают погостить к себе домой других членов этого сообщества. В основном, это люди, объединенные общими интересами (спортсмены, историки, каратисты и т.д.). Одним из первых, под лозунгом «Мир во всем мире», был проект обмена гостеприимством, который реализовала, открывшейся в 1949 году, международная некоммерческая служба Servas Open Doors. Эта организация стремилась развить межнациональноей и межкультурное взаимодействие, а также толерантные отношения между представителями различных национальностей, создав всемирную сеть людей, готовых открыть свои дома путешественникам.

В течение следующих десятилетий возникли сходные организации и множество подобных «групп по интересам», которые в том числе помогали с ночлегом байкерам, автостопщикам и т.д. Поиск жилья через «гостеприимные сети» имеет ряд преимуществ, а именно:

- возможность встретить интересных людей, единомышленников;
- возможность познакомиться «изнутри» с бытом местных жителей;
- сэкономить на жилье;
- пройти хорошую практику в изучении иностранных языков;

Существует множество сетей гостеприимства. Они отличаются друг от друга количеством членов, направленностью, устройством сайтов. Многие люди являются одновременно членами сразу нескольких сетей.

Наиболее крупные из них – сети общей направленности CouchSurfing (couchsurfing.org) и Hospitality Club (Клуб гостеприимства, hospitalityclub.org), специализированные WarmShowers (сеть ориентирована на велосипедистов-путешественников), Pasporta Servo (сеть обмена гостеприимством среди эсперантистов).

В нашей стране аналога подобных сервисов нет, вдохновившись идеей, я решил разработать первую казахстанскую сеть гостеприимства. Конечно, такой формат путешествий заставляет задуматься над вопросом, насколько он безопасен. Основными инструментами обеспечения безопасности являются:

- отзывы других пользователей;
- запись переговоров с администрацией сайта;
- указание участниками своих паспортных данных.

Разработанный мною сервис призван развить в стране туризм и популяризировать страну за границей.

В этом заключается актуальность выбранной мною темы.

1 Описание программного продукта

Web-сервис «Сеть гостеприимства» это лишь общее название проекта, итоговым названием должен быть придуманный бренд, окончательным названием будет «Клуб гостеприимства NomadTrip».

Чтобы вступить в клуб необходимо зарегистрироваться на сайте, и если возможно, как можно более полно и интересно заполнить анкету, которую увидят другие участники. После этого становится доступным поиск людей, у которых можно остановиться.

Если пользователь готов отправиться в путешествие, ему необходимо написать письмо человеку, который готов предоставить свое жиль, т.е. у которого можно остановиться. В письме сообщаются даты приезда/отъезда, цели путешествия, планы, увлечения и всё остальное, что может заинтересовать собеседника. Часто при поиске рассылают письма сразу нескольким участникам, так как в ответном письме может быть отказ (хозяин может быть занят в эти дни, находиться в другом городе, он может уже договориться с другими путешественниками о том, что они приедут к нему в указанные дни, да и просто может захотеть отдохнуть без гостей). Так же через письма обмениваются телефонами и/или договариваются о встрече.

Договариваться о приезде лучше заранее, хотя некоторые участники готовы принять гостя и буквально через пару часов после того, как договорились на сайте.

Также существует возможность, написать отзыв о другом участнике. В отзыве коротко описывается, как и где встретились с участником, чем он заинтересовал, были ли негативные моменты, общее впечатление от общения.

Резиденты клуба должны уважительно относиться друг к другу, считаться с привычками и традициями друг друга, ведь часто таким образом встречаются представители различных культур.

Деньги за проживание не берутся, но, по договорённости с хозяином, от гостя может потребоваться покупать себе еду, мыть за собой посуду и т. п.

Через клуб можно не только найти где остановиться, но и договориться с местными жителями о встрече. Таким образом, даже те, кто не может принимать гостей, тоже могут участвовать в обмене гостеприимством: они могут встречаться с путешественниками, рассказывать им о городе, показывать интересные места.

1.1 Постановка цели задач

Планирование неотъемлемая часть процесса разработки. Реализация, напрямую зависит от того, насколько качественно и последовательно распределены задачи для успешного выполнения проекта. И именного поэтому, следует разработать чётко выработанный план.

Эмоциональное состояние человека, также является одним из важнейших факторов процесса разработки проекта, существование четкого

плана и видение, какое большое количество задач выполняется при правильной расстановке приоритетов и выделения главного – все это положительно влияет на человеческое настроение, отсюда следует простой вывод: постоянное планирование помогает избежать лишней прокрастинации.

Идея проекта состоит в том, чтобы активизировать межкультурный обмен и общение людей в дружеской неформальной обстановке. Это даёт возможность реализовать естественное желание узнавать что-то новое, обмениваться опытом в различных сферах жизни. В том числе, благодаря такому общению людей из разных стран, представителей разных культур и национальностей, они имеют возможность взглянуть на многие международные проблемы с разных сторон.

В профиле участник может рассказать о своих увлечениях, опыте путешествий, владении языками, разместить фотографии и другую информацию.

Также в профиле указывается, может ли участник принимать гостей. Если может, то он указывает количество, желательный пол гостей, условия проживания, какой транспорт ходит до его дома и т. п. Если участник не может принимать гостей, то он может поставить статус «готов встретиться и пообщаться».

Каждый участник после знакомства с другим участником может оставить о нём отзыв. Отзыв показывается в профилях обоих участников. Рекомендуется оставлять отзывы только после личного знакомства (не в сети) Зарегистрированным участникам доступен поиск по анкетам по разным параметрам: географическое положение, информация об участнике, возможность принять гостей и т. п.

Большое внимание уделяется обеспечению безопасности участников при путешествиях. Для этого существует несколько инструментов, дающих возможность узнать мнение других участников об интересующем пользователе, и система верификации имени и адреса пользователя.

1.2 Выбор средств и технологий

В данном подразделе я опишу вам, какие технологии были использованы мною, при разработке дипломного проекта, их основные достоинства и недостатки. Также поясняется, почему были выбраны именно эти технологии, и за какую часть в приложении они несут ответственность.

Каждое веб-приложение состоит из двух частей: фронтенда – клиентской части веб-приложения, которая содержит всё, что вы видите и с чем взаимодействуете в браузере как пользователь приложения, и бэкенда, или серверной части, в которой хранятся данные веб-приложения, обновляются и обрабатываются, а затем передаются клиентской части.

Разработчикам часто приходится принимать решения, которые повлияют на всю архитектуру приложения. Веб-разработчикам важно выбрать

правильное место для реализации логики и рендеринга приложения. Это может быть непросто, так как сайт можно создать разными путями.

При выборе подхода для рендеринга нужно понимать разницу между возможными вариантами, чтобы не прогадать с производительностью.

SSR (Server-Side Rendering, серверный рендеринг) – рендеринг на сервере клиентской части или универсального приложения в HTML;

CSR (Client-Side Rendering, рендеринг на клиенте) – рендеринг приложения на стороне клиента (в браузере), обычно с помощью DOM.

Пререндеринг – запуск клиентского приложения во время сборки для сохранения его начального состояния в виде статического HTML.

При серверном рендеринге в ответ на запрос на сервере генерируется весь HTML страницы. Это исключает необходимость дополнительных запросов, данных со стороны клиента, так как сервер берёт всю работу на себя, прежде чем отправить ответ.

Такой подход позволяет добиться быстрой первой отрисовки и первой содержательной отрисовки. Выполнение логики страницы и рендеринг на сервере позволяют избежать отправки клиенту большого количества скриптов, что приводит к меньшему времени до интерактивности. И это логично, ведь при серверном рендеринге пользователю отсылаются только текст и ссылки. Этот подход хорошо сработает на широком диапазоне устройств и сетевых условий, а также откроет возможности для интересных браузерных оптимизаций вроде потокового парсинга документа.

Статический рендеринг происходит на этапе сборки и предоставляет быструю первую отрисовку, первую содержательную отрисовку и время до интерактивности – при условии, что количество клиентского кода ограничено. В отличие от серверного рендеринга здесь удаётся добиться стабильно быстрого времени до первого байта, так как HTML-код страницы не должен генерироваться на лету.

Как правило, статический рендеринг подразумевает предварительное создание отдельного HTML-файла для каждого URL. Поскольку HTML-ответы созданы заранее, статический рендеринг можно развернуть на нескольких CDN, чтобы воспользоваться преимуществом кеширования.

Для статического рендеринга существуют самые разные решения. Инструменты вроде Gatsby разработаны так, чтобы создавать впечатление динамического рендеринга. Другие, вроде Jekyl и Metalsmith, принимают свою статическую природу и предлагают подход, основанный в большей степени на шаблонах.

Но у такого способа рендеринга есть один недостаток – необходимо заранее создать HTML-файлы для всех возможных URL.

Это может быть очень сложно или даже невыполнимо, если вы не можете заранее сказать, какие URL возможны, или если у вас сайт с большим количеством уникальных страниц.

1.2.1 Язык разметки HTML

HTML (HyperText Markup Language) – язык гипертекстовой разметки, предназначенный для размещения объектов web-страницы в определенном порядке. Этот язык в первую очередь позволяет определять тип помещаемого объекта, например, ссылка, рисунок, медиа-файл, какой-либо скриптовый файл, либо самый обычный текст. Это все осуществляется при помощи набора тегов. Также этот язык является связующим звеном между страницей браузера и дополнительными технологиями, которые будут использоваться.

Ни одна современная web-страница не обходится без использования этой разметки. Для разработки предложенного проекта был использован стандарт последней версии – HTML5, который отличается от своих предшественников более строгими правилами, является продуктом, который сочетает в себе свойства и синтаксические нормы стандартов HTML и XHTML, а также направлен на поддержку большего числа мультимедийных технологий

При написании HTML мы задаём структуру контента. Определяем всякие абзацы, списки, таблицы и заголовки. Писать семантический код – значит уметь подобрать правильный элемент для определения нужной структуры. Совместимость от этого тоже выигрывает.

У каждого элемента есть свои общепризнанные значение и функция. Например, тег <p> отвечает за абзац статического текста, элемент <a> – интерактивный и может загрузить новый ресурс. Не будь оно общепризнанным, браузеры не смогли бы определить оформление и функциональность по умолчанию для многих элементов.

HTML также задёт смысл контента. У большинства HTML-элементов есть неявная роль, определяющая назначение элемента. Например, неявная роль элемента <a> – «ссылка», у <img> – «графика» или «изображение», а у <header> – это «баннер».

При написании HTML разработчики всегда стараются написать код так, чтобы он был семантичным.

Подразумеваемый смысл вполне ясен: использование HTML так, чтобы он был читаемым, описание вещей простым языком. Но «семантика» означает совсем не это. Можно было сказать просто «хорошо написанный» или «проверенный редактором». Это подошло бы лучше. Разметка, которую мы сегодня называем «несемантичной», на самом деле не такова: для конечных пользователей-то смысл есть, просто он передается слишком многословной разметкой.

Употребление слова «семантичный», при котором речь идет о простоте, и подразумеваемое при этом значение этого слова говорят о глубокой путанице насчет того, как работает язык, роли разработчиков в донесении этого значения и реального потенциала смыслового развития для веба.

Огромная значимость высокоуровневых систем вроде HTML в том, что они дают возможность манипулировать множеством жутких программных

слоев, не заставляя в них углубляться. Даже средь исканий объяснения структуры слоев веб-платформы в низкоуровневых терминах мы не вправе забывать важности высокоуровневых, по большей части декларативных форм.

Простой для модификации и повторения формат, дающий разработчикам (косвенную) власть над этим программными слоями, причем так, что это оказывается по силам даже тем, кто и не представляли себя «разработчиками», порождает монументальную мощь. Мощь в стиле «Вижуал-Бейсика»: то, благодаря чему сообразительные люди могут полностью обойтись своими силами даже при технологических ограничениях. Ctrl-R в роли команды «make all» для веба – не идеал, но и он дает огромные возможности. Нынешняя революция в инструментах разработчика – тем более. Влияние непосредственности нельзя переоценить. Слово Брету Виктору: «творцам нужна мгновенная связь». Декларативные формы сполна оправдывают себя, когда отвечают нуждам пользователей.

HTML превращает смертных в героев. Подобно тому, как это бывает в завязках дешевых комиксов, непосредственность и нетребовательность HTML дают «не-разработчикам» способность создавать вещи, которыми они могут делиться с другими людьми. Вещи, которые нужны людям. Именно этим отличаются хорошие инструменты. Он даже и на код не очень-то похож.

Значимость HTML с точки зрения разработчиков состоит полностью в показеразных штук на экранах, чтобы смотреть на них и взаимодействовать с ними. У модных течений «связанных данных» и «семантического HTML» есть странная склонность забывать об этом. Главная причина, по которой люди создают что-то в вебе – желание, чтобы другие люди нашли это что-то, и оно обогатило бы их жизнь. Обычно речь о визуальной и интерактивной значимости: показать, ввести в контекст, отобразить.

Это диалог между разработчиком и пользователями. Диалог, в котором разработчик орудует зачастую неподходящими средствами, чтобы сообщить что-то – чаще всего визуально. То, что достается всем остальным (поисковикам, издателям и т.д.) из этой расстановки – лишь побочный эффект, эхо изначального диалога.

Разработчики пытаются добиться от пользователей, чтобы те восприняли именно то, что сами разработчики имели в виду. Что в статье важно. Что тут навигация, а что – основной контент. Какие элементы формы действительно обязательны. Для пользователей всё это – не «технические подробности». Семантичный контент – это то, что воспринимают конечные пользователи.

Пусть это отложится как следует. С точки зрения пользвателей, слайды PowerPoint или формы в PDF могут передавать практически ту же семантику, что HTML – визуальный и интерактивный контент, воспринимаемый как «значение» контента большую часть времени.

Я утверждаю, что HTML преуспел потому, что превратил людей, ничего не знавших о программировании, в великих мастеров коммуникации. Они

вдруг стали способны создавать формы, и чекбоксы, и абзацы, и картинки из ничего.

Дабы устоять пред искушением недооценить визуальные и интерактивные соглашения, представьте себе мир без них. Или мир, где у браузеров есть JS-овые API для перехода к другим документам, но нет встроенной системы, связывающей клики с переходами. Конечно, стихийный порядок мог бы возникнуть, но едва ли. За всё время, сколько существует флэш-контент, так и не появилось стандартного способа для «ссылок» между его единицами. Можно смело сказать, что обеспечение пользователей наглядными возможностями, чтобы помочь им решать свои задачи, и есть то, благодаря чему HTML преуспел.

Эти наглядные возможности, с точки зрения пользователей, и есть семантические соглашения веба. Это строительные блоки визуального и интерактивного языка. Они – и содержание, и значение.

1.2.2 Язык стилей CSS

CSS (Cascading Style Sheets) – каскадные таблицы стилей – формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Главная задача CSS – это корректное визуальное представление страницы перед пользователем согласно заданным стилевым правилам.

CSS работает с большим множеством элементов страницы: шрифты, цвета фона и символов, поля, таблицы, строки, высота и ширина элементов, отступы, с изображениями, их корректным отображением, с позиционированием элементов и многим другим. При помощи CSS можно изменить любой компонент страницы и заставить его выглядеть так, как оно требуется для дизайна страницы.

Основная цель CSS – это разгрузить файл HTML, поскольку раньше логическая разметка и описание внешнего вида страницы хранилось в одном файле и значительно затрудняло поддержку сайта и его дальнейшую разработку.

CSS является неотъемлемой частью современного web-программирования, так как от визуального содержания страницы зависит его удобство в использовании и популярность среди пользователей. Чем лучше сделан сайт с визуальной точки зрения, тем он более конкурентоспособен на поисковых платформах.

1.2.3 Высокоуровневый язык программирования Python

Python – это мощный высокоуровневый объектно-ориентированный язык программирования, созданный Гвидо ван Россумом. Python – это язык общего назначения. На нем пишут веб-приложения с использованием различный фреймворков, системные утилиты и приложения для

автоматизации различных действий. он имеет широкий спектр приложений от веб-разработки, научных и математических вычислений для настольных графических пользовательских интерфейсов.

Он также хорошо подходит для создания прототипов. То есть, сначала создается прототип на Python, затем концепцию можно перенести на более быстрые и компилируемые языки программирования. При помощи этого языка можно создавать desktop-приложения с графическим интерфейсом и писать игры, для чего существует специальная библиотека. Основы алгоритмизации и программирования на языке Python подходят для создания приложений для мобильных устройств. Во многом Python популярен потому, что он позволяет программисту не погружаться в дебри компьютерных технологий, а сосредоточиться на своей прикладной задаче. С другой стороны, именно технические тонкости зачастую определяют преимущества и ограничения разных языков, поэтому крутому специалисту следует с ними познакомиться.

Как уже было сказано выше, Python – это высокоуровневый язык, который не взаимодействует с компьютерным «железом» напрямую. Ваш код компилируется в так называемый байт-код, с которым работает интерпретатор Python, передающий инструкции процессору. Многие специалисты называют эту утилиту виртуальной машиной (ВМ), хотя между этими понятиями есть разница.

Именно интерпретатор Python распоряжается имеющимися ресурсами, позволяя программам записывать новые данные и удаляя устаревшие. Чтобы понять, какая информация зря занимает место, ВМ использует структуру PyObject. Она учитывает существующие между объектами связи, исходя из того, что если на какой-то из них нет ссылок, то и смысла в его существовании нет.

Когда переменная делает ссылку на объект, передаёт его в качестве аргумента или добавляет в список, PyObject добавляет единицу на соответствующем счётчике. Если счётчик оказывается на нуле, занятая объектом память освобождается. Это может запустить цепочку удалений, если в устаревшем объекте были ссылки на другие. С его удалением соответствующие счётчики уменьшатся на единицу, в каких-то случаях – до нуля. Цикл повторится.

Программист может в любой момент узнать количество отсылок к тому или иному объекту через метод getrefcount(). Полученное число следует уменьшить на единицу – ведь отправленная команда также создала дополнительную связь.

Ещё один важный момент в управлении памятью в Python на протяжении многих лет вызывает горячие споры. Речь о глобальной блокировке интерпретатора (Global Interpreter Lock, GIL), которая, с одной стороны, играет ключевую роль в корректном выполнении приложений, а с другой – создаёт те самые ограничения, о которых мы говорили в начале статьи.

Представьте, что два потока одновременно запишут данные в одну ячейку. Очевидно, что результат будет неприемлем ни для одного приложения. Операции, которые угрожают подобными коллизиями, называются потоконебезопасными. На практике они вызывают порчу памяти и критические ошибки ПО. Более того, отследить эти ошибки непросто, поскольку они проявляются случайным образом.

Принцип GIL обеспечивает потокобезопасность при подсчёте ссылок, не позволяя нескольким объектам менять свои значения одновременно. Для этого в каждый момент интерпретатор открывает доступ к объектам только одному потоку. В некоторых ситуациях (например, в операциях ввода/вывода) блокировка приостанавливается, чем пользуются продвинутые Python-программисты. Но это исключения, а в большинстве случаев GIL твёрдо контролирует соблюдение очереди.

В результате Python фактически закрыт для многопоточных сценариев, зато однопоточные выполняются значительно быстрее, чем с другими техниками потокобезопасности (например, при использовании системы отдельных запретов). Именно этот довод играет ключевую роль для автора Python Гвидо ван Россума, который не раз говорил оппонентам, что GIL останется в языке до тех пор, пока программисты не смогут иначе гарантировать нынешнюю скорость однопоточных операций.

Альтернатива механизму подсчёта ссылок и, как следствие, глобальной блокировке – это «мусоросборочные» механизмы (garbage collection) вроде тех, что используются в Java или языках .NET. Как можно догадаться по названию, эти компоненты сами находят в памяти, устаревшие данные и удаляют их. Это более ресурсоёмкая задача, поэтому, например, Android-приложения на языке Kotlin (использует сборщик мусора) могут работать медленнее своих iOS-аналогов на SWIFT (считает ссылки).

Зато такие программы получают все преимущества многопоточных операций, а близость Java к «физическому» компьютеру ускоряет вычисления. С другой стороны, зачем это обычному пользователю, который сидит на компьютере с четырьмя ядрами… В общем, вы сами видите, что этот спор может продолжаться вечно.

Самое главное, что даже с этими запретами Python остаётся самым эффективным языком с точки зрения скорости разработки. Возможность абстрагироваться от компьютерной архитектуры и писать код, с которым легко разберётся сторонний специалист, позволяет программистам быстрее создавать свои продукты. Это экономит деньги заказчика ПО, значит, при прочих равных рынок будет склоняться в пользу Python. А там и производительность подтянется

1.2.4 Фреймворк Django

Веб-фреймворк – инструмент, облегчающий процесс написания и запуска веб-приложения. Вам не нужно самостоятельно писать кучу кода и тратить время на поиск потенциальных просчётов и ошибок.

На рассвете эры веб-разработки все приложения писались вручную, и только разработчик приложения мог изменить или развернуть его. Веб-фреймворки позволили выбраться из этой западни. С 1995 года вся морока, связанная с изменением структуры приложения, была приведена в порядок благодаря появлению общего подхода к разработке веб-приложений. В это время появились языки для веба. Сейчас их разнообразие позволяет выбрать подходящий как для статических, так и для динамических страниц. В зависимости от поставленной задачи, вы можете выбрать один фреймворк, покрывающий все нужды, или совместить несколько.

Серверные фреймворки. Правила и архитектура таких фреймворков не даёт возможности создать веб-приложение с богатым интерфейсом. Они ограничены в своей функциональности, однако вы всё равно можете создавать простые страницы и разные формы. Также они могут формировать выходные данные и отвечать за безопасность в случае атак. Всё это определённо может упростить процесс разработки. Серверные фреймворки в основном отвечают за отдельные, но критически важные части приложения, без которых оно не сможет нормально работать. Вот несколько самых популярных фреймворков и языки, с которыми они работают:

- Django – Python;
- Zend – PHP;
- Express.js – JavaScript;
- Ruby on Rails – Ruby.

Клиентские фреймворки. В отличие от серверных, клиентские фреймворки никак не связаны с логикой приложения. Этот тип фреймворков работает в браузере. С их помощью можно улучшить и внедрить новые пользовательские интерфейсы. Фронтенд-фреймворки позволяют создавать разные анимации и одностраничные приложения. Все клиентские фреймворки отличаются по функциональности и использованию. Вот некоторые из них:

- Backbone+Marionette;
- Angular;
- Ember.js;
- Vue.js.

Все эти фреймворки используют JavaScript.

Несмотря на то, что все фреймворки отличаются друг от друга и выбрать какой-нибудь из них может быть очень сложно, есть несколько вещей, общих для них всех. Речь идёт об архитектуре и особенностях, которые так же важны, как и функции. Архитектура почти всех фреймворков основана на декомпозиции нескольких отдельных слоёв (приложения, модули и т.д.), что означает, что вы можете расширять функциональность исходя из своих потребностей и использовать изменённую версию вместе с кодом фреймворка или использовать сторонние приложения. Такая гибкость

является ещё одним ключевым преимуществом фреймворков. Существует множество open-source сообществ и коммерческих организаций, которые создают приложения или расширения для популярных фреймворков, например, Django REST Framework, ng-bootstrap и т.д.

Как мы уже убедились, выбор и использование веб-фреймворка может стать тем ещё испытанием. Однако сам процесс не такой уж и сложный, как могло показаться. Есть достаточное количество документов, библиотек и руководств, призванных помочь изучить фреймворк и ответить на все возникающие вопросы. Существуют сайты, которые предоставляют быстрое введение в любой фреймворк.

Так как, я буду писать проект на Pyhton, то мой выбора пал на Django.

Django - это фреймворк для создания веб-приложений с помощью языка программирования Python. Django был создан в 2005 году, когда веб-разработчики из газеты Lawrence Journal-World стали использовать Python в качестве языка для создания веб-сайтов. А в 2008 году вышел публичный первый релиз фреймворка.

На сегодняшний день он продолжает развиваться. Так, текущей версией фреймворка на момент написания этой статьи является версия 2.0, которая вышла 3 декабря 2017 года. Ну и также постоянно выходят подверсии. Django довольно популярен. Он используется на многих сайтах, в том числе таких, как Pinterest, PBS, Instagram, BitBucket, Washington Times, Mozilla и многих других.

Фреймворк Django реализует архитектурный паттерн Model-View-Template или сокращенно MVT, который по факту является модификацией распростаненного в веб-программировании паттерна MVC (Model=-View-Controller).

Django может быть использован для создания практически любого типа веб-сайта - от систем управления контентом и вики, до социальных сетей и новостных сайтов. Он может работать с любой клиентской платформой и может доставлять контент практически в любом формате (включая HTML, RSS-каналы, JSON, XML и так далее).

Начинка предоставляет выбор практически для любой функциональности, которую вы можете захотеть (например, несколько популярных баз данных, шаблонизаторы и так далее), он также может быть расширен для использования других компонентов, если это необходимо.

Django помогает разработчикам избегать многих распространенных ошибок безопасности, предоставляя инфраструктуру, которая была разработана для «правильного решения», чтобы автоматически защитить сайт. Например, Django обеспечивает безопасный способ управления учетными записями пользователей и паролями, избегая распространенных ошибок, таких как включение информации о сеансе в файлы cookie, где она уязвима (вместо этого куки-файлы содержат только ключ, а фактические данные хранятся в базе данных), или хранение паролей в открытом виде, вместо их хэшей.

Хэш пароля - это значение фиксированной длины, созданное путем обработки пароля через криптографическую хэш-функцию. Django может проверить правильность введенного пароля, пропустив его через хэш-функцию и сравнив вывод с сохраненным значением хэша. Благодаря «одностороннему» характеру функции, даже если сохраненное хэш-значение скомпрометировано, злоумышленнику будет сложно извлечь исходный пароль.

Django обеспечивает защиту от многих уязвимостей по умолчанию, включая SQL-инъекцию, межсайтовый скриптинг, подделка межсайтовых запросов и кликджекинг (см. Website security для получения дополнительной информации об этих атаках).

Django использует компонентную “shared-nothing” архитектуру (каждая часть архитектуры не зависит от других, и следовательно, может быть заменена или изменена при необходимости). Четкое разделение между различными частями означает, что оно может масштабироваться для увеличения трафика путем добавления оборудования на любом уровне: кеширующие серверы, серверы баз данных или серверы приложений. Некоторые из самых посещаемых сайтов успешно масштабируются Django для удовлетворения своих требований (например, Instagram и Disqus, назовём лишь два).

Код Django написан с использованием принципов и шаблонов дизайна, которые поощряют создание поддерживаемого и многоразового кода. В частности, он использует принцип Do not Repeat Yourself (DRY), поэтому нет ненужного дублирования, что уменьшает количество кода. Django также способствует группировке связанных функций в многоразовые «приложения» и на более низком уровне группирует связанный код модулей в соответствии с Model View Controller (MVC) паттерном.

Django написан на Python, который работает на многих платформах. Это означает, что вы не привязаны к какой-либо конкретной серверной платформе и можете запускать приложения во многих вариантах Linux, Windows и Mac OS X. Кроме того, Django хорошо поддерживается многими поставщиками веб-хостинга, которые часто предоставляют определенную инфраструктуру и документацию для размещения сайтов Django.

Веб-фрейморки часто можно поделить на "упрямые" и "не упрямые".

Упрямые фреймворки - это те, у которых есть мнение о «правильном пути» для решения какой-либо конкретной задачи. Они часто поддерживают быструю разработку в определенной области (решение проблем определенного типа), потому что правильный способ сделать что-либо обычно хорошо понимается и хорошо документируется. Однако они могут быть менее гибкими при решении проблем за пределами их основной сферы и, как правило, предлагают меньше вариантов того, какие компоненты и подходы они могут использовать.

Напротив, у неупрямых фреймворков гораздо меньше ограничений на лучший способ склеивания компонентов для достижения цели или даже того, какие компоненты следует использовать. Они облегчают разработчикам

использование наиболее подходящих инструментов для выполнения конкретной задачи, хотя и за счет того, что вам нужно самим найти эти компоненты.

Django «немного упрямый» и, следовательно, обеспечивает «лучшее из обоих миров». Он предоставляет набор компонентов для обработки большинства задач веб-разработки и один (или два) предпочтительных способа их использования. Однако такая архитектура Django означает, что вы обычно можете выбирать из нескольких различных опций или при необходимости добавлять поддержку для совершенно новых.

Веб-приложения Django обрабатывают и запрашивают данные через объекты Python, называемые моделями. Модели определяют структуру хранимых данных, включая типы полей и, возможно, их максимальный размер, значения по умолчанию, параметры списка выбора, текст справки для документации, текст меток для форм и т. д. Определение модели не зависит от базовой базы данных - Вы можете выбрать один из нескольких компонентов вашей настройки проекта. После того, как вы выбрали базу данных, которую хотите использовать, Вам не нужно напрямую обращатся к ней - вы просто пишете свою структуру модели и другой код, а Django обрабатывает всю грязную работу по обращению к базе данных за вас.

1.3 Обзор существующих аналогичных программных продуктов

Одной из самых популярных сетей гостеприимства является американская сеть CouchSurfing.

CouchSurfing – это одна из самых крупнейших гостевых сетей, существующая в виде онлайн-сервиса. Объединяет более 14 миллионов человек в 200 000 населенных пунктах. Члены сети предоставляют друг другу помощь и ночлег во время путешествий и организуют совместные путешествия.

До 2011 Couchsurfing являлся некоммерческой организацией, зарегистрированной на территории штата Нью-Гэмпшир в США. Сайт совершенствовался во многом благодаря работе волонтёров. В 2011 году статус организации был сменён на B-corporation. Этот статус подразумевает, что целями организации является как создание пользы для общества, так и получение прибыли её владельцами. По утверждению владельцев организации, переход к новому статусу вызван тем, что статус некоммерческой организации не позволяет достаточно гибко ею управлять.

Всего за время существования Couchsurfing.com было привлечено 22,5 миллионов долларов инвестиций - 7,6 миллионов в 2011 году и 15 млн в 2012 году.

Еще одной из самых крупных сетей является – Hospitality club. Она существует с июля 2000 года и в настоящее время объединяет более 320000 членов из более чем 200 стран.

Главная цель сообщества – создать мир без предрассудков и расизма, развивать толерантность и взаимопонимание, знакомя и объединяя людей разных национальностей и культур – путешественников и аборигенов, гостей и хозяев.

Проект поддерживается группой добровольцев.

Регистрация и участие в проекте бесплатны. Заполняется профиль члена Клуба, в котором указываются паспортные данные и другая информация (о своих хобби, о своих прошлых туристических поездках и т. д.), и обязательно, о возможности и условиях для приема других членов сообщества в гости. Паспортные данные выводятся в общий просмотр только по пожеланию.

Члены сети добровольно предоставляют друг другу помощь и ночлег во время путешествий и организуют совместные путешествия и экскурсии. Просить денег за эти услуги запрещено правилами сообщества (однако, по взаимной договорённости, иногда гости платят за телефон или еду).

Каждый член сети может оставить комментарий о гостившем у него туристе или о принимавших его хозяевах прямо у них в профиле. Человека, получившего много негативных отзывов, начинают сторониться. Также, для обеспечения безопасности в сообществе, добровольцы проводят проверку паспортных данных, модерацию сообщений и фильтровку спама.

При помощи общих форумов на сайте участники обсуждают свои поездки, делятся впечатлениями, организуют встречи и лагеря.

2 Проектирование программного продукта

Под программным продуктом понимается совокупность программ, выполняемых вычислительной системой.

К программному обеспечению относится также вся область деятельности по проектированию и разработке ПО:

- технология проектирования программ;
- методы тестирования программ;
- методы доказательства правильности программ; анализ качества работы программ;
- документирование программ;
- разработка и использование программных средств, облегчающих процесс проектирования программного обеспечения, и многое другое.

Программное обеспечение – неотъемлемая часть компьютерной системы. Оно является логическим продолжением технических средств.

Сфера применения конкретного компьютера определяется созданным для него программным обеспечением.

Сам по себе компьютер не обладает знаниями ни в одной области применения. Все эти знания сосредоточены в выполняемых на компьютерах программах. Программное обеспечение современных компьютеров включает миллионы программ – от игровых до научных.

Существует два основных типа программного обеспечения: системное (называемое также общим) и прикладное (называемое специальным). Каждый тип программного обеспечения выполняет различные функции.

Системное программное обеспечение – это набор программ, которые управляют компонентами компьютера, такими как процессор, коммуникационные и периферийные устройства. Программистов, которые создают системное программное обеспечение, называют системными программистами.

К прикладному программному обеспечению относятся программы, написанные для пользователей или самими пользователями, для задания компьютеру конкретной работы. Программы обработки заказов или создания списков рассылки – примеры прикладного программного обеспечения. Программистов, которые пишут прикладное программное обеспечение, называют прикладными программистами.

Оба типа программного обеспечения взаимосвязаны и могут быть представлены в виде диаграммы. Каждая область тесно взаимодействует с другой. Системное программное обеспечение обеспечивает и контролирует доступ к аппаратному обеспечению компьютера.

Прикладное программное обеспечение взаимодействует с аппаратными компонентами через системное.

Конечные пользователи в основном работают с прикладным программным обеспечением. Чтобы обеспечить аппаратную совместимость, каждый тип программного обеспечения разрабатывается для конкретной аппаратной платформы.

2.1 Структура программного продукта

Начнем проектирование web-сервиса с помощью фреймворка Django. Но прежде чем начать, у нас должно быть установлено программное обеспечение языка Python. Высокоуровневый язык программирования Python, первично ориентированный на повышение производительности разработчика и читаемости кода, что является его главной особенностью.

Синтаксис ядра Python минималистичен и прост в своем исполнении. В то же время стоит отметить, что стандартная библиотека включает большой объём полезных и необходимых для реализации необходимого объема работы функций.

Python в свою очередь поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное и функциональное, тем самым становясь более гибким и удобным для реализации задуманного.

Основные архитектурные особенности данного языка это в свою очередь это динамическая типизация присущая немногим объектно-ориентированным языкам, а также автоматическое управление памятью и

корректный механизм обработки исключений, что является ключевым фактором для работы.

Код в программном обеспечении, написанном на Python организовывается в функции и классы, которые могут объединяться в модули, с которыми впоследствии и проводится работа.

Далее загружаем последнею версию фрейворка Django с официального сайта. После корректной установки необходимого программного обеспечения, разрабатываем первичный дизайн web-сервиса.

2.2 Функциональная структура web-приложения

Функциональная структура web-приложения предназначена для того, чтобы наглядно показать какие функции может предоставить приложение. Глядя на такую структуру, сразу становится ясным стоит ли использовать это приложение и приведет ли оно к желаемому результату.

Процесс аутентификации представляет собой процедуру проверки подлинности пользователя. Другими словами, пользователю необходимо доказать, что он именно тот человек, которому принадлежит идентификатор. В данном приложении в качестве проверяемого значения был использован пароль. Программа сравнивает пароль, введенный пользователем и пароль, хранящийся в базе данных, и в случае совпадения осуществляет процедура авторизации.

Авторизация – это предоставление доступа к какому-либо ресурсу, в данном случае к аккаунту пользователя. Этот процесс осуществляется после идентификации и аутентификации, которые предшествуют его выполнению.

Далее пользователь, попадая в систему, может выбрать на какой функцией он желает воспользоваться.

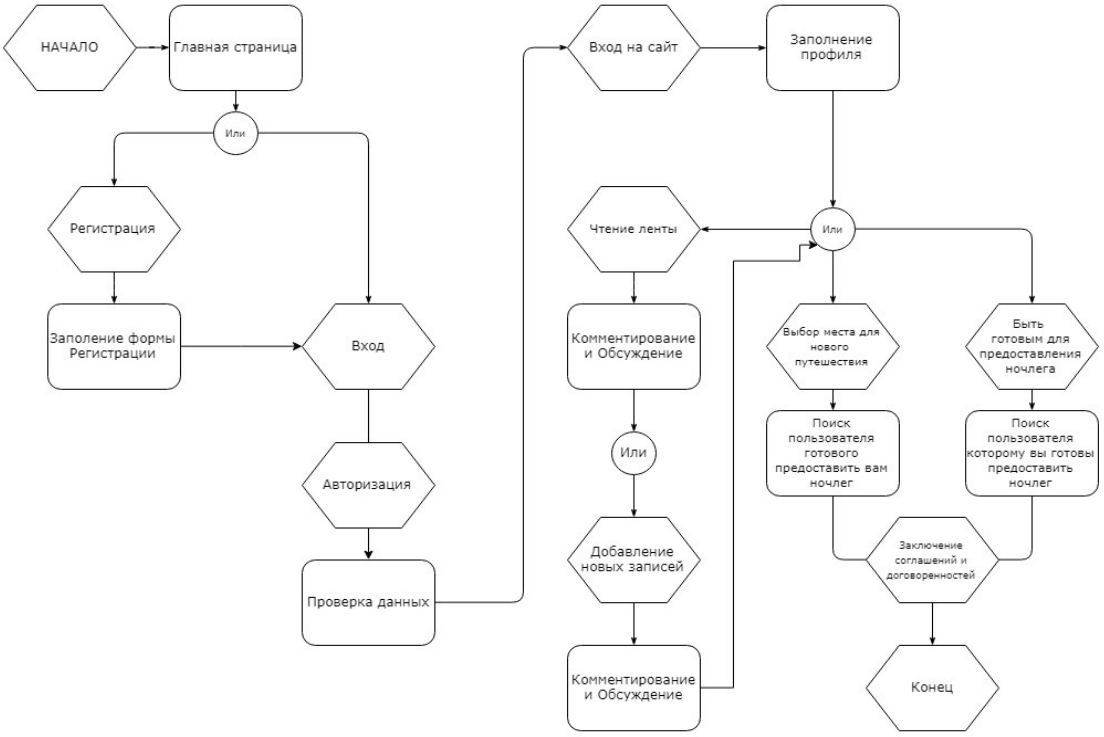

Рисунок 2.2.1 – блок-схема web-сервиса

3 Разработка программного продукта

В основе любого web-проекта (да и не только web) лежит клиент-серверная архитектура.

Исторически, первым вариантом клиент-серверной архитектуры являлась так называемая «Однозвенная» архитектура, характеризующаяся тем, что клиент не нес никакой функциональной нагрузки, кроме отображения информации, предоставляемой мейнфреймом (сервером). Примером такой архитектуры может являться терминальный доступ к удаленному серверу или удаленный рабочий стол. В этом случае весь объем вычислительной нагрузки приходился на сервер. Следующим шагом в развитии клиент-серверной архитектуры было появление так называемой двухзвенной архитектуры.

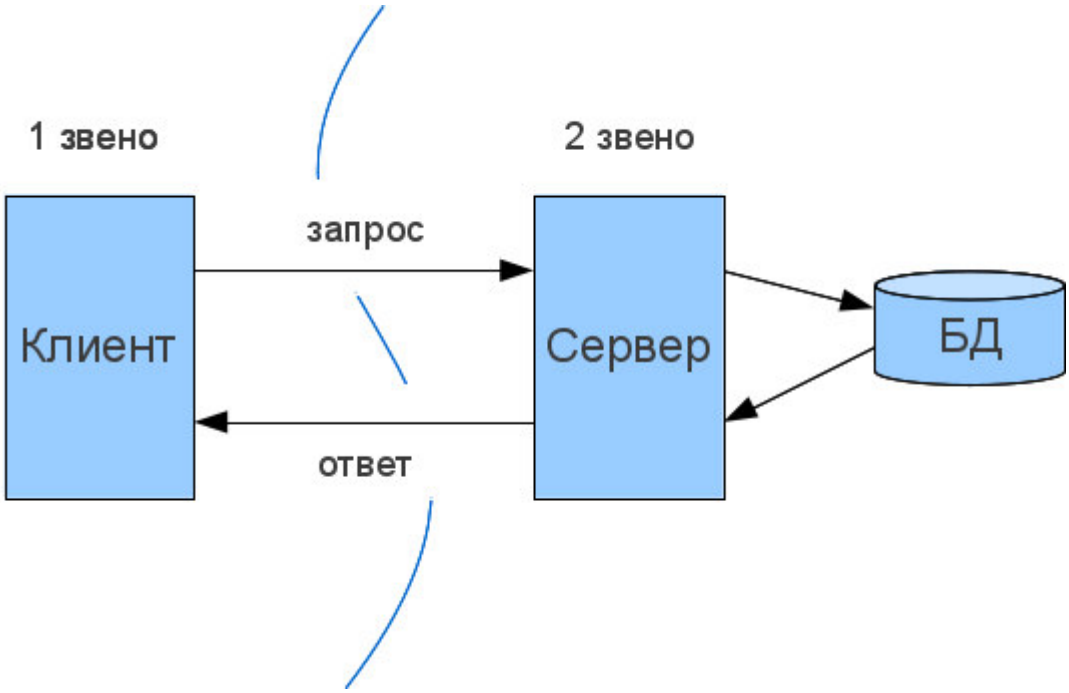

Рисунок 3.1 – «Однозвенная» архитектура

Особенностью данного подхода является использование «толстых» клиентов, на которые возлагались основные задачи по отображению информации пользователю и обработке всех данных. Центральный сервер реализовывал лишь функции хранения и предоставления данных. Этот подход являлся наиболее распространенным решением для корпоративных распределенных вычислительных систем вплоть до начала 2000-х годов. Поскольку большинство логики клиент-серверного приложения находится в клиентской части, клиентская рабочая станция несет ответственность за большую часть обработки. Для оценки разделения объемов работ часто используется соотношение 80/20: сервер базы данных обычно выполняет порядка двадцати процентов работы. Несмотря на это, база данных часто становится узким местом производительности в этих средах. Двухуровневая

клиент-серверная система часто требует, чтобы каждый клиент устанавливал свои собственные соединения с базой данных. Постоянная поддержка такого множества соединений с базой данных стоят дорого, и потребность в ресурсах иногда может привести к перегрузке сервера баз данных и задержки обработки запросов всех пользователей.

Одной из основных причин отказа от двухзвенного клиент-серверного подхода было постоянное увеличение расходов на поддержание логики работы приложения на рабочих станциях пользователей. Поскольку код приложения реализуется в каждом клиенте, каждое обновление для приложения требует переустановки клиентского программного обеспечения на всех рабочих станциях. В больших средах, это приводит к высокой сложности администрирования.

В ответ на затраты и ограничения, связанные с двухуровневой клиент-серверной архитектурой, зародилась концепция компонентно-ориентированной разработки приложений.

Многоуровневая клиент-серверная архитектура обеспечила переход к объектноориентированному подходу в задаче построения распределенных вычислительных систем. Применение такого подхода позволило распределить уровень бизнес-логики среди нескольких компонентов (часть – на клиенте, часть – на сервере) и сократить количество проблем при развертывании системы путем централизации большего количества логики на серверах.

Серверные компоненты, перешедшие на выделенные сервера приложений, обеспечили возможность управления пулами соединений с базой данных, облегчая задачу серверу баз данных посредством значительного уменьшения одновременного количества соединений (рис. 3.4), так как одно соединение может обеспечить работу нескольким клиентам.

В качестве примера рассмотрим поисковую машину в Интернете. Пользовательский интерфейс поисковой машины очень прост: пользователь вводит строку, состоящую из ключевых слов, и получает список заголовков web-страниц. Результат формируется из гигантской базы просмотренных и проиндексированных web-страниц.

Ядром поисковой машины является программа, трансформирующая введенную пользователем строку в один или несколько запросов к базе данных. Затем она помещает результаты запроса в список и преобразует этот список в набор HTML-страниц. В рамках модели клиент-сервер часть, которая отвечает за выборку информации, обычно находится на уровне обработки.

На смену двухуровневой, пришла трехуровневая архитектура клиент-сервер, в ней на каждом из серверов содержится один и тот же набор web-страниц, и всякий раз, когда одна из web-страниц обновляется, ее копии незамедлительно рассылаются на все серверы. Сервер, которому будет передан приходящий запрос, выбирается по правилу «карусели». Эта форма горизонтального распределения весьма успешно используется для выравнивания нагрузки на серверы популярных web-сайтов.

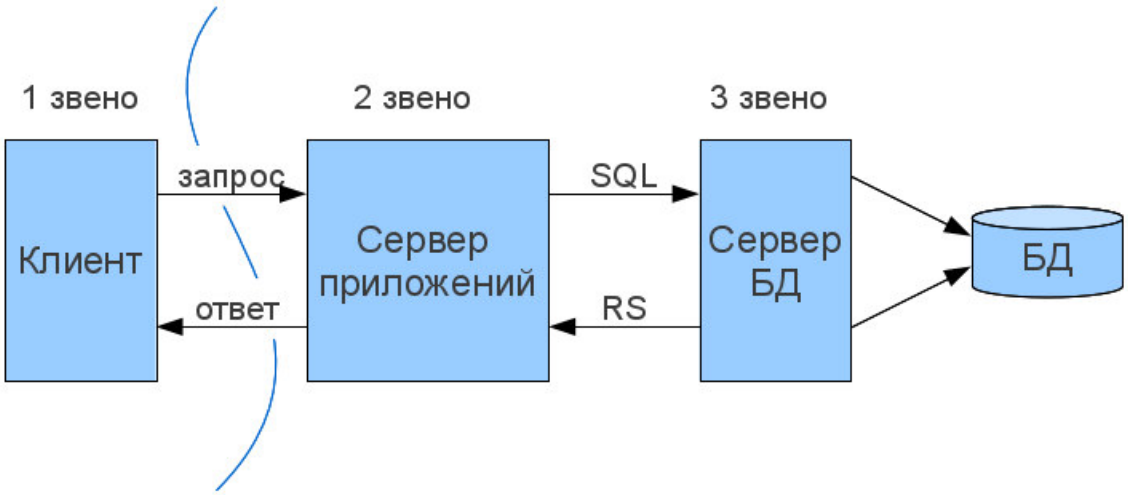

Рисунок 3.2 – трехуровневая архитектура клиент-сервер

Таким же образом, хотя и менее очевидно, могут быть распределены и клиенты. Для несложного приложения, предназначенного для коллективной работы, мы можем не иметь сервера вообще. В этом случае мы обычно говорим об одноранговом распределении. Подобное происходит, например, если пользователь хочет связаться с другим пользователем. Оба они должны запустить одно и то же приложение, чтобы начать сеанс. Третий клиент может общаться с одним из них или обоими, для чего ему нужно запустить то же самое приложение.

Имено по такой архитектуре будет строится мой web-сервис.

3.1 HTML компонент web-сервиса

Первым делом напишем «скелет» нашего сайта. Если открыть любую веб-страницу, то она будет содержать в себе типичные элементы, которые не меняются от вида и направленности сайта. Разница между строгим и переходным описанием документа состоит в различном подходе к написанию кода документа.

Строгий HTML требует жесткого соблюдения спецификации HTML и не прощает ошибок. Переходный HTML более «спокойно» относится к некоторым огрехам кода, поэтому этот тип в определенных случаях использовать предпочтительнее. В моем случае, у нас будет переходный HTML файл.

Для создания шаблона потребуется редактор, в который нужно будет вставить приведённый ниже код. Это может быть, как простой виндовский Блокнот, так и любой другой текстовый редактор. Я буду использовать редактор Sublime Text 3. Так как, Sublime Text поддерживает плагины на языке программирования Python.

Создаем главный html-шаблон, который будет выглядеть примерно так, для того чтобы упростить настройку стилей подключаем фреймворк Bootstrap:

```
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Nomad Trip</title>
    <link rel="shortcut icon" href="{% static 'img/icon.png' %}" type="image/png">
    <link href="{% static 'css/bootstrap-4/bootstrap.css' %}" rel="stylesheet">
    <link href="{% static 'css/bootstrap-4/bootstrap-grid.css' %}" rel="stylesheet">
    <link href="{% static 'css/bootstrap-4/bootstrap-reboot.css' %}" rel="stylesheet">
    <link href="{% static 'css/font-awesome.min.css' %}" rel="stylesheet">
    <link href="{% static 'css/styles.css' %}" rel="stylesheet">
    <link href="{% static 'css/zbs.css' %}" rel="stylesheet">
    <script src="{% static 'js/jquery-3.2.1.min.js' %}"></script>
    <script src="{% static 'js/bootstrap-4/popper.min.js' %}"></script>
    <script src="{% static 'js/bootstrap-4/bootstrap.min.js' %}"></script>
    <script src="{% static 'js/scripts.js' %}"></script>
</head>
<body>
    {% include 'blocks/navbar.html' %}
    {% include 'blocks/messages.html' %}

    {% block content %}
    {% endblock %}
    <footer id="footer" class="footer">
        <div class="container">
            <div class="row">
                <div class="col-lg-12">
                    <ul class="socials d-flex justify-content-center">
                        <li class="socials__item socials__item_fb">
                            <a href="#"></a>
                        </li>
                        <li class="socials__item socials__item_tw">
                            <a href="#"></a>
                        </li>
                        <li class="socials__item socials__item_g">
                            <a href="#"></a>
                        </li>
                    </ul>
                </div>
            </div>
            <div class="row">
                <div class="col-lg-4">
                    <div class="line"></div>
                </div>
                <div class="col-lg-4">
                    <div class="credits">
                        2019 @  NomadTrip.kz
```

Рисунок 3.1.1 – файл base.html

Тут видно какие статик файлы я подключил. styles.css и scripts.js - это пока пустые файлы, которые мы будем далее заполнять.

Есть один блок content, в котором будет основная часть страницы. Внутри body мы включаем другой шаблон для навигационного бара.

Этот шаблон будет лежать в папке templates/blocks в файле navbar.html. Так как мы подключили навбар в файле base.html, то он будет теперь во всех страницах.

Теперь начинаем писать код главной страницы нашего web-сервиса.

```
<body>
    <header id="header" class="header" style="background-color: #452467;">
        <div class="main-header">
            <div class="container">
            <div class="row">
                <div class="col-lg-4">
                    <a href="/"><img src="{% static 'img/logo.png' %}"></a>
                </div>
                <div class="col-lg-4">
                    <nav>
                        <ul class="menu d-flex justify-content-between">
                            <li class="menu__item">
                                <a href="/">
                                    на главную
                                </a>
                            </li>
                            <li class="menu__item">
                                <a href="#">
                                    о нас
                                </a>
                            </li>
                            <li class="menu__item">
                                <a  href="#">
                                    как это работает?
                                </a>
                            </li>
                        </ul>
                    </nav>
                </div>
            {% if user.is_authenticated %}
                <div class="col-lg-3 d-flex justify-content-between" >

                   <button class="auth">
                        <a href="{% url 'logout' %}">выйти</a>
                    </button>
                </div>
            {% else %}
                <div class="col-lg-3 d-flex justify-content-between" >
                    <button class="auth">
                        <a href="{% url 'login' %}">вход</a>
                    </button>
                    <button class="register">
                        <a href="{% url 'register' %}">регистрация</a>
                    </button>
            {% endif %}
                </div>
            </div>
            </div>
```

Рисунок 3.1.2 – файл navbar.html

```
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Nomad Trip</title>
    <link rel="shortcut icon" href="{% static 'img/icon.png' %}" type="image/png">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="
    sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
    <link href="https://fonts.googleapis.com/css?family=M+PLUS+Rounded+1c:400,700|Montserrat:400,700&amp;subset=cyrillic" rel="
    stylesheet">
    <link rel="stylesheet" type="text/css" href="{% static 'css/home.css' %}">
</head>
<body>
    <header id="header" class="header">
    </header>

    <section id="about" class="about">
    </section>

    <section id="info" class="info">
    </section>

<footer id="footer" class="footer">
    </footer>


</body>
</html>
```

Рисунок 3.1.3 – файл home.html

Добавим View и url для главной страницы:

```
class HomeView(TemplateView):
    template_name = "home.html"
```

Рисунок 3.1.4 – View для главной страницы

```
from mysite.views import HomeView

urlpatterns = [
    url(r'^$', HomeView.as_view(), name="home"),
    url(r'^admin/', admin.site.urls),
] + static(settings.STATIC_URL, document_root=settings.STATICFILES_DIRS)
```

Рисунок 3.1.5 – Url для главной страницы

Создаем «скелет» основных функций сайта

```
{% extends 'base.html' %}
{% load staticfiles my_filters %}
{% block content %}
    <div class="container">
        <div class="row">
            <div class="col-3">
                <div class="block left-menu">
                    <a href="{% url 'profile' %}">
                        <i class="fa fa-user-circle"></i> Мой профиль
                    </a>
                    <a href="{% url 'home' %}">
                        <i class="fa fa-newspaper-o"></i> Новости
                    </a>
                    <a href="{% url 'home' %}">
                        <i class="fa fa-users"></i> Мои друзья
                    </a>
                </div>
            </div>
            <div class="col-6 content">
                <div class="card">
                    <div class="card-body">
                        <form method="post" name="new-post-form" enctype="multipart/form-data">
                            {% csrf_token %}
                            <textarea class="form-control form-control-sm" type="text" name="text"
                                      placeholder="Что нового?"></textarea>
                            <label for="image">Прикрепить картинку:</label>
                            <input class="form-control form-control-sm" type="file" name="image"><br>
                            <input class="form-control btn btn-outline-success btn-sm" type="submit" value="Добавить">
```

Рисунок 3.1.6 – файл timeline.html

3.2 CSS компонент web-сервиса

Главная задача CSS – это корректное визуальное представление страницы перед пользователем согласно заданным стилевым правилам.

CSS работает с большим множеством элементов страницы: шрифты, цвета фона и символов, поля, таблицы, строки, высота и ширина элементов, отступы, изрбражения, их корректным отображением, с позиционированием элементов и многое другое.

Основная цель CSS – это разгрузить файл HTML, поскольку раньше логическая разметка и описание внешнего вида страницы хранилось в одном

файле и значительно затрудняло поддержку сайта и его дальнейшую разработку.

В папке static создаем файлы стилей проекта

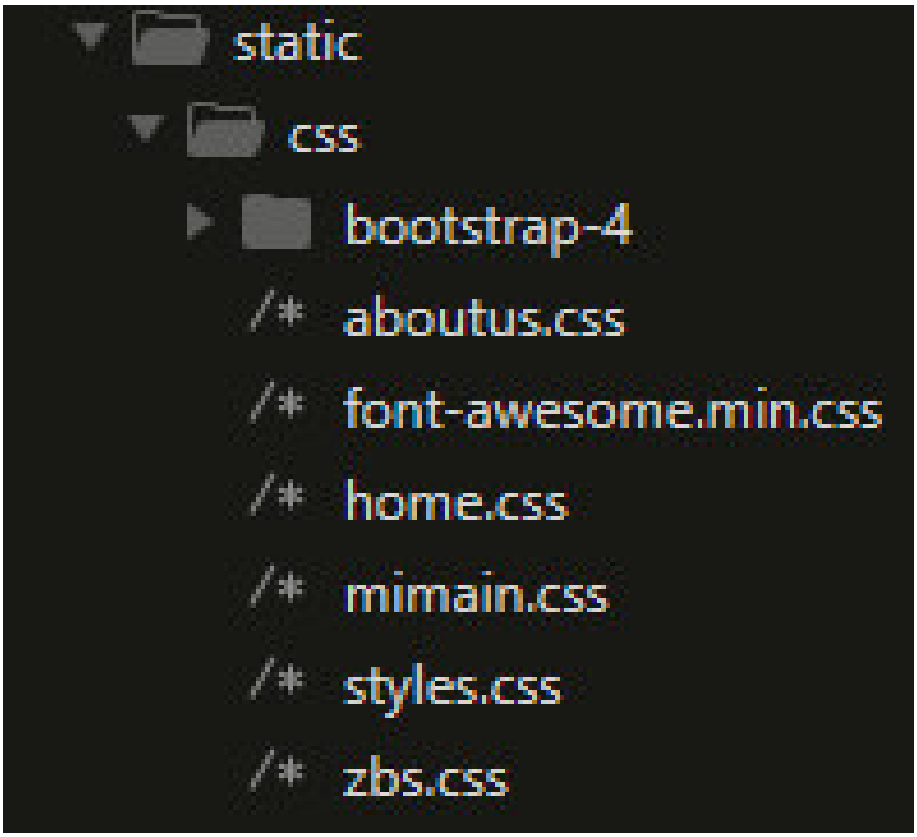

Рисунок 3.2.1 – Стили

Стили лежат в папке css, в файлах styles.css и mimain.css прописан код для основных страниц сайта.

Содержание сайта сродни внутреннему миру человека, дизайн одёжке и лицу. Язык css предназначен для грамотного оформления внешнего вида страниц сайтов, созданных на основе языков html/xhtml или xml. Именно поэтому уроки css жизненно необходимы для каждого начинающего веб-программиста.Раздел содержит множество полезностей и тонкостей использования каскадных таблиц стилей наложение слоёв, выпадающие меню, свойства текста, подсветка ссылок, сборники рецептов и множество других интересных статей.

CSS является неотъемлемой частью современного программирования, так как от визуального содержания страницы зависит его удобство в использовании и популярность среди пользователей.

Чем лучше сделан сайт с визуальной точки зрения, тем он более конкурентоспособен на поисковых платформах.

```css
body {
    font-family: 'Montserrat', sans-serif;
    color:#444B51;
}
ul, li {
    display: block;
    padding: 0;
    margin: 0;
}
h1, h2 {
    font-family: 'M PLUS Rounded 1c', sans-serif;
    color: #fff;

}

.menu {
    margin-top: 15px;
}
.menu__item {
    font-weight: 700;
    font-size: 14px;
}
.menu__item a{
    color:#452467;
    text-transform: uppercase;
}
.auth a{
    color:#fff;
}
.register a{
    color:#fff;
}
.register {
    background:#007bff;
    color:#fff;
    font-size: 14px;
    font-weight: 700;
    border-radius: 4px;
    text-transform: uppercase;
    padding: 15px 0;
    display: block;
    width: 160px;
    border: 0;
}
.register:hover{
    text-decoration: none;
    color: #000;
}
```

Рисунок 3.2.2 – файл home.css

Как видно на рисунке, мы прописали стили для кнопок регистрации, авторизации, также мы выбрали основным шрифтом страницы – шрифт “Montserrat”

```css
body{
    background-color: #efefef;
}

.content{
    border: 1px solid #e2e2e2;
    border-top: none;
    border-radius: 0 0 4px 4px;
    box-shadow: 0 0 5px -4px rgba(128, 128, 128, 0.69);
    background: white;
    min-height: 500px;
    padding-top: 20px;
    padding-bottom: 20px;
}

ul.errorlist{
    color: red;
    list-style: none;
    font-size: 12px;
}

.avatar-img{
    width: 200px;
    height: 200px;
    object-fit: cover;
}
.viti{
    color:#fff;
}

.block{
    border: 1px solid #e2e2e2;
    border-radius: 4px;
    box-shadow: 0 0 5px -4px rgba(128, 128, 128, 0.69);
    background: white;
    padding: 10px;
    margin: 5px;
}

.left-menu a{
    display: block;
    line-height: 30px;
    color: rgb(20, 93, 148);
}

.mainpage-avatar-img{
    width: 100px;
    height: 100px;
```

Рисунок 3.2.3 – файл styles.css

3.3 Создание Базы данных и регистрации пользователей

При регистрации человек будет вводить как обычно email, имя и пароль. После регистрации он должен ввести свои личные данные. Модель User из комплекта Django содержит такие поля как email, first_name, last_name и пароль.

Поэтому чтобы хранить остальные личные данные пользователя, мы создадим другую модель Profile, который будет иметь связь с моделью User один-к-одному.

Опишем модель Profile в файле базы данных models.py:

```python
# coding=utf-8
from django.contrib.auth.models import User
from django.db import models

GENDER_CHOICES = [
    ['male', u"Мужской"],
    ['female', u"Женский"],
]

REL_CHOICES = [
    ['none', u"Не определенно"],
    ['single', u"Холост"],
    ['in_a_rel', u"В отношениях"],
    ['engaged', u"Помолвлен(а)"],
    ['married', u"Женат/Замужем"],
    ['in_love', u"Влюблен(а)"],
    ['complicated', u"Все сложно"],
]


class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, verbose_name=u"Пользователь")
    avatar = models.FileField(verbose_name=u"Аватар", null=True, blank=True)
    bio = models.TextField(max_length=500, blank=True, null=True, verbose_name=u"О себе")
    city = models.CharField(max_length=30, blank=True, null=True, verbose_name=u"Город")
    birth_date = models.DateField(null=True, blank=True, verbose_name=u"Дата рождения")
    gender = models.CharField(max_length=10, verbose_name=u"Пол", choices=GENDER_CHOICES, default="male")
    relationship = models.CharField(max_length=20, verbose_name=u"Статус отношений", choices=REL_CHOICES, default="none")
```

Рисунок 3.3.1 – файл models.py

Для двух полей мы указали какие варианты там могут быть(choices). В choices мы указываем список значений, которые может принимать данное поле. В каждом значении по две части: первое для хранения в БД ('single', 'male', ..), а второе для отображения ("Помолвлена", ,..).

Все, теперь создаем миграции и проводим их. (makemigrations, migrate)

Дальше создаем страницу регистрации, логина и профиля.

Сначала добавим RegisterView во views.py:

```python
class RegisterView(TemplateView):
    template_name = "registration/register.html"

    def dispatch(self, request, *args, **kwargs):
        form = RegisterForm()
        if request.method == 'POST':
            form = RegisterForm(request.POST)
            if form.is_valid():
                self.create_new_user(form)
                messages.success(request, u"Вы успешно зарегистрировались!")
                return redirect("/")

        context = {
            'form': form
        }
        return render(request, self.template_name, context)

    def create_new_user(self, form):
        email = None
        if 'email' in form.cleaned_data:
            email = form.cleaned_data['email']
        User.objects.create_user(form.cleaned_data['username'], email, form.cleaned_data['password'],
                                 first_name=form.cleaned_data['first_name'],
                                 last_name=form.cleaned_data['last_name'])
```

Рисунок 3.3.2 – файл views.py

Класс RegisterView будет обрабатывать страницу регистрации. Тут мы создаем джанго форму RegisterForm. Такие формы сами показывают на страницу html форму с полями ввода и умеют проверять правильность заполнения.

Если форма правильно заполнена, то мы создаем нового пользователя и переносим пользователя на главную страницу.

RegisterForm мы опишем в файле forms.py:

```
# coding=utf-8
from django import forms


class RegisterForm(forms.Form):
    username = forms.CharField(label=u"Имя пользователя")
    first_name = forms.CharField(label=u"Имя")
    last_name = forms.CharField(label=u"Фамилия")
    email = forms.EmailField(label=u"Email", required=False)
    password = forms.CharField(label=u"Пароль", widget=forms.PasswordInput)
    password_confirm = forms.CharField(label=u"Подтвердите пароль", widget=forms.PasswordInput)

    def is_valid(self):
        valid = super(RegisterForm, self).is_valid()
        if self.cleaned_data['password'] != self.cleaned_data['password_confirm']:
            self.add_error("password_confirm", u"Пароли не совпадают")
            return False
        return valid
```

Рисунок 3.3.3 – файл forms.py

Джанго-формы похожи на модели, тут описываются какие поля будут в форме. Метод is_valid() проверяет правильно ли заполнена форма.

Мы дополняем проверку сравнением двух введенных паролей. Если они не похожи, то возвращаем ошибку. Теперь эту форму импортируем в файл views.py.

Добавим классы для страницы профиля и для логаута в файл views.py:

```
class LogoutView(View):
    def dispatch(self, request, *args, **kwargs):
        logout(request)
        return redirect("/")


class ProfileView(TemplateView):
    template_name = "registration/profile.html"
```

Рисунок 3.3.4 – файл views.py

Добавим сразу все нужные url(ы) в urls.py:

```
url(r'^accounts/login/$', login, name="login"),
url(r'^accounts/logout/$', LogoutView.as_view(), name="logout"),
url(r'^accounts/register/$', RegisterView.as_view(), name="register"),
url(r'^accounts/profile/$', ProfileView.as_view(), name="profile"),
```

Рисунок 3.3.5 – файл urls.py

И импортируем View классы и функцию login:

```
from django.contrib.auth.views import login
```

Рисунок 3.3.5 – импортирование функции login

Теперь нам нужно создать три html-шаблона для трех страниц. Создадим их в папке templates/registration. Создадим папку registration, в templates.

```
{% extends 'base.html' %}
{% load staticfiles %}
{% block content %}
    <div class="container content" style="text-align: center;">
        <div style="display: inline-block; width: 300px;">
            <h2><br></h2>
            <form method="post" class="bootstrap-form">
                {% csrf_token %}
                {{ form.as_p }}
                <input type="submit" class="btn btn-success" value="Войти">
            </form>
        </div>
    </div>
{% endblock %}
```

Рисунок 3.3.6 – файл login.html

```
{% extends 'base.html' %}
{% load staticfiles %}
{% block content %}
    <div class="container content" style="text-align: center;">
        <div style="width: 300px; display: inline-block;">
            <h2>Регистрация</h2>
            <form method="post" class="bootstrap-form">
                {% csrf_token %}
                {{ form.as_p }}
                <input type="submit" class="btn btn-success" value="Регистрация">
            </form>
        </div>
    </div>
{% endblock %}
```

Рисунок 3.3.7 – файл register.html

```
{% extends 'base.html' %}
{% load staticfiles %}
{% block content %}

    <div class="container content">
        <h2>{{ selected_user.get_full_name }}</h2>

        <div class="row" style="margin-left: 20px;">
            <div class="col5">
                {% if selected_user.profile.avatar.name %}
                    <img src="{{ selected_user.profile.avatar.url }}" class="avatar-img img-thumbnail">
                {% else %}
                    <img src="{% static 'img/user.jpg' %}" class="avatar-img img-thumbnail">
                {% endif %}
                <br>
                <br>
                {% if selected_user.id == user.id %}
                    <a href="{% url 'edit_profile' %}" class="btn btn-sm btn-info">Редактировать профиль</a>
                {% endif %}
            </div>
            <div class="col">
                <dl class="row">
                    <dt class="col-sm-3 text-right">Полное имя</dt>
                    <dd class="col-sm-9">{{ selected_user.get_full_name }}</dd>

                    <dt class="col-sm-3 text-right">Email</dt>
                    <dd class="col-sm-9">{{ selected_user.email }}</dd>

                    <dt class="col-sm-3 text-right">Город</dt>
                    <dd class="col-sm-9">{{ selected_user.profile.city }}</dd>

                    <dt class="col-sm-3 text-right">Дата рождения</dt>
                    <dd class="col-sm-9">{{ selected_user.profile.birth_date|date:"d M Y" }}</dd>

                    <dt class="col-sm-3 text-right">Пол</dt>
                    <dd class="col-sm-9">{{ selected_user.profile.get_gender_display }}</dd>

                    <dt class="col-sm-3 text-right">Статус отношений</dt>
                    <dd class="col-sm-9">{{ selected_user.profile.get_relationship_display }}</dd>

                    <dt class="col-sm-3 text-right">О себе</dt>
                    <dd class="col-sm-9"><em style="font-family: 'Times New Roman', serif;">{{ selected_user.profile.bio|linebreaks
                    }}</em></dd>
                </dl>
            </div>
        </div>
    </div>
{% endblock %}
```

Рисунок 3.3.8 – файл profile.html

Я добавил в форму class="bootstarp-form". Всем потомкам форм с таким классом мы через jQuery будем добавлять класс "form-control", который делает поле ввода, красивой. Вот этот код в файле scripts.js:

```
function initBootstrapForms() {
    $("form.bootstrap-form").find("input,textarea").addClass("form-control");
    $("form.bootstrap-form").find("input[type='submit']").removeClass("form-control");
}


$(document).ready(function(){
    initBootstrapForms();
});
```

Рисунок 3.3.9 – файл scripts.js

Для показа различных сообщений, я буду использовать пакет messages в комплекте django. Его использование есть в RegisterView, после завершения регистрации. Для отображения сообщения во всех страницах, мы добавим в base.html:

```
<body>
    {% include 'blocks/navbar.html' %}
    {% include 'blocks/messages.html' %}

    {% block content %}
    {% endblock %}
</body>
```

Рисунок 3.3.10 – пакет messages

В папке blocks создадим еще один файл messages.html:

```
{% for message in messages %}
    {% if message.tags == 'success' %}
        <div class="alert alert-success" role="alert">
            {{ message }}
        </div>
    {% elif message.tags == 'error' %}
        <div class="alert alert-danger" role="alert">
            {{ message }}
        </div>
    {% endif %}
{% endfor %}
```

Рисунок 3.3.11 – файл messages.html

В этом разделе я успешно создал Базу данных, регистрацию пользователей и разделы профиля.

3.4 Заполнение личных данных

После регистрации, перенесем пользователя на страницу логина. Когда пользователь впервые успешно авторизуется, мы покажем ему форму заполнения личных данных.

Откроем RegisterView и после успешной регистрации вернем пользователя на страницу логина, а не на главную.

```
return redirect(reverse("login"))
```

Рисунок 3.4.1 – фрагмент кода

После логина, Django перенаправляет пользователя на страницу профиля. Там пока у нас пусто. Поэтому, заполняем страницу профиля так:

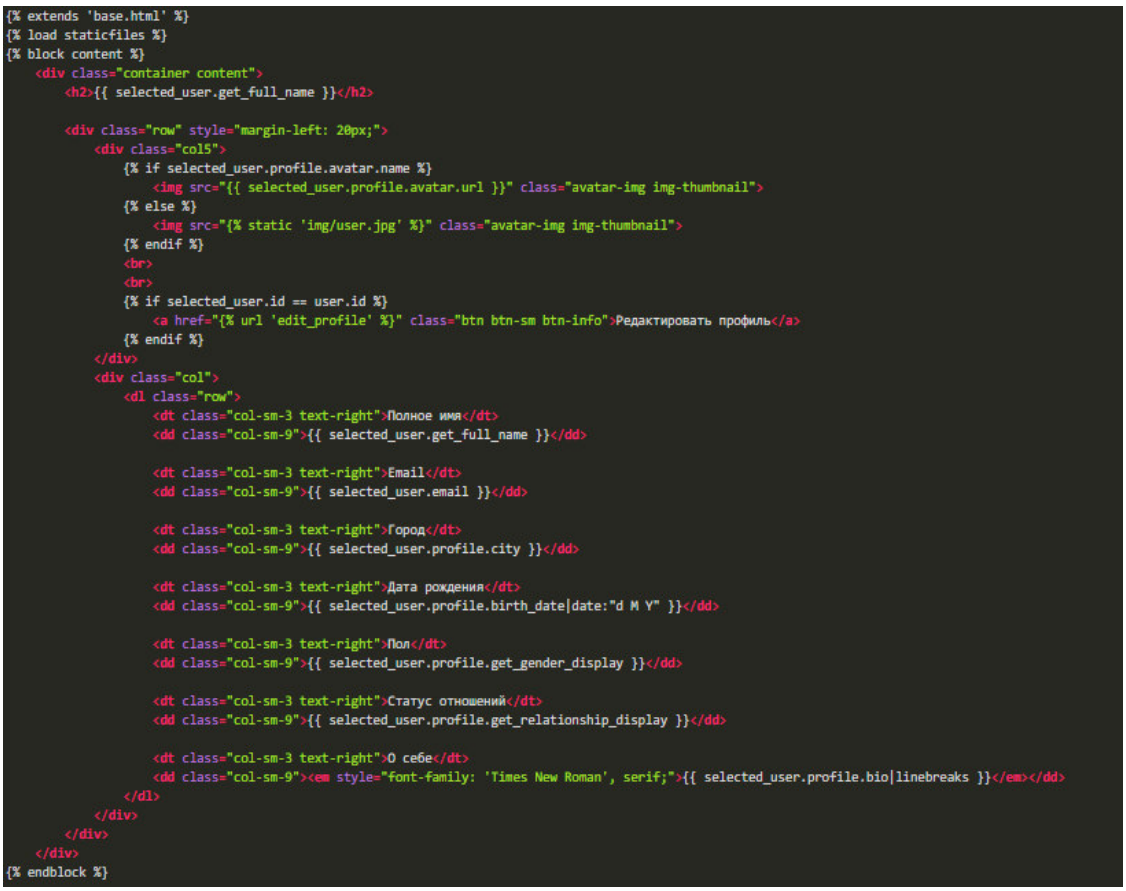

Рисунок 3.4.2 – заполнение файла profile.html

Здесь selected_user - это выбранный пользователь, профиль которого показывается. Этот шаблон в будущем мы будем использовать для просмотра чужих профилей тоже. А user - это текущий пользователь. selected_user.profile - так мы обращаемся к связанному объекту Profile. Как видно из кода, если у пользователя нет аватарки, мы показываем статическую картинку 'img/user.jpg'.

Дальше мы показываем ссылку "Редактировать профиль", если selected_user равно user, т.е. если мы просматриваем свой профиль, то мы видим кнопку редактирования. А внизу страницы информация о пользователе. Теперь, нам нужно поменять ProfileView так, чтобы он передавал в контекст шаблона переменную selected_user. Для этого откроем класс ProfileView и добавим метод dispatch:

```
class ProfileView(TemplateView):
    template_name = "registration/profile.html"

    def dispatch(self, request, *args, **kwargs):
        if not Profile.objects.filter(user=request.user).exists():
            return redirect(reverse("edit_profile"))
        context = {
            'selected_user': request.user
        }
        return render(request, self.template_name, context)
```

Рисунок 3.4.3 – добавление метода dispatch

Проверяем есть ли в БД объект Profile связанный с текущим пользователем, т.е. заполнял ли пользователь свой профиль уже. Если нет, то мы его редиректим(перенаправляем) на страницу редактирования профиля. Этой страницы у нас пока нет. Ниже мы его создадим. Дальше, в создаем переменную context, в котором указываем selected_user как request.user - это текущий пользователь.

Чтобы редактировать профиль, нам нужно создать джанго-форму. Откроем файл forms.py и добавляем туда класс ProfileForm:

```
from mysite.models import Profile

class ProfileForm(forms.ModelForm):

    class Meta:
        model = Profile
        exclude = ['user']
```

Рисунок 3.4.4 – добавление класса ProfileForm

Вначале мы импортировали нашу модель Profile из файла models. Класс ProfileForm наследует класс ModelForm. А ModelForm умеет создавать форму на основе указанной модели, в нашем случае Profile. Мы указали чтобы он не включал поле 'user'. В модели Profile есть поле user, который ссылается на объект Пользователя, и определяет какому пользователя принадлежит этот профиль. Это поле мы будем вручную выставлять равным текущему пользователю.

Форму создали, теперь создадим view для страницы редактирования профиля. Откроем views.py и добавим туда класс EditProfileView:

```
class EditProfileView(TemplateView):
    template_name = "registration/edit_profile.html"

    def dispatch(self, request, *args, **kwargs):
        form = ProfileForm(instance=self.get_profile(request.user))
        if request.method == 'POST':
            form = ProfileForm(request.POST, request.FILES, instance=self.get_profile(request.user))
            if form.is_valid():
                form.instance.user = request.user
                form.save()
                messages.success(request, u"Профиль успешно обновлен!")
                return redirect(reverse("profile"))
        return render(request, self.template_name, {'form': form})

    def get_profile(self, user):
        try:
            return user.profile
        except:
            return None
```

Рисунок 3.4.5 – добавление класса EditProfileView

Сюда нужно импортировать Profile из файла models.py и ProfileForm из файла forms.py.

При обращении пользователя на эту страницу, мы создаем форму ProfileForm и просто отображаем страницу с этой формой. А если метод POST, т.е. пользователь заполнил и отправил форму, мы снова создаем форму ProfileForm уже с отправленными пользователем данными и файлами, проверяем правильно ли заполнена форма, и если правильно, то сохраняем форму. Эта форма при сохранении сохраняет объект Profile в базу данных.

К этому объекту мы можем обратиться через form.instance. И как видно из кода, мы сначала объекту Profile устанавливаем пользователя.

При создании формы, она не связана ни с каким объектом из БД. И при сохранении форма создаст новый объект.

Но, а если мы хотим редактировать просто старый профиль, мы должны указать форме ссылку на эту профиль. После этого он, во-первых, не будет создавать новый объект в БД, а просто менять старый, и во-вторых он заполнит форму имеющимися старыми данными. Метод get_profile возвращает объект Profile если он есть для текущего пользователя, иначе возвращает None.

Теперь, создадим html шаблон для этой страницы. И он будет в файле registartion/edit_profile.html:

```
{% extends 'base.html' %}
{% load staticfiles %}
{% block content %}
    <div class="container content" style="text-align: center;">
        <div style="display: inline-block; width: 400px;">
            <h2>Редактировать профиль</h2>
            <form method="post" class="bootstrap-form" enctype="multipart/form-data">
                {% csrf_token %}
                {{ form.as_p }}
                <input type="submit" class="btn btn-success" value="Сохранить">
            </form>
        </div>
    </div>
{% endblock %}
```

Рисунок 3.4.6 – шаблон редактирования профиля

В теге <form> мы добавили атрибут enctype="multipart/form-data". Это значит что через эту форму можно загружать не только данные, но и файлы. Также нужно добавить url для этой страницы:

```
url(r'^accounts/profile/edit/$', EditProfileView.as_view(), name="edit_profile"),
```

Рисунок 3.4.7 – url файл

При загрузке аватарки пользователя, она сохраняется в папку медиа файлов, путь к которой указан в файле settings.py в переменной MEDIA_ROOT. Обычно она равна:

```
MEDIA_ROOT = os.path.join(BASE_DIR, "media")
```

Рисунок 3.4.8 – переменная MEDIA_ROOT

Т.е. папка media внутри корня нашего проекта. После загрузки аватарки, файл появится в этой папке. А чтобы через бразуер получить эти файлы, мы обращаемся через специальный медиа url:

```
MEDIA_URL = '/media/'
```

Рисунок 3.4.9 – медиа url

Добавляем в urls.py ссылку для медиа:

```
] + static(settings.STATIC_URL, document_root=settings.STATICFILES_DIRS) +\
              static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Рисунок 3.5.10 – фрагмент из urls.py

3.5 Разработка функционала web-сервиса

В предыдущих разделах мы создали основу для нашего сайта, где есть регистрация пользователей, заполнение дополнительных данных о пользователе и страница профиля.

В этом разделе мы добавим ленту событий, возможность добавлять посты, оставлять комментарии и лайкать посты. Начнем с модели Поста.

Пост – это то, что выкладывает пользователь у себя на ленте. Это может быть просто текст, картинка или картинка с текстом. И так, у нашего поста будут поля:

- время создания;
- автор;
- текст;
- картинка.

Откроем models.py и создадим там новую модель Post:

```python
class Post(models.Model):
    datetime = models.DateTimeField(verbose_name=u"Дата", auto_now_add=True)
    author = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name=u"Автор", related_name="posts")
    text = models.CharField(max_length=1000, verbose_name=u"Текст", null=True, blank=True)
    image = models.FileField(verbose_name=u"Картинка", null=True, blank=True)

    class Meta:
        ordering = ["-datetime"]
```

Рисунок 3.5.1 – создание модели Post

У ForeignKey есть свойство on_delete, которое говорит, как нужно реагировать если ссылаемый объект будет удален. Мы указали models.CASCADE - что означает каскадное удаление.

В данном случае, если какой-то пользователь будет удален, то все его посты тоже будут удалены. ordering = ["-datetime"] - указывает что посты будут отсортированы по убыванию даты создания. Дальше создадим миграции и проведем их (makemigrations, migrate).

Теперь, напишем код для нашей ленты. Нам нужно сделать так, чтобы на главной странице отображалась лента пользователя (timeline), если он залогинен.

В ленте будут все посты. Главную страницу обслуживает класс HomeView, поэтому мы в нем добавим метод dispatch. И если пользователь уже залогинен, то будем рендерить шаблон для ленты (timeline.html), иначе шаблон главной страницы (home.html):

```python
class HomeView(TemplateView):
    template_name = "home.html"
    timeline_template_name = "timeline.html"

    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated():
            return render(request, self.template_name)

        context = {
            'posts': Post.objects.all()
        }
        return render(request, self.timeline_template_name, context)
```

Рисунок 3.5.2 – класс HomeView

Редактируем файл timeline.html, у нас получился очень большой шаблон. Который состоит из трех блоков:

– левый;
– правый;
– основной.

Слева у нас будут три кнопки. А справа информация о пользователе. В основном блоке есть форма добавления нового поста, и дальше перечислены все имеющиеся посты.

Пользователи могут добавлять новые посты. Помимо этого, они имеют возможность комментировать как чужие, так и свои посты.

```
{% extends 'base.html' %}
{% load staticfiles %}
{% block content %}
    <div class="container">
        <div class="row">
            <div class="col-3">
                <div class="block left-menu">
                    <a href="{% url 'profile' %}">
                        <i class="fa fa-user-circle"></i> Мой профиль
                    </a>
                    <a href="{% url 'home' %}">
                        <i class="fa fa-newspaper-o"></i> Новости
                    </a>
                    <a href="{% url 'home' %}">
                        <i class="fa fa-users"></i> Мои друзья
                    </a>
                </div>
            </div>
            <div class="col-6 content">
                <div class="card">
                    <div class="card-body">
                        <form method="post" name="new-post-form" enctype="multipart/form-data">
                            {% csrf_token %}
                            <textarea class="form-control form-control-sm" type="text" name="text"
                                      placeholder="Что нового?"></textarea>
                            <label for="image">Прикрепить картинку:</label>
                            <input class="form-control form-control-sm" type="file" name="image"><br>
                            <input class="form-control btn btn-outline-success btn-sm" type="submit" value="Добавить">
                        </form>
                    </div>
                </div>
                <div class="timeline">
                 {% for post in posts %}
                    <div class="card">
                        <div class="card-body">
                            {% if post.image.name %}
                                <img src="{{ post.image.url }}" class="img-thumbnail"><br>
                            {% endif %}
                            {{ post.text }}
                        </div>
                    </div>
                {% endfor %}
            </div>
        </div>
        <div class="col-3">
            <div class="block" style="text-align: center;">
                <b>{{ user.get_full_name }}</b>
                {% if user.profile.avatar.name %}
                    <img src="{{ user.profile.avatar.url }}" class="mainpage-avatar-img img-thumbnail">
                {% else %}
                    <img src="{% static 'img/user.jpg' %}" class="mainpage-avatar-img img-thumbnail">
                {% endif %}
                <div class="right-menu-links">
                    <a href="{% url 'profile' %}" class="btn btn-outline-primary btn-sm">
                        <i class="fa fa-user-circle"></i> Мой профиль
                    </a>
                    <a href="{% url 'edit_profile' %}" class="btn btn-outline-success btn-sm">
                        <i class="fa fa-pencil-square"></i> Редактировать
                    </a>
                </div>
            </div>
        </div>
```

Рисунок 3.5.3 – файл timeline.html

Добавляем несколько стилей в файл styles.css:

```css
.block{
    border: 1px solid #e2e2e2;
    border-radius: 4px;
    box-shadow: 0 0 5px -4px rgba(128, 128, 128, 0.69);
    background: white;
    padding: 10px;
    margin: 5px;
}

.left-menu a{
    display: block;
    line-height: 30px;
    color: rgb(20, 93, 148);
}

.mainpage-avatar-img{
    width: 100px;
    height: 100px;
    object-fit: cover;
}

.right-menu-links{
    margin-top: 20px;
}
```

```css
.right-menu-links a{
    margin-top: 10px;
    margin-bottom: 5px;
}

.card{
    margin-bottom: 20px;
}
```

Рисунок 3.5.4 – файл styles.css

Подключим иконки CSS-библиотеки fontawesome. Установим их в папку static файлов проекта. Затем, найдем файл font-awesome.min.css. и скопируем его в папку css нашего проекта. Далее, этот css-файл нужно подключаем в base.html:

```html
<link href="{% static 'css/font-awesome.min.css' %}" rel="stylesheet">
```

Рисунок 3.5.5 – подключение CSS-библиотеки fontawesome

После я могу добавлять разные иконки в свой проект использую css-класс fa.

```html
<i class="fa fa-pencil"></i>
<i class="fa fa-apple"></i>
<i class="fa fa-user-circle"></i>
```

Рисунок 3.5.6 – пример иконок

В ленте у нас есть форма добавления нового поста. Давайте напишем обработчик для этой формы. Для начала, создадим джанго форму PostForm в файле forms.py:

```
class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        exclude = ['author']
```

Рисунок 3.5.7 – форма PostForm

Импортируем Post из моделей.

Так, как добавления поста происходит на главной странице, я добавлю обработчик в класс HomeView. Теперь метод dispatch выглядит так:

```
def dispatch(self, request, *args, **kwargs):
    if not request.user.is_authenticated():
        return render(request, self.template_name)

    if request.method == 'POST':
        form = PostForm(request.POST, request.FILES)
        if form.is_valid():
            form.instance.author = request.user
            form.save()
            return redirect(reverse("home"))
    context = {
        'posts': Post.objects.all()
    }
    return render(request, self.timeline_template_name, context)
```

Рисунок 3.5.8 – измененный метод dispatch

Используя PostForm мы получаем введенные пользователем данные и после проверки сохраняем форму, присвоив текущего пользователя как автора нового поста.

Теперь, пользователи могут писать посты и видеть их. Комментарии будут связаны с постами. У комментария будет дата, связанный пост, текст и автор. Вот модель:

```
class Comment(models.Model):
    datetime = models.DateTimeField(verbose_name=u"Дата", auto_now_add=True)
    author = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name=u"Автор", related_name="comments")
    post = models.ForeignKey(Post, on_delete=models.CASCADE, verbose_name=u"Пост", related_name="comments")
    text = models.CharField(max_length=1000, verbose_name=u"Текст", null=True, blank=True)

    class Meta:
        ordering = ["datetime"]
```

Рисунок 3.5.9 – модель создания комментарий

После этого создаем миграции (makemigrations, migrate). Перед тем как начать оставлять комментарии. Мы немного поменяем внешний вид поста в ленте. Теперь в файле timeline.html выглядит так:

```
<div class="timeline">
    {% for post in posts %}
        <div class="card">
            <div class="card-body post">
                <div class="post-title">
                    {% if post.author.profile.avatar.name %}
                        <img src="{{ post.author.profile.avatar.url }}" class="post-author-img img-thumbnail">
                    {% else %}
                        <img src="{% static 'img/user.jpg' %}" class="post-author-img img-thumbnail">
                    {% endif %}
                    <div class="post-author">
                        {{ post.author.get_full_name }}
                    </div>
                    <div class="post-datetime">
                        {{ post.datetime|date:"d M Y H:i" }}
                    </div>
                </div>
                {% if post.image.name %}
                    <img src="{{ post.image.url }}" class="img-thumbnail"><br>
                {% endif %}
                <div class="post-text">
                    {{ post.text|default_if_none:""|linebreaks|urlize }}
                </div>
            </div>
            <div class="card-footer">
                <div id="comments-list-post-{{ post.id }}">
                    {% for comment in post.comments.all %}
                        {% place_comment comment %}
                    {% endfor %}
                </div>
                <div class="comment-form">
                    {% if post.author.profile.avatar.name %}
                        <img src="{{ post.author.profile.avatar.url }}" class="post-author-img img-thumbnail">
                    {% else %}
                        <img src="{% static 'img/user.jpg' %}" class="post-author-img img-thumbnail">
                    {% endif %}
                    <div style="display: flex; margin-top: 4px;">
                        <input class="form-control form-control-sm comment-input"
                               placeholder="Оставить комментарий"  data-post-id="{{ post.id }}">
                    </div>
                </div>
            </div>
        </div>
    {% endfor %}
</div>
```

Рисунок 3.5.10 – измененный timeline.html

У нас есть аватарка, имя автора поста, время поста, комментарии и форма отправки комментария. Для вывода всех комментариев мы написали:

```
{% for comment in post.comments.all %}
    {% place_comment comment %}
{% endfor %}
```

Рисунок 3.5.11 – скрипт для вывода всех комментариев

Мы берем все комментарии данного поста, и для каждого в цикле вызываем функцию place_comment. Такие функции в Джанго называются шаблонными тегами и филтрами (template tags).

Наша функция place_comment здесь вставляет html код, для вывода комментария. Эту функцию мы опишем в специальном файле my_filters.py.

В папке вашего приложения mysite создайте новый пакет templatetags (New->Python Package). Пакет это обычная папка, в котором есть пустой файл с названием __init__.py . В этой папке создайте файл my_filters.py. И в этот файл вписываем:

```python
from django.template.loader_tags import register


@register.inclusion_tag("blocks/comment.html")
def place_comment(comment):
    return {'comment': comment}
```

Рисунок 3.5.12 – файл my_filters.py

Эта функция является шаблонным тегом типа inclusion_tag. Она умеет рендерить указанный html-шаблон с параметрами. В данном случае шаблон находится в файле blocks/comment.html:

```
{% load staticfiles %}
<div class="comment">
    <div class="post-title">
        {% if comment.author.profile.avatar.name %}
            <img src="{{ comment.author.profile.avatar.url }}" class="post-author-img img-thumbnail">
        {% else %}
            <img src="{% static 'img/user.jpg' %}" class="post-author-img img-thumbnail">
        {% endif %}
        <div class="comment-body">
            <span class="comment-author">{{ comment.author.get_full_name }}</span> {{ comment.text|default_if_none:""|urlize }}
            <div class="post-datetime">
                {{ comment.datetime|date:"d M Y H:i" }}
            </div>
        </div>
    </div>
</div>
```

Рисунок 3.5.13 – файл comment.html

В этом шаблоне есть маленький блок для отображения комментария. Чтобы функция place_comment сработало мы должны его подключить в шаблон. Для этого в файле timeline.html необходимо добавить сверху под {% extends %}:

```
{% load my_filters %}
```

Рисунок 3.5.14 – подкючение функции place_comment

После этого нужно мы перезагрузим сервер вручную.

Теперь добавим очень много css-классов для кода что мы написали. После этого, наш файл styles.css теперь выглядит следующим видом:

```
.post{
    padding-bottom: 6
}

.post-title{
    margin-bottom: 16
}

.post-author-img{
    width: 40px;
    height: 40px;
    object-fit: cover
    float: left;
    margin-right: 10p
}

.post-author{
    font-weight: bold
    width: auto;
    overflow: hidden;
    height: 20px;
    font-size: 14px;
    display: block;
}

.post-datetime{
    display: block;
    width: auto;
    overflow: hidden;
    height: 20px;
    font-size: 11px;
    color: grey;
}

.post-text{
    color: #414142;
    font-size: 14px;
}

.comment{
    color: #414142;
    font-size: 14px;
}
```

Рисунок 3.5.15 – добавление новых стилей

Отображение комментариев успешно работает. Теперь дадим пользователям возможность добавлять комментарии.

Мы будем добавлять комментарии асинхронно, т.е. не перезагружая страницу с помощью ajax. Будем отправлять ajax запрос на url /post-comment/

в ответ будем получать кусок html-кода, который будем добавлять в конец списка комментариев.

Для начала давайте создаем view для обработки добавления комментария. Открыв views.py и создаем класс PostCommentView:

```
class PostCommentView(View):
    def dispatch(self, request, *args, **kwargs):
        post_id = request.GET.get("post_id")
        comment = request.GET.get("comment")
        if comment and post_id:
            post = Post.objects.get(pk=post_id)
            comment = Comment(text=comment, post=post, author=request.user)
            comment.save()
            return render(request, "blocks/comment.html", {'comment': comment})
        return HttpResponse(status=500, content="")
```

Рисунок 3.5.16 – класс PostCommentView

Теперь, view.py принимает по GET запросу два параметра post_id и comment. Если они правильно переданы, то создаем новый объект Comment и сохраняет. Возвращает срендеренный шаблон comment.html, который мы использовали для отображения комментария. Здесь тоже возвращаем этот блок с новым комментарием.

А если параметры заполнены неправильно, то возвращает HTTP ответ с кодом 500, это код ошибки.

Теперь нужно в urls.py добавить url для этого view:

```
url(r'^post-comment/$', PostCommentView.as_view()),
```

Рисунок 3.5.17 – обновленный urls.py

Теперь добавим обработку отправки сообщения на фронтенде. Для этого открываем scripts.js и добавим функцию initCommenting().

В этой функции мы берем элементы с классом. comment-input (поля ввода комментария) и вешаем обработчик на событие keyup (клавише отпущена). Если отпущенная клавиша была Enter (13), то берем введенный текст комментария, получаем id поста и отправляем запрос на url /post-comment/. Если запрос выполнене успешно(success), то мы в ответ получаем кусок html с комментарием.

Этот кусок добавляем в конец блока с комментариями данного поста и очищаем поле ввода.

Если вы внимательно посмотрите код timeline.html в поле ввода комментарий мы указываем data-post-id={{ post.id }}.

Этот атрибут будет указывать какому посту будет принадлежать введенный комментарий.

Так как на странице будет много постов, без этого атрибута мы не можем узнать какому же посту был добавлен комментарий.

Значение атрибута data-post-id мы можем получить с помощью jQuery $(this).data("post-id") что и делаем выше в js коде. вместо this может быть любоей html элемент у которого есть такой атрибут.

Scripts.js теперь выглядит следующим образом:

```
function initBootstrapForms() {
    $("form.bootstrap-form").find("input,textarea").addClass("form-control");
    $("form.bootstrap-form").find("input[type='submit']").removeClass("form-control");
}

function initCommenting(){
    $(".comment-input").on("keyup", function(event){
        var input = $(this);
        if(event.keyCode === 13) {
            var comment = $(this).val().trim();
            var post_id = $(this).data("post-id");
            if (comment.length > 0) {
                $.ajax("/post-comment/", {
                    data: {
                        post_id: post_id,
                        comment: comment
                    },
                    success: function(html){
                        $("#comments-list-post-"+post_id).append(html);
                        $(input).val("");
                    }
                })
            }
            return false;
        }
    })
}

$(document).ready(function(){
    initBootstrapForms();
    initCommenting();
});
```

Рисунок 3.5.18 - функция initCommenting()

4 Технико-экономическое обоснование

Темой дипломного проекта является «Разработка web-сервиса «Сеть гостеприимства».

Цель данного дипломного проекта заключается в разработке web-сервиса, пользователи которого смогут обмениваться информацией, сообщениями в туристических целях.

4.1 Определение сложности разработки ПО

Для определения трудоемкости разработки программного обеспечения необходимо последовательно определить задачи и время на их реализацию.

Таблица 1 – Этапы разработки ПО

Этапы разработки ПО	Вид работы	Трудоемкость, чел. час.
Этап 1	Определение задачи	20
Этап 2	Составление технического задания	20
Этап 3	Изучить функционально схожее Программное обеспечение	30
Этап 4	Разработка и верстка адаптивного дизайна web-сервиса	20
Этап 5	Разработка фронтенд части	60
Этап 6	Разработка бэкенд части и создание БД	60
Этап 7	Тестирование проекта	20
Этап 9	Исправление ошибок и недочетов	40
Итого: время на выполнение дипломного проекта		270

Продолжительность рабочего дня равна 8 часам. Поэтому, учитывая то, что итоговое время на выполнение дипломного проекта составляет 270 часов, то для реализации программного обеспечения потребуется 34 рабочих дня(270/8).

4.2 Расчет затрат на разработку ПО

Смета по программному обеспечению включает в себя следующие затраты:

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

Рассчитаем материальные затраты. Они делятся на основные и вспомогательные.

Таблица 2 – Затраты на материальные ресурсы

Наименование материала	Марка	Ед. измерения	Количество	Цена за ед. в тенге	Сумма в тенге
Бумага для офиса	International Paper	Упаковка	1	900	900
Тетрадь (96 листов)	Senner	Штук	1	200	200
Ручка	Mark Tres	Штук	2	150	300
Компьютерная мышь	BLOODY V8m	Штук	1	6000	6000
Итого:					7400

Общая сумма материальных затрат ($З_м$) рассчитывается по формуле:

$$З_м = \sum Р_i * Ц_i , \quad (4.1)$$

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

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

i - вид материального ресурса;

n - количество видов материальных ресурсов.

Расчет затрат на необходимое оборудование и программное обеспечение производится по форме, приведенной в таблице 3.

Таблица 3 – Расчет затрат на оборудование и ПО, необходимое для проекта

Наименование материала	Марка	Ед. измерения	Количество	Цена за ед. в тенге	Сумма в тенге
Ноутбук	HP Pavillion	Штук	1	450000	450000
Принтер	Panasonic	Штук	1	31540	31540
ОС	Windows 8.1	Штук	1	52394	52394
Итого:					533934

$$Зм = 7400 + 533934 = 541334 \text{ (тг)}$$

Для реализации программного обеспечения необходимы материалы на сумму 541 334 тенге.

4.3 Расчет затрат на электроэнергию

Так как при разработке программного обеспечения нам потребуется ноутбук и принтер, то необходимо рассчитать необходимые затраты на электроэнергию.

Согласно таблице 1 для разработки программного продукта необходимо порядка 270 часов, теперь необходимо рассчитать стоимость электроэнергии, которая будет потрачена в течении 270 часов.

Расчет необходимой электроэнергии определяется по формуле:

$$З_{эл.\ эн.\ обор.} = \sum W * К_{исц} * S * T, \qquad (4.3)$$

где W – потребляемая мощность, Вт;

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

T – время работы;

S – тариф (1кВт/ч = 18,32 тг).

Итоги по расчетам стоимости затрачиваемой электроэнергии представлены в таблице 4.

Таблица 4 – Затраты на электроэнерги

Наименование приборов	Паспортная мощность, кВт	Коэффициент мощности	Время работы оборудования, ч	Цена ЭЭ тг/кВтч	Сумма, тг.
Ноутбук	0,7	0,7	270	18,32	2423,74
Принтер	0,4	0,9	10	18,32	65,95
Итого:					2489,69

$$З_{э\ ноутбук} = 0,7 \times 0,7 \times 270 \times 18,32 = 2423,74 \text{ тенге}$$

$$З_{э\ принтер} = 0,4 \times 0,9 \times 10 \times 18.32 = 65,95 \text{ тенге}$$

$$З_{э\ общие} = 2423,74 + 65,95 = 2489,69 \text{ тенге}$$

4.4 Расчет затрат на оплату труда

Для разработки программного обеспечения, как указывалось ранее, необходимо два работника:

– руководитель проекта – управление и корректировка рабочих процессов и времени разработчика;

– разработчик – разработка и тестирование программного обеспечения.

Сумму расходов на оплату труда можно рассчитать по следующей формуле:

$$З_{тр} = \sum ЧС_i * Т_i \quad (4.5)$$

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

$Т_i$ - трудоемкость разработки модели, чел.×ч;

i - категория работника;

n - количество работников, занятых разработкой ПП.

Во время реализации проекта рабочее время участников не равномерно, поэтому имеет смысл установить часовую ставку каждого работника и общий объем заработной платы.

Часовую ставку сотрудника можно рассчитать по следующей формуле:

$$ЧС_i = \frac{ЗП_i}{ФРВ_i} \quad (4.6)$$

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

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

Месячная заработная плата руководителя равняется 210 000 тенге и месячная заработная плата разработчика равняется 205 000 тенге. Рассчитаем часовую ставку каждого работника согласно формуле (4.6):

$$ЧС_{руководитель} = \frac{210\ 000}{21*8} = 1250,00 тг/ч$$

$$ЧС_{разработчик} = \frac{205000}{21*8} = 1220,24 тг/ч$$

Часовая ставка руководителя составляет 1250,00 (тг/ч), трудоемкость разработки равняется 120 часам. Часовая ставка разработчика составляет

1220,24 (тг/ч), трудоемкость разработки равняется 270 часам. Согласно формуле (4.5) можно рассчитать сумму расходов на заработную плату работников:

$$З_{тр}=1250,00*120+1220,24*270=479467,50$$

Расчеты затрат по оплате труда показаны в таблице 5.

Таблица 5. – Расчет заработной платы

Категория работника	Квалификация	Трудоемкость разработки ПП, час.	Часовая ставка, тг/ч	Сумма, тг.
Руководитель	Инженер	120	1250,00	150000,00
Разработчик	Программист	270	1220,25	329467,50
Итого:				479467,50

4.5 Расчет затрат по социальному налогу

Согласно Налоговому кодексу Республики Казахстан социальный налог составляет 9,5% от фонда оплаты труда и мед страховки 1.5%. Итого 11% социальный налога. Социальный налог можно рассчитать по следующей формуле:

$$З_{по} = З_{тр} \times 0,1 = 479467,50 \times 0,1 = 47946,75$$

$$З_{тр}(\text{с уч п. о. }) = З_{тр} - З_{по} = 479467,50 - 47946,75 = 431520,75$$

$$Н_{с} = З_{тр}(\text{с уч п. о. }) \times 0,11 = 431520,75 \times 0,11 = 47467,28$$

4.6 Амортизация основных фондов и прочие затраты

Годовые нормы амортизации ОФ принимаются по налоговому кодексу РК или определяются, исходя из возможного срока полезного использования ОФ:

$$Н_{Ai}=\frac{100}{Т_{Ni}}$$

$$Н_{Ai}=\frac{100}{5}=20\%$$

, (4.7)

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

Возможный срок полезного использования ОФ примем: 5 лет для ноутбука и 7 лет для принтера. Эффективный фонда рабочего времени для компьютера и

$$З_{АМ1}=\frac{450000\times20\times270}{100\times1968}=12347{,}56 \text{ тенге}$$

$$З_{АМ2}=\frac{31540\times20\times10}{100\times1968}=32{,}05 \text{ тенге}$$

$$З_{АМобщие}= 12379{,}61 \text{ тенге}$$

Таблица 6 – 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	450000	20	1968	270	12347,56
Принтер	31540	20	1968	10	32,05
ИТОГО					12379,61

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

Таблица 7 – Смета затрат на разработку ПО

Статьи затрат	Сумма, тг
Затраты на материальные ресурсы	541334,00
Затраты на оплату труда	479467,50
Социальные налоги	47467,28
Затраты на электроэнергию	2489,69
Амортизация основных фондов	12379,61
Итого по смете:	1083138,08

Теперь, создаем диаграму всех затрат на наш проект.

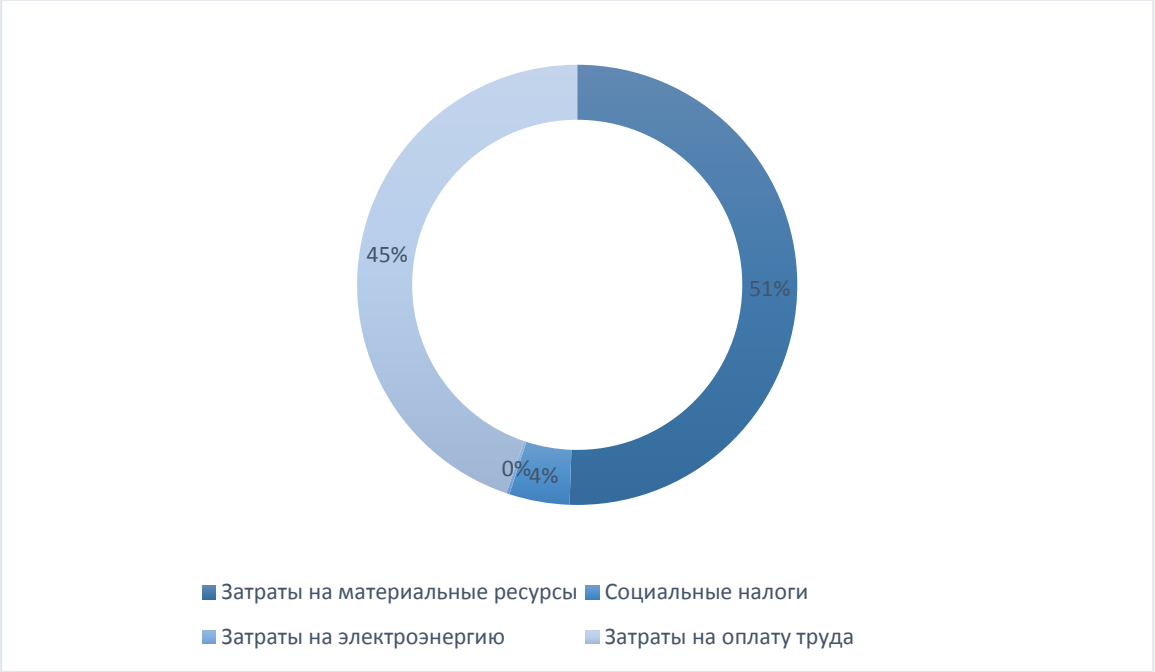

Диаграмма 4.6.1 – Затраты проекта

4.7 Определение возможной (договорной) цены ПО

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

$$Ц_Д = З_{нир}\left(1+\frac{Р}{100}\right), \quad (4.9)$$

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

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

$$Ц_Д = 1083138{,}08 + 1083138{,}08 * 0{,}25 = 1353922{,}60 \text{ тенге}$$

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

$$Ц_р = Ц_д + Ц_д * НДС, \quad (4.10)$$

$$Ц_р = 1353922{,}60 + 1353922{,}60 * 0{,}12 = 1516393{,}31 \text{ тенге}$$

Данную цену можно округлить до 1516400,00 тенге.

Таким образом, себестоимость программного продукта составляет – 1083138,08;

Прибыль составляет – 270784,52;

Договорная цена – 1516400,00.

5. Охрана труда и безопасность жизнедеятельности

В данном разделе рассматриваются мероприятия по обеспечению безопасности жизнедеятельности и охране труда в рамках использования разрабатываемого программного продукта. Сервис будет разрабатываться в домашних условиях, в связи с чем необходимо учитывать вопросы создания оптимальных условий разработки, хорошее самочувствие, безопасность и сохранение здоровья. Существует два основных вредных фактора, воздействующих на разработчика:

– неудовлетворительный микроклимат помещений;

– психоэмоциональное напряжение.

Без строгого учёта правил техники безопасности и производственной санитарии, неточного выполнения требований техники безопасности может привести к аварии, либо к профессиональным заболеваниям и производственному травматизму. Охрана труда обеспечивается системой законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно–профилактических мероприятий и средств, направленных на создание таких условий труда, при которых исключено воздействие на работающих опасных и вредных производственных факторов. Создание наиболее благоприятных, комфортных условий труда, улучшение охраны труда и техники безопасности, без сомнения, ведет к более высокой производительности труда, социальному развитию и повышению благосостояния.

С целью создания нормальных условий для разработчика установлены нормы производственного микроклимата.

5.1 Расчет аспирационной системы для производственного помещения

Таблица 5.1 – Исходные данные

Город	Алматы
Параметры помещения (Д*Ш*В), м	4 * 3 * 2.5
Данные по оборудованию	
Количество компьютеров, шт.	4
Мощность $P_{об}$, кВт/ч	0.5
КПД η	0.95
Данные по источнику света	
Мощность $N_{осв}$, Вт/м^2	60
Вид источника освещения	люминесцентные лампы
Количество сотрудников	1 мужчины
Окна	
Количество, шт.	1
Площадь 1 окна, м^2	0.5

Продолжение таблицы 5.1

Расположение	СВ
Вид	остекление в один-х метал. переплет
Расчетное время суток, ч.	12-13
Температура в помещении	
Летом, °С	23
Зимой, °С	21
Вид положения работы	Сидячее

5.1.1 Расчет тепловых нагрузок в помещении

В помещениях различного назначения действуют в основном тепловые нагрузки, возникающие снаружи помещения (наружные); тепловые нагрузки, возникающие внутри здания (внутренние).

5.1.2 Расчет тепловых нагрузок снаружи помещения

Наружные тепловые нагрузки подставлены следящими 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 является отрицательной и направления потока тепла меняется;

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

– теплопоступления от инфильтрации.

Следует отметить, что наружные тепловые нагрузки могут обладать различными свойствами, т.е. могут быть положительными в зависимости от времени года и времени суток.

В зависимости от времени года и времени суток наружные тепловые нагрузки могут быть положительными.

Теплопоступления и теплопотери в результате разности температур определяется по формуле (5.1):

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

где $V_{пом}$ – объем помещения (5.2), м^3

$$V_{пом}=4*3*2.5=30 \text{ м}^3; \quad (5.2)$$

где X_o – удельная тепловая характеристика, Вт/м^3 °С, X_o = 0.42;

$t_{Нрасч}$ – наружная температура. Для холодного периода - средняя температура самого холодного месяца в 13 часов, для теплого периода - средняя температура 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 (5.3):

$t_{Нрасч}$ = 29,4 °С

$t_{Врасч}$ = 26 °С

$$Q_{огр}=30*0.42*3,4=548 \text{ Вт} \quad (5.3)$$

Для холодного времени года (5.4):

$t_{Нрасч}$ = -9 °С

$t_{Врасч}$ = 19 °С

$$Q_{огр}=30*0.42*|-30|=378 \text{ Вт} \quad (5.4)$$

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 (5.5):

$$Q_р=\left(q^I F_O^I+q^{II} F_O^{II}\right)*\beta_{с.з} \quad (5.5)$$

$$Q_P=(q^I F_O^I+q^{II}F_O^{II})*\beta_{c.з} \quad (5.5)$$

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

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

$\beta_{c.з}$ – коэффициент теплопропускания.

$\beta_{c.з}$=0.15;

При отсутствии наружных затеняющих козырьков, ребер и т.д. для периода облучения остекления солнцем, когда его лучи проникают через окно в помещение $F_O^I=F_O$; $F_O^{II}=0$, (5.6):

$$Q_P=q^I F_O*\beta_{c.з}=\left(q_{вп}+q_{вр}\right)*K_1^C*K_2*n*S_O \quad (5.6)$$

где $q_{вп}$, $q_{вр}$- тепловые потоки от прямой рассеянной радиации, Вт/м^2.

Для широты в 44°СШ до полудня в 11-12 ч. При расположении З: $q_{вп}$, $q_{вр}$

$q_{вп}$=73 Вт/м^2

$q_{вр}$=77 Вт/м^2

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

$$F_O=n*S_o=2*0.5=1 \text{ м}^2 \quad (5.7)$$

где K_1 – коэффициент затемнения остекления переплетами (K_1^C – для облученных проемов).

K_1^C=0.72

K_2 – коэфициент загрязнения остекления.

K_2=0.9

Тогда (5.8):

$$Q_P=(73+77)*0.72*0.9*0.15*1=14{,}6 \text{ Вт} \quad (5.8)$$

Поступление солнечного излучения равно (5.9):

$$Q_P=290 \text{ Вт} \quad (5.9)$$

5.1.3 Расчет тепловых нагрузок внутри помещения

Внутренние тепловые нагрузки в жилых, офисных или относящийся к сфере обслуживание помещения слагаются в основном из тепла:

– выделяемого людьми;

– выделяемого лампами и осветительными, электробытовыми приборами;

– выделяемого компьютерами, печатающими устройствами и фотокопировальными машинами и пр. (в офисных и других помещениях).

В производственных и технологических помещениях различного назначения дополнительными источниками тепловыделение могут быть: нагретое производственное оборудование; горячие материалы, в том числе жидкости 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 путём конвекции и лучеиспусканий, и скрытого тепла, затрачиваемого на испарение влаги с поверхности кожи и из легких.

Летом при 24 один мужчина выделяет явного тепла 61 Вт, а общего – 102 Вт (при работе стоя, легком движении). Женщина выделяет 85% ль нормы тепловыделений взрослого мужчины. Выделение явного тепла составит (5.10):

$$Q_л^я=61*2+61*2*0,85=225,7 \text{ Вт} \quad (5.10)$$

Выделение общего тепла (5.11):

$$Q_л^о=102*2+102*2*0,85=377,4 \text{ Вт} \quad (5.11)$$

Зимой при 20 °С один мужчина выделяет явного тепла 82 Вт, а общего – 103 Вт. Женщина выделяет 85% нормы тепловыделений взрослого мужчины. Тогда выделение явного тепла в помещении составит (5.12):

$$Q_л^я=82*2+82*2*0,85=303,4 \text{ Вт} \quad (5.12)$$

Выделение общего тепла (5.13):

$$Q_л^о=103*2+103*2*0,85=381,1 \text{ Вт} \quad (5.13)$$

Теплопоступление от осветительных приборов, оргтехники и оборудования рассчитывается следующим образом. Теплопоступление от ламп определяется по формуле (5.15):

$$Q_{осв}= \eta*N_{осв}*F_{пол} \quad (5.15)$$

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

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

$F_{пол}$ – площадь пола ($F_{пол}$=12*8=96 м^2);

Тогда (5.16):

$$Q_{осв}=0.5*60*96=2880 \text{ Вт} \quad (5.16)$$

Тепло, выделяемое производственным оборудованием, определяется по формуле (5.17):

$$Q_{об}=N_{уст}*K \quad (5.17)$$

где $N_{уст}$ – паспортная мощность оборудования ($N_{уст}$=1.8 кВт/ч);

K - единиц оборудования (K=4);

Тогда (5.18):

$$Q_{об}=1.8*4*0.95=10.3 \text{ кВт} \quad (5.18)$$

Теплопритоки, возникающие за счет находящейся оргтехники – это 30% мощности оборудования (5.19):

$$Q_{об}=1.8*4*0.3=3.24 \text{ кВт} \quad (5.19)$$

5.2 Расчет теплового баланса помещения

На основании выполненных расчетов поставим баланс теплопоступлений в помещении:

Летом (5.20):

$$Q_{изб}=191,66+56,95+662,4+490+195-0=1596,01 \text{ Дж} \quad (5.20)$$

Зимой (5.21):

$$Q_{изб}=191,66+56,95+662,4+490+195+378=2014,81 \text{ Дж} \quad (5.21)$$

Т.к. тепловой баланс для зимы больше летнего теплового баланса, то рассчитаем тепло-напряжённость воздуха по формуле (5.22, 5.23):

$$Q_Н=\frac{Q_{изб}*860}{V_{пом}} \quad (5.22)$$

$$Q_Н=\frac{2014,81*860}{128}=13,53 \text{ ккал/м}^3 \quad (5.23)$$

При $Q_Н$>20 ккал/м^3, Δt=8 °C

Определение количества воздуха, необходимое для поступления в помещение (5.24) и расчет (5.25):

$$L=\frac{Q_{изб}*860}{C*\Delta t*\gamma} \quad (5.24)$$

где C=0.24 ккал/(кг°C) – теплоемкость воздуха,
γ=1.206кг/м^3 – удельная масса приточного воздуха.

$$L=\frac{2014,81*860}{0.24*10^4*8*1.206}=748.314\,\frac{м^3}{час} \quad (5.25)$$

Определение кратности воздухообмена (5.26, 5.27):

$$n=\frac{L}{V_{пом}} \quad (5.26)$$

$$n=\frac{789,22}{384}=5,846\ час^{-1} \quad (5.27)$$

5.3 Выбор кондиционера

Исходя из найденных значений, подбираем соответствующую модель кондиционера. Приведем основные характеристики выбранного кондиционера:

- режим охлаждения – 2400Вт;
- режим обогрева – 2400Вт;
- поток – 404 м3/ч;
- уровень шума 31Дб;
- габариты – 754х272х176 мм;
- вес – 39 кг;
- рассчитан на помещение площадью до 21 м^2.

Приведем схему расположения кондиционера в помещении и схему подачи воздуха.

Кондиционер состоит из двух блоков. 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;

5.5 Вывод по части безопасности жизнедеятельности

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 блсков 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. Так же 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.

Заключение

В ходе выполнения дипломного проекта была разработана гостевая сеть, названная «Nomad Trip».

При выполнении дипломного проекта были выполнены следующие задачи:

– был проведен анализ существующих web-сервисов подобного направлением;

– определены основные требований к системе учета и регистрации;

– спроектирована базы данных, программная часть и аппаратная часть;

– проведен анализ существующих технологий для решения поставленных задач;

– проведен анализ и выбор компонентов для сборки разработанного проекта;

– разработка системы контроля и учета.

А также, было произведено экономическое обоснование целесообразности разрабатываемого продукта, где были сделано следующее заключение:

– цена реализации окупает все затраты, потрачены на разработку. Прибыль от реализации проекта равна 270 784 тенге;

Помимо этого, были исследованы условия труда в помещении, где велась разработка, и были предложены мероприятия по улучшению качества воздуха в помещении.

Список литературы

1 David Beazley (Author), Brian K. Jones. / – «Python Cookbook, Third edition» / – 2013.

2 Дронов В.А./ – «Python 3 и PyQt 5. Разработка приложений» / – 2017.

3 М. Лутц. / – «Программирование на Python. Т. 2» / – М.: Символ, 2016. – 992 с.

4 М. Лутц. / – «Программирование на Python. Т. 1» / – М.: Символ, 2016. – 992 с.

5 М. МакГрат. / – «Программирование на Python для начинающих» / – М.: Эксмо, 2015. – 192 с.

6 Э. Мэтиз. /– «Изучаем PYTHON.Программирование игр, визуализация данных, веб-приложения» / – СПб. Питер, 2017. - 496 с.

7 М. Саммерфилд. / – «Программирование на Python 3. Подробное руководство» / – М.: Символ, 2016. - 608 с.

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

9 Доусон М. / – «Программируем на Python» / – СПб. Питер, 2014. – 416 с.

10 Chris Fidao. / – «Implementing Laravel. Real-world implementation of testable and maintainable code» / – 2014. – 105с.

11 Терри Фельке-Моррис. / – «Большая книга веб-дизайна» / – 2017.

12 «Веб-разработка – с чего начать?» / – URL: https://habr.com/ru/post/357720/

13 Wikipedia: Сеть гостеприимства / URL: https://ru.wikipedia.org/wiki/Сеть_гостеприимства

14 Wikipedia: CouchSurfing / URL: https://ru.wikipedia.org/wiki/CouchSurfing

15 Wikipedia: Hospitality Club / URL: https://ru.wikipedia.org/wiki/Hospitality_Club

16 Wikipedia: Django / URl: https://ru.wikipedia.org/wiki/Django

17 Веб-фреймворк Django (Python) / URL: https://developer.mozilla.org/ru/docs/Learn/Server-side/Django

18 А. Головатый. / – «Django. Подробное руководство» / – М.: Символ-плюс, 2014. - 512 с.

19 Дж. Форсье. /– «Django. Разработка веб-приложений на Python» / – М.: Символ-плюс, 2014. - 343 с.

20 Python (статья Википедии) / URL: https://ru.wikipedia.org/wiki/Python (дата обращения: 12.02.2019)

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

Приложение А

(обязательное)

Техническое задание

Наименование программного продукта (ПП): Клуб гостеприимства «NomadTrip».
Цель разработки ПО: создание web-приложение со структурированной базой данных по данной тематике, представляющее из себя сервис, который помогает организовывать путешествия.
Идеология программного обеспечения: идеологией программного обеспечения является создание небольшой социальной сети, позволяющей пользователям организовывать путешествия в различные точки мира.
Используемая технология создания ПО: среда Sublime Text 3, Python 3.6.7, Django.
Выбор модели ПО: структурная модель – позволяет оформлять отдельные блоки программы (повторяющиеся и неповторяющиеся) в процедуры и функции, которые затем могут использоваться в других частях программы. Изменения в коде функций и процедур не влекут за собой изменение других частей кода программы.
Описание: модуль авторизация – главный модуль, с помощью которого непосредственно будут совершаться различные функции. Включение этого модуля происходит после регистрации пользователя. Модуль регистрация пользователя – модуль, в котором пользователь заполняет свои личные данные. После выбора данные пользователя будут занесены в базу. Модуль заполнения профиля активируется сразу после регистрации этого пользователя, этот модуль сразу после регистрации открывает страницу профиля пользователя для его дальнейшей регистрации. База пользователей и их результатов – база, содержащая данные обо всех пользователях.
Выбор архитектуры построения ПП: Модульные виды.
Выбор метода разработки структуры ПП: нисходящий метод.
Выбор языка программирования: Python, Javascript.
Предполагаемая аудитория (тип, возраст и т.д.): люди совершеннолетнего возраста
Общий объем ПП, Мб: 40 Мб.

Приложение Б

(обязательное)

Листинг программы

home.html

```
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Nomad Trip</title>
        <link rel="shortcut icon" href="{% static 'img/icon.png' %}" type="image/png">
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
<link
href="https://fonts.googleapis.com/css?family=M+PLUS+Rounded+1c:400,700|Montserrat:400,7
00&amp;subset=cyrillic" rel="stylesheet">
<link rel="stylesheet" type="text/css" href="{% static 'css/home.css' %}">
</head>
<body>
<header id="header" class="header">
        <div class="main-header">
                <div class="container">
                <div class="row">
                        <div class="col-lg-4">
                                <img src="{% static 'img/home.png' %}">
                        </div>
                        <div class="col-lg-4">
                                <nav>
                                        <ul class="menu d-flex justify-content-between">
                                                <li class="menu__item">
                                                        <a href="/">
                                                                на главную
                                                        </a>
                                                </li>

                                                <li class="menu__item">
                                                        <a href="#">
                                                                о нас
                                                        </a>
                                                </li>
                                                <li class="menu__item">
                                                        <a href="#">
                                                                как это работает?
                                                        </a>
```

```
                                        </li>
                                    </ul>
                                </nav>
                            </div>

        <div class="col-lg-3 d-flex justify-content-between" >
                                <button class="auth">
                                    <a href="{% url 'login' %}">вход</a>
                                </button>
                                <button class="register">
                                    <a href="{% url 'register' %}">регистрация</a>
                                </button>
                            </div>
                        </div>
                        </div>
                </div>
                <div class="explore">
                    <div class="container">
                    <div class="row">
                        <div class="col-lg-12">
                            <div class="title">
                                <h1 class="title__heading">
                                    Путешествуй, Знакомься
                                </h1>
                                <h2 class="title__down">
                                    <span class = "h2">Открой для себя страну
заново</span>
                                </h2>
                            </div>
                        </div>
                    </div>
                    </div>

                </div>
        </header>

                <section id="about" class="about">
                <div class="container">
                    <div class="row">
                        <div class="col-lg-6">
                            <div class="history">
                                <h2 class="history__title">
                                    О нас
                                </h2>
                                <span class="history__text">
                                <p>NomadTrip - это Казахстанский клуб
гостеприимства.
                                Наша цель - соединять людей, хозяев и гостей,
путешественников и местных жителей.</p>
```

```
                                        <p>Резиденты Клуба по всему миру помогают друг другу путешествовать - дают им крышу над головой, показывают город - да что угодно! </p>
                                        <p>Вступить в Клуб может любой желающий. Резиденты Клуба могут получать информацию о других резидентах Клуба онлайн, как только они зарегистрируются.</p>
                                        <p>В Клубе работают добровольцы, уверенные в одном: давая путешественникам шанс встречаться с людьми во всех уголках мира и предоставляя местным жителям возможность встречать представителей разных культур, мы можем сделать этот мир лучше.</p>
                                        </span>
                                        <button class="readmore">
                                                <a href="#">
                                                        Подробнее
                                                </a>
                                        </button>
                                </div>
                        </div>
                        <div class="col-lg-3">
                                <div class="nature-images nature-images-down">
                                        <img src="{% static 'img/nature1.png' %}" alt="bao" class="nature-images__item">
                                        <img src="{% static 'img/nature2.png' %}" alt="charyn" class="nature-images__item">
                                </div>
                        </div>
                        <div class="col-lg-3">
                                <div class="nature-images nature-images-up">
                                        <img src="{% static 'img/nature3.png' %}" alt="burabay" class="nature-images__item">
                                        <img src="{% static 'img/nature4.png' %}" alt="astana" class="nature-images__item">
                                </div>
                        </div>
                </div>
        </div>
</section>

<section id="info" class="info">
        <div class="container">

        </div>
</section>

<footer id="footer" class="footer">
        <div class="container">
                <div class="row">
                        <div class="col-lg-12">
                                <ul class="socials d-flex justify-content-center">
                                        <li class="socials__item socials__item_fb">
```

```
                                        <a href="#"></a>

                                    </li>
                                <li class="socials__item socials__item_tw">
                                        <a href="#"></a>
                                </li>
                                <li class="socials__item socials__item_g">
                                        <a href="#"></a>
                                </li>
                            </ul>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-lg-4">
                            <div class="line"></div>
                        </div>
                        <div class="col-lg-4">
                            <div class="credits">
                                2019 ©  NomadTrip.kz
                            </div>
                        </div>
                        <div class="col-lg-4">
                            <div class="line"></div>
                        </div>
                    </div>
            </div>
</footer>
```

base.html

```
</body>
</html>
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Nomad Trip</title>
   <link rel="shortcut icon" href="{% static 'img/icon.png' %}" type="image/png">
   <link href="{% static 'css/bootstrap-4/bootstrap.css' %}" rel="stylesheet">
   <link href="{% static 'css/bootstrap-4/bootstrap-grid.css' %}" rel="stylesheet">
   <link href="{% static 'css/bootstrap-4/bootstrap-reboot.css' %}" rel="stylesheet">
   <link href="{% static 'css/font-awesome.min.css' %}" rel="stylesheet">
   <link href="{% static 'css/styles.css' %}" rel="stylesheet">
   <link href="{% static 'css/zbs.css' %}" rel="stylesheet">
   <script src="{% static 'js/jquery-3.2.1.min.js' %}"></script>
   <script src="{% static 'js/bootstrap-4/popper.min.js' %}"></script>
   <script src="{% static 'js/bootstrap-4/bootstrap.min.js' %}"></script>
   <script src="{% static 'js/scripts.js' %}"></script>
</head>
```

```
<body>
   {% include 'blocks/navbar.html' %}
   {% include 'blocks/messages.html' %}
   {% block content %}
   {% endblock %}
   <footer id="footer" class="footer">
      <div class="container">
         <div class="row">
            <div class="col-lg-12">
               <ul class="socials d-flex justify-content-center">
                  <li class="socials__item socials__item_fb">
                     <a href="#"></a>
                  </li>
                  <li class="socials__item socials__item_tw">
                     <a href="#"></a>
                  </li>
                  <li class="socials__item socials__item_g">
                     <a href="#"></a>
                  </li>
               </ul>
            </div>
         </div>
         <div class="row">
            <div class="col-lg-4">
               <div class="line"></div>
            </div>
            <div class="col-lg-4">
               <div class="credits">
                  2019 ©  NomadTrip.kz
               </div>
            </div>
            <div class="col-lg-4">
               <div class="line"></div>
            </div>
         </div>
      </div>
   </footer>


</body>
</html>
```

timeline.html

```
{% extends 'base.html' %}
{% load staticfiles my_filters %}
{% block content %}
   <div class="container">
      <div class="row">
         <div class="col-3">
            <div class="block left-menu">
```

```
                    <a href="{% url 'profile' %}">
                      <i class="fa fa-user-circle"></i> Мой профиль
                    </a>
                    <a href="{% url 'home' %}">
                      <i class="fa fa-newspaper-o"></i> Новости
                    </a>
                    <a href="{% url 'home' %}">
                      <i class="fa fa-users"></i> Мои друзья
                    </a>
                  </div>
                </div>
                <div class="col-6 content">
                  <div class="card">
                    <div class="card-body">
                      <form method="post" name="new-post-form" enctype="multipart/form-
data">
                        {% csrf_token %}
                        <textarea class="form-control form-control-sm" type="text" name="text"
                            placeholder="Что нового?"></textarea>
                        <label for="image">Прикрепить картинку:</label>
                        <input class="form-control form-control-sm" type="file"
name="image"><br>
                        <input class="form-control btn btn-outline-success btn-sm"
type="submit" value="Добавить">
                      </form>
                    </div>
                  </div>
                  <div class="timeline">
                    {% for post in posts %}
                      <div class="card">
                        <div class="card-body post">
                          <div class="post-title">
                            {% if post.author.profile.avatar.name %}
                              <img src="{{ post.author.profile.avatar.url }}"
                                class="post-author-img img-thumbnail">
                            {% else %}
                              <img src="{% static 'img/user.jpg' %}" class="post-author-img
img-thumbnail">
                            {% endif %}
                            <div class="post-author">
                              <a href="{% url 'view_user' post.author.username %}">
                                {{ post.author.get_full_name }}
                              </a>
                            </div>
                            <div class="post-datetime">
                              <a href="{% url 'post_view' post.id %}">
                                {{ post.datetime|date:"d M Y H:i" }}
                              </a>
                            </div>
```

```
                        </div>
                        {% if post.image.name %}
                            <img src="{{ post.image.url }}" class="img-thumbnail"><br>
                        {% endif %}
                        <div class="post-text">
                            {{ post.text|default_if_none:""|linebreaks|urlize }}
                        </div>
                    </div>
                    <div class="card-footer">
                        <div id="comments-list-post-{{ post.id }}">
                            {% for comment in post.comments.all %}
                                {% place_comment comment %}
                            {% endfor %}
                        </div>
                        <div class="comment-form">
                            {% if user.profile.avatar.name %}
                                <img src="{{ user.profile.avatar.url }}" class="post-author-img
img-thumbnail">
                            {% else %}
                                <img src="{% static 'img/user.jpg' %}" class="post-author-img
img-thumbnail">
                            {% endif %}
                            <div style="display: flex; margin-top: 4px;">
                                <input class="form-control form-control-sm comment-input"
                                       placeholder="Оставить комментарий" data-post-id="{{
post.id }}">
                            </div>
                        </div>
                    </div>
                </div>
            {% endfor %}
        </div>
    </div>
    <div class="col-3">
        <div class="block" style="text-align: center;">
            <b>{{ user.get_full_name }}</b>
            {% if user.profile.avatar.name %}
                <img src="{{ user.profile.avatar.url }}" class="mainpage-avatar-img img-
thumbnail">
            {% else %}
                <img src="{% static 'img/user.jpg' %}" class="mainpage-avatar-img img-
thumbnail">
            {% endif %}
            <div class="right-menu-links">
                <a href="{% url 'profile' %}" class="btn btn-outline-primary btn-sm">
                    <i class="fa fa-user-circle"></i> Мой профиль
                </a>
                <a href="{% url 'edit_profile' %}" class="btn btn-outline-success btn-sm">
                    <i class="fa fa-pencil-square"></i> Редактировать
```

```
                    </a>
                  </div>
                </div>
              </div>
            </div>
          </div>
        {% endblock %}
```

navbar.html

```
        {% load staticfiles %}


        <!DOCTYPE html>
        <html lang="en">
        <head>
          <meta charset="UTF-8">
          <title>Nomad Trip</title>
          <link rel="shortcut icon" href="img/icon.png" type="image/png">
          <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
          <link
href="https://fonts.googleapis.com/css?family=M+PLUS+Rounded+1c:400,700|Montserrat:400,7
00&amp;subset=cyrillic" rel="stylesheet">
          <link rel="stylesheet" type="text/css" href="{% static 'css/mimain.css' %}">
        </head>
        <body>
          <header id="header" class="header" style="background-color: #452467;">
            <div class="main-header">
              <div class="container">
              <div class="row">
                <div class="col-lg-4">
                  <a href="/"><img src="{% static 'img/logo.png' %}"></a>
                </div>
                <div class="col-lg-4">
                  <nav>
                    <ul class="menu d-flex justify-content-between">
                      <li class="menu__item">
                        <a href="/">
                          на главную
                        </a>
                      </li>
                      <li class="menu__item">
                        <a href="#">
                          о нас
                        </a>
                      </li>
                      <li class="menu__item">
```

```
                        <a  href="#">
                           как это работает?
                        </a>
                      </li>
                    </ul>
                  </nav>
                </div>
              {% if user.is_authenticated %}
                  <div class="col-lg-3 d-flex justify-content-between" >

                    <button class="auth">
                       <a href="{% url 'logout' %}">выйти</a>
                    </button>
                  </div>
                {% else %}
                  <div class="col-lg-3 d-flex justify-content-between" >
                    <button class="auth">
                       <a href="{% url 'login' %}">вход</a>
                    </button>
                    <button class="register">
                       <a href="{% url 'register' %}">регистрация</a>
                    </button>
                {% endif %}
                  </div>
                </div>
                </div>
              </div>
            </header>

        </body>
```

forms.py

```
        # coding=utf-8
        from django import forms
        from mysite.models import Profile, Post


        class RegisterForm(forms.Form):
            username = forms.CharField(label=u"Имя пользователя")
            first_name = forms.CharField(label=u"Имя")
            last_name = forms.CharField(label=u"Фамилия")
            email = forms.EmailField(label=u"Email", required=False)
            password = forms.CharField(label=u"Пароль", widget=forms.PasswordInput)
            password_confirm = forms.CharField(label=u"Подтвердите пароль",
widget=forms.PasswordInput)

            def is_valid(self):
                valid = super(RegisterForm, self).is_valid()
                if 'password' in self.cleaned_data and 'password_confirm' in self.cleaned_data:
                    if self.cleaned_data['password'] != self.cleaned_data['password_confirm']:
```

```
                self.add_error("password_confirm", u"Пароли не совпадают")
                return False
            elif len(self.cleaned_data['password']) < 5:
                self.add_error('password', u"Пароль слишком короткий")
        return valid


class ProfileForm(forms.ModelForm):
    class Meta:
        model = Profile
        exclude = ['user']


class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        exclude = ['author']
```

models.py

```
# coding=utf-8
from django.contrib.auth import get_user_model
from django.contrib.auth.models import User
from django.db import models
from rest_framework import serializers


GENDER_CHOICES = [
    ['male', u"Мужской"],
    ['female', u"Женский"],
]

REL_CHOICES = [
    ['none', u"Не определенно"],
    ['single', u"Холост"],
    ['in_a_rel', u"В отношениях"],
    ['engaged', u"Помолвлен(а)"],
    ['married', u"Женат/Замужем"],
    ['in_love', u"Влюблен(а)"],
    ['complicated', u"Все сложно"],
]


class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
verbose_name=u"Пользователь")
    avatar = models.FileField(verbose_name=u"Аватар", null=True, blank=True)
    bio = models.TextField(max_length=500, blank=True, null=True, verbose_name=u"О
себе")
    city = models.CharField(max_length=30, blank=True, null=True,
verbose_name=u"Город")
```

```
        birth_date = models.DateField(null=True, blank=True, verbose_name=u"Дата
рождения")
        gender = models.CharField(max_length=10, verbose_name=u"Пол",
choices=GENDER_CHOICES, default="male")
        relationship = models.CharField(max_length=20, verbose_name=u"Статус
отношений", choices=REL_CHOICES, default="none")


    class Post(models.Model):
        datetime = models.DateTimeField(verbose_name=u"Дата", auto_now_add=True)
        author = models.ForeignKey(User, on_delete=models.CASCADE,
verbose_name=u"Автор", related_name="posts")
        text = models.CharField(max_length=1000, verbose_name=u"Текст", null=True,
blank=True)
        image = models.FileField(verbose_name=u"Картинка", null=True, blank=True)

        class Meta:
            ordering = ["-datetime"]


    class Comment(models.Model):
        datetime = models.DateTimeField(verbose_name=u"Дата", auto_now_add=True)
        author = models.ForeignKey(User, on_delete=models.CASCADE,
verbose_name=u"Автор", related_name="comments")
        post = models.ForeignKey(Post, on_delete=models.CASCADE,
verbose_name=u"Пост", related_name="comments")
        text = models.CharField(max_length=1000, verbose_name=u"Текст", null=True,
blank=True)

        class Meta:
            ordering = ["datetime"]
```

urls.py

```
    from django.conf import settings
    from django.conf.urls import url, include
    from django.conf.urls.static import static
    from django.contrib import admin
    from django.contrib.auth.views import login

    from mysite.views import HomeView, LogoutView, RegisterView, ProfileView,
EditProfileView, PostCommentView, \
        ViewUserView, PostView

    urlpatterns = [
        url(r'^$', HomeView.as_view(), name="home"),

        url(r'^accounts/login/$', login, name="login"),
        url(r'^accounts/logout/$', LogoutView.as_view(), name="logout"),
        url(r'^accounts/register/$', RegisterView.as_view(), name="register"),
        url(r'^accounts/profile/$', ProfileView.as_view(), name="profile"),
```

```
    url(r'^accounts/profile/edit/$', EditProfileView.as_view(), name="edit_profile"),
    url(r'^post-comment/$', PostCommentView.as_view()),
    url(r'^post-view/(?P<post_id>\d+)/$', PostView.as_view(), name="post_view"),
    url(r'^user/@(?P<username>[-\w\s\d]+)$', ViewUserView.as_view(),
name="view_user"),

    url(r'^api/', include("mysite.api.urls")),

    url(r'^admin/', admin.site.urls),
] + static(settings.STATIC_URL, document_root=settings.STATICFILES_DIRS) +\
        static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

views.py

```
# coding=utf-8
from django.contrib import messages
from django.contrib.auth import logout
from django.contrib.auth.models import User
from django.http.response import HttpResponse
from django.shortcuts import redirect
from django.shortcuts import render
from django.urls import reverse
from django.views import View
from django.views.generic import TemplateView

from mysite.forms import RegisterForm, ProfileForm, PostForm
from mysite.models import Profile, Post, Comment


class LogoutView(View):
    def dispatch(self, request, *args, **kwargs):
        logout(request)
        return redirect("/")


class ProfileView(TemplateView):
    template_name = "registration/profile.html"

    def dispatch(self, request, *args, **kwargs):
        if not Profile.objects.filter(user=request.user).exists():
            return redirect(reverse("edit_profile"))
        context = {
            'selected_user': request.user
        }
        return render(request, self.template_name, context)


class RegisterView(TemplateView):
    template_name = "registration/register.html"
```

```
        def dispatch(self, request, *args, **kwargs):
            form = RegisterForm()
            if request.method == 'POST':
                form = RegisterForm(request.POST)
                if form.is_valid():
                    self.create_new_user(form)
                    messages.success(request, u"Вы успешно зарегистрировались!")
                    return redirect(reverse("login"))

            context = {
                'form': form
            }
            return render(request, self.template_name, context)

        def create_new_user(self, form):
            email = None
            if 'email' in form.cleaned_data:
                email = form.cleaned_data['email']
                User.objects.create_user(form.cleaned_data['username'], email,
form.cleaned_data['password'],
                                         first_name=form.cleaned_data['first_name'],
                                         last_name=form.cleaned_data['last_name'])


    class HomeView(TemplateView):
        template_name = "home.html"
        timeline_template_name = "timeline.html"

        def dispatch(self, request, *args, **kwargs):
            if not request.user.is_authenticated():
                return render(request, self.template_name)

            if request.method == 'POST':
                form = PostForm(request.POST, request.FILES)
                if form.is_valid():
                    form.instance.author = request.user
                    form.save()
                    return redirect(reverse("home"))
            context = {
                'posts': Post.objects.all()
            }
            return render(request, self.timeline_template_name, context)


    class EditProfileView(TemplateView):
        template_name = "registration/edit_profile.html"

        def dispatch(self, request, *args, **kwargs):
            form = ProfileForm(instance=self.get_profile(request.user))
```

```
            if request.method == 'POST':
                form = ProfileForm(request.POST, request.FILES,
instance=self.get_profile(request.user))
                if form.is_valid():
                    form.instance.user = request.user
                    form.save()
                    messages.success(request, u"Профиль успешно обновлен!")
                    return redirect(reverse("profile"))
            return render(request, self.template_name, {'form': form})

        def get_profile(self, user):
            try:
                return user.profile
            except:
                return None


    class PostCommentView(View):
        def dispatch(self, request, *args, **kwargs):
            post_id = request.GET.get("post_id")
            comment = request.GET.get("comment")
            if comment and post_id:
                post = Post.objects.get(pk=post_id)
                comment = Comment(text=comment, post=post, author=request.user)
                comment.save()
                return render(request, "blocks/comment.html", {'comment': comment})
            return HttpResponse(status=500, content="")


    class ViewUserView(TemplateView):
        template_name = "registration/profile.html"

        def dispatch(self, request, *args, **kwargs):
            username = kwargs['username']
            try:
                user = User.objects.get(username=username)
                return render(request, self.template_name, {'selected_user': user})
            except:
                return redirect("/")


    class PostView(TemplateView):
        template_name = 'blocks/post.html'

        def dispatch(self, request, *args, **kwargs):
            try:
                post = Post.objects.get(id=kwargs['post_id'])
                return render(request, self.template_name, {'post': post})
            except:
```

```
            return redirect(reverse("home"))
```

post.html

```
{% extends 'base.html' %}
{% load staticfiles my_filters %}
{% block content %}
   <div class="container">
      <div class="row">
         <div class="col-3">
            <div class="block left-menu">
               <a href="{% url 'profile' %}">
                  <i class="fa fa-user-circle"></i> Мой профиль
               </a>
               <a href="{% url 'home' %}">
                  <i class="fa fa-newspaper-o"></i> Новости
               </a>
               <a href="{% url 'home' %}">
                  <i class="fa fa-users"></i> Мои друзья
               </a>
            </div>
         </div>
         <div class="col-6 content">
            <div class="timeline">
               <div class="card">
                  <div class="card-body post">
                     <div class="post-title">
                        {% if post.author.profile.avatar.name %}
                           <img src="{{ post.author.profile.avatar.url }}"
                              class="post-author-img img-thumbnail">
                        {% else %}
                           <img src="{% static 'img/user.jpg' %}" class="post-author-img
img-thumbnail">
                        {% endif %}
                        <div class="post-author">
                           <a href="{% url 'view_user' post.author.username %}">
                              {{ post.author.get_full_name }}
                           </a>
                        </div>
                        <div class="post-datetime">
                           {{ post.datetime|date:"d M Y H:i" }}
                        </div>
                     </div>
                     {% if post.image.name %}
                        <img src="{{ post.image.url }}" class="img-thumbnail"><br>
                     {% endif %}
                     <div class="post-text">
                        {{ post.text|default_if_none:""|linebreaks|urlize }}
                     </div>
                  </div>
                  <div class="card-footer">
```

```
                        <div id="comments-list-post-{{ post.id }}">
                            {% for comment in post.comments.all %}
                                {% place_comment comment %}
                            {% endfor %}
                        </div>
                        <div class="comment-form">
                            {% if user.profile.avatar.name %}
                                <img src="{{ user.profile.avatar.url }}" class="post-author-img
img-thumbnail">
                            {% else %}
                                <img src="{% static 'img/user.jpg' %}" class="post-author-img
img-thumbnail">
                            {% endif %}
                            <div style="display: flex; margin-top: 4px;">
                                <input class="form-control form-control-sm comment-input"
                                    placeholder="Оставить комментарий" data-post-id="{{
post.id }}">
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            <div class="col-3">
                <div class="block" style="text-align: center;">
                    <b>{{ user.get_full_name }}</b>
                    {% if user.profile.avatar.name %}
                        <img src="{{ user.profile.avatar.url }}" class="mainpage-avatar-img img-
thumbnail">
                    {% else %}
                        <img src="{% static 'img/user.jpg' %}" class="mainpage-avatar-img img-
thumbnail">
                    {% endif %}
                    <div class="right-menu-links">
                        <a href="{% url 'profile' %}" class="btn btn-outline-primary btn-sm">
                            <i class="fa fa-user-circle"></i> Мой профиль
                        </a>
                        <a href="{% url 'edit_profile' %}" class="btn btn-outline-success btn-sm">
                            <i class="fa fa-pencil-square"></i> Редактировать
                        </a>
                    </div>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

comment.html

```
{% load staticfiles %}
```

Приложение В

(Обязательное)

Акты внедрения web-сервиса «Сеть гостеприимства»

Настоящий Акт свидетельствует, что web-сервис «Сеть гостеприимства», разработанный Сагимбековым М.А, внедрен в ТОО «AROS».

Процесс внедрения проходил с 14 по 16 мая 2019 г.

Заявленные характеристики системы предполагали наличие следующих основных возможностей:

- Авторизация и регистрация пользователей;
- Публикация записей (с возможностью комментирования);
- Связь между пользователями;
- Поиск зарегистрированных пользователей.

В ходе опытной эксплуатации web-сервиса подтверждено, что он обладает всеми заявленными возможностями.

На момент подписания настоящего Акта web-сервис находится на завершающей стадии разработки.

Генеральный директор _____