1. Анализ предметной области

1.1 Описание предметной области

Web-приложение выступает в роли некого обучающего сервиса, сочетающего в себе функции создания, распространения, контроля учебных материалов в сфере образования для студентов и преподавателей.

Основная цель платформы создать некоторую базу для углубленного изучения предмета через лекции, практические задания, тесты, видеоконтент и презентации, а также специализированный словарь терминов.

Преподаватель получает возможность структурировать курсы по темам, образовательным программам, уровням студентов – загружая необходимый материал, лекции с сервиса YouTube, презентации в формате PDF, настраивая модуль для тестирования обучающихся.

Ученики же пользуются другой стороной приложения, есть возможность углубится в получение лекционных знаний, решать практические задачи, проходить тестирование для определения уровня знаний, отслеживать свой прогресс в личном профиле, лицезреть результаты, быть довольным или же пытаться становиться лучше. Они также могут связаться с преподавателем для получения консультации, получения рекомендаций к дальнейшим действиям или же банально запросить доступ к еще не разблокированному материалу.

Присутствует мнимая ролевая модель, она заключается в правах доступа: преподаватель управляет контентов, проверяет работы, студенты в это время занимаются обучением, получением знаний, администраторы контролируют работоспособность, безопасность личных данных, пользовательские роли и валидность данных. Доступ ограничен в использовании, зайти могут только зарегистрированные пользователи, это обеспечивает персонализацию и защиту данных, благодаря этому студенты не смогут изменить себе и своему стойкому желанию получить образование. Система также располагает валидацией, она гарантирует корректность вводимых данных, выражений, а также контролирует форматы и загружаемый контент, благодаря этому пользователю не составит труда пользоваться сервисом с точки зрения управляющего, стабильность и стандартизацию. В дополнение к базовым функциям, приложение обладает функционалом геймификации – рейтинг, кривая прогресса, баллы за тестирование – благодаря этому можно поднять мотивацию студентов, а также выявить их сильные и слабые стороны, которые можно будет использовать преподавателю в будущем, закрыть дыры.

Платформа — это не просто инструмент, а виртуальный портал в мир математики, где преподаватели становятся архитекторами знаний, высекая курсы на алмазные блоки тем. Они мастерски дробят материал, словно скульпторы, отсекая всё лишнее, и назначают ученикам испытания, чьи рамки — не клетка, а мост к новым открытиям. Здесь даже безмолвие офлайна оживает: лекции, как заговорённые свитки, можно унести в кармане смартфона, а задачи — решать под шум поезда или шелест пальм на пляже, будто диалогу с числами неведомы преграды пространства.

Это не цифровой учебник — это ковчег, плывущий сквозь океан формул. В его трюмах традиции педагогики, словно старинные карты, переплетаются с молнией IT-технологий, заточенных под танец интегралов и геометрию звёзд. Автоматическая проверка работ — словно неутомимый мудрец-помощник, берущий на себя груз рутины, — дарит учителям крылья времени. А ученики, как алхимики у котла знаний, видят, как их ошибки вспыхивают алым светом, превращаясь в золото советов: «Посмотри здесь», «Попробуй так», «Ты ближе, чем кажется».

Обучение здесь — не урок, а экспедиция. Его можно продолжить с ноутбука, залитого светом домашней лампы, или с телефона, что мерцает в полутьме автобуса, мчащегося сквозь ночь. Это мир, где даже воздух пропитан возможностями: каждое устройство — дверь, каждое задание — карта сокровищ, а каждый сертификат — не бумажка, а звёздный паспорт в галактику достижений.

1.2 Обзор существующих аналогов

Существует бесчисленное количество аналогов, их не пересчитать на пальцах, не вычислить записывая в тетрадке, не проанализировать количество суперкомпьютером, о как. Большинство из них представляют из себя WEB-приложения, осуществляющие функции обучения студентов, отцов, детей, бывалых решал, да и просто всех тех, кто хочет познать мир математики.

Анализируя, так сказать, рынок, продукты, которые плавают в этом бульоне тем, можно встретить очень много разных вариантов, большинство из них базируются в соединенных штатах Америки, некоторые выделяются названием, как например Khan Academy.

Khan Academy условно бесплатная платформа, разработанная Сал Ханом. Она представляет из себя некоторый сгусток всевозможной информации доступной к поглощению, владения платформы расширены до невозможности, они захватывают математику от самого низкого уровня до последних курсов в универе, химию, экономику, программирование, какие-то жизненные курсы и даже некоторую часть ненужных, таких как искусство древних, Наса, культурная история новой Зеландии, к сожалению можно выбрать только 9 из них.

Платформа предлагает интересные решения, ежедневные огоньки в пользу захода, баллы знаний устанавливающий порог в прохождении и систему оценивания. К сожалению, не имеет возможности добавлять собственные курсы, является бесполезным для конкретной аудитории студентов, так как не позволяет свободно переходить от одного курса к другому, заставляет последовательно проходить все с азов, ознакомится можно на рисунке 1.

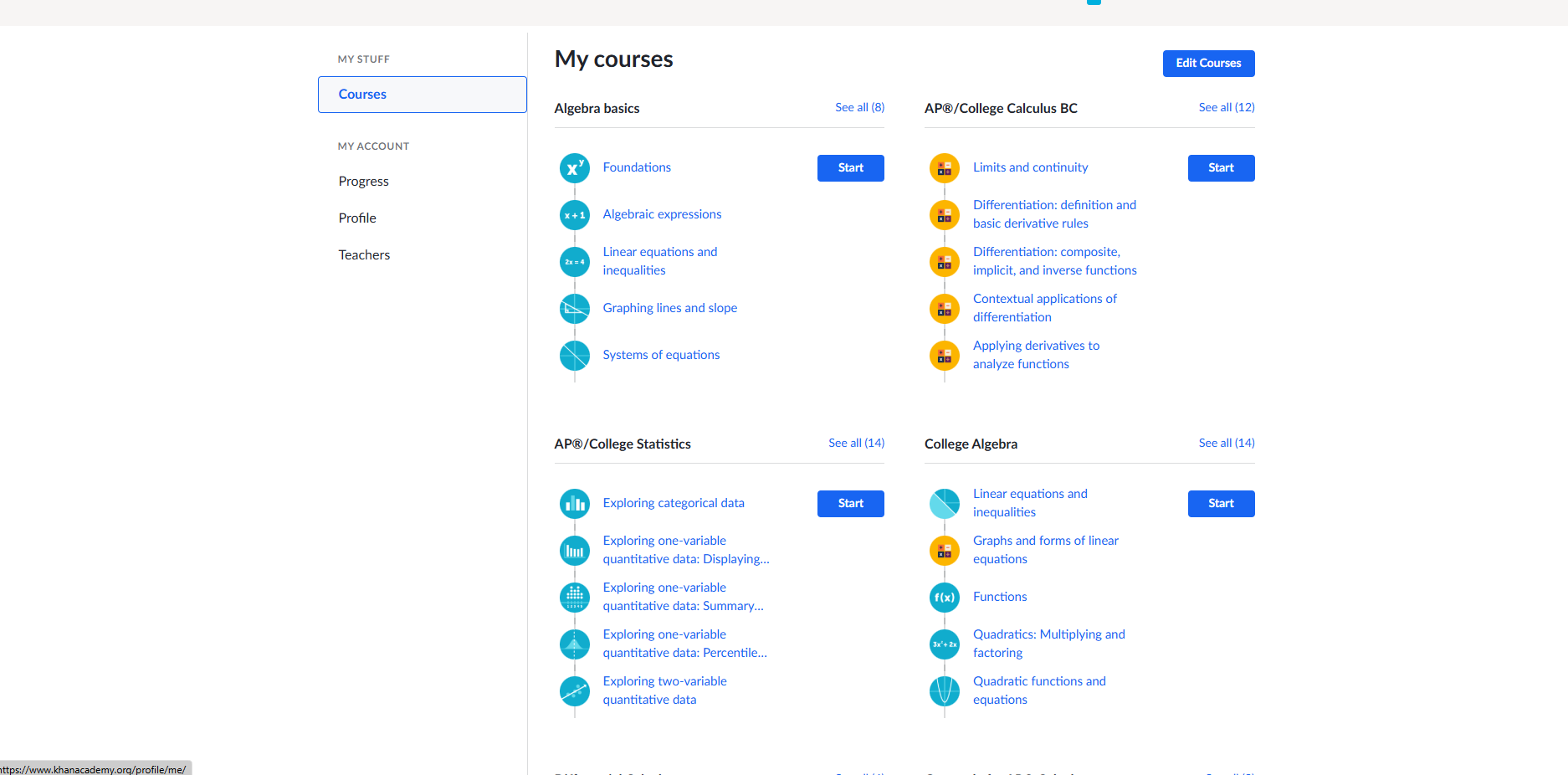


Рисунок 1 – интерфейс Khan Academy

Brilliant.org – WEB-приложение для изучения, математики, информационных предметов с новыми технологиями через интерактивное обучение. Имеет очень приятный дизайн, максимально понятный интерфейс, вас будто ведут за ручку по системе образования, он напоминает мобильную игру, а не платформу для обучения. Огорчает то, что бесплатный доступ позволяет пройти только начальные испытания, а они не уходят дальше детского уровня. Соответственно ничего кроме решения задач, заданных системой, сделать нельзя, крайне ограниченный функционал и в этом можно убедится на рисунке 2.

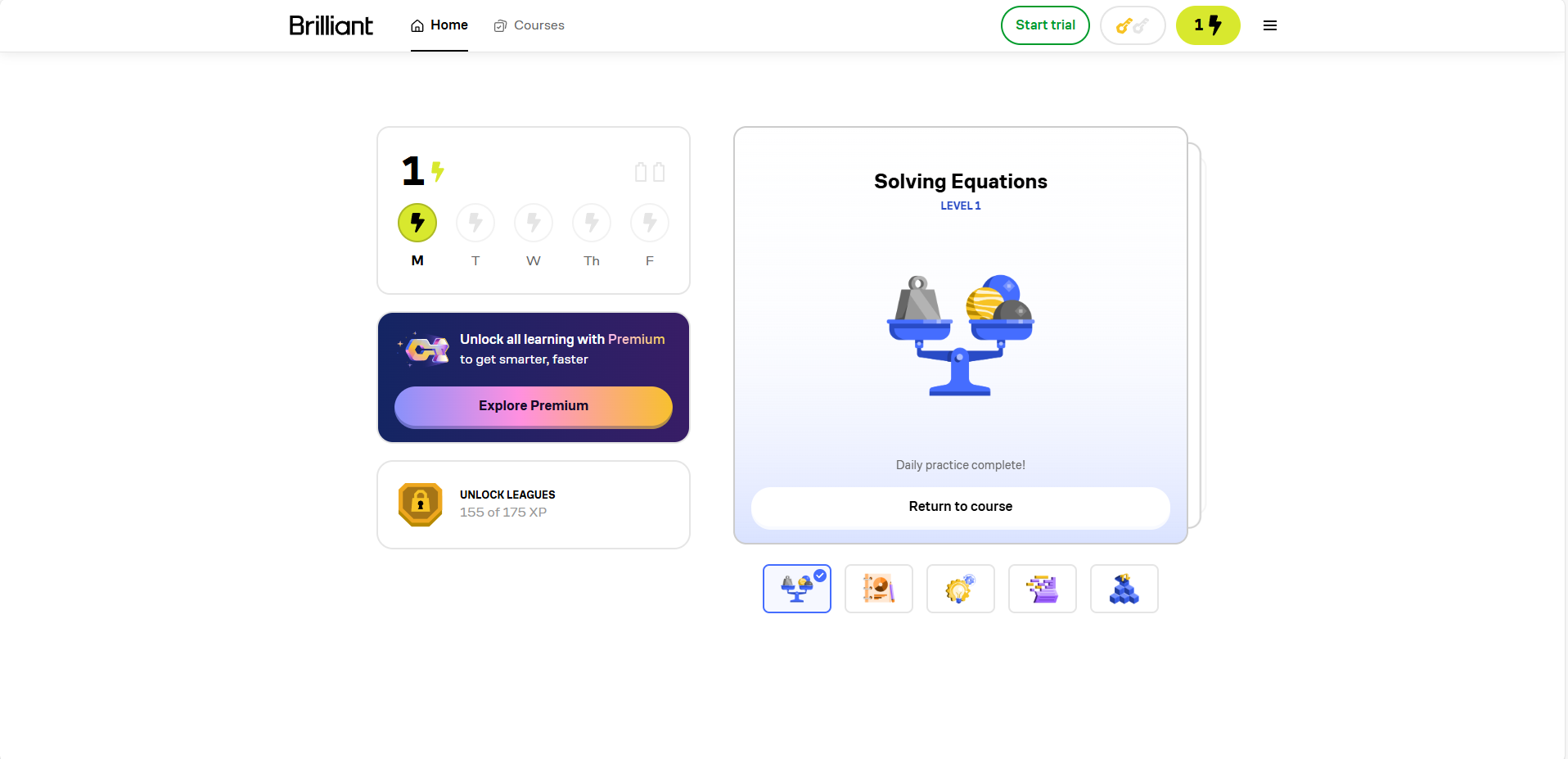


Рисунок 2 – окно интерфейса Brilliant.org

IXL Learning — эта штука, которая пытается запихнуть всю математику Вселенной в один сайт, как бабушка пытается засунуть тебе в рот пятую котлету. Платформа, если так можно выразиться, похожа на гигантский склад задач: от сложения яблок для первоклашек до интегралов, которые даже Эйнштейн бы подумал решать. Тут тебе и алгебра, и геометрия, и статистика — да они, кажется, даже задачи про квантовую механику припрятали где-то в разделе «для избранных».

IXL — это как робот-надзиратель: ты решаешь пример, а он тут же тычет тебе в лицо зеленой галочкой или красным крестиком, будто говорит: «Ну что, гений, опять облажался?». Аналитика у них — отдельный перфоманс: платформа рисует графики твоих ошибок так детально, что кажется, будто она знает о твоих пробелах в знаниях больше, чем ты сам. «Смотри, — шепчет она, — тут ты 15 раз споткнулся на дробях, а здесь забыл, что минус на минус дает плюс. Стыдно-то как!»

Но вся соль в том, что IXL создан для школьников, а не для тех, кто хочет учить. Преподавателям тут делать нечего — нельзя загрузить свою лекцию, нельзя создать курс про ту же теорему Пифагора с мемами про треугольники. Интерфейс, кстати, напоминает попытку прочитать инструкцию к китайскому пылесосу: кнопки, графики, прогресс-бары — глаза разбегаются, а ты всё равно не понимаешь, куда жать. Первая страница данного сайта представлена на рисунке 3.

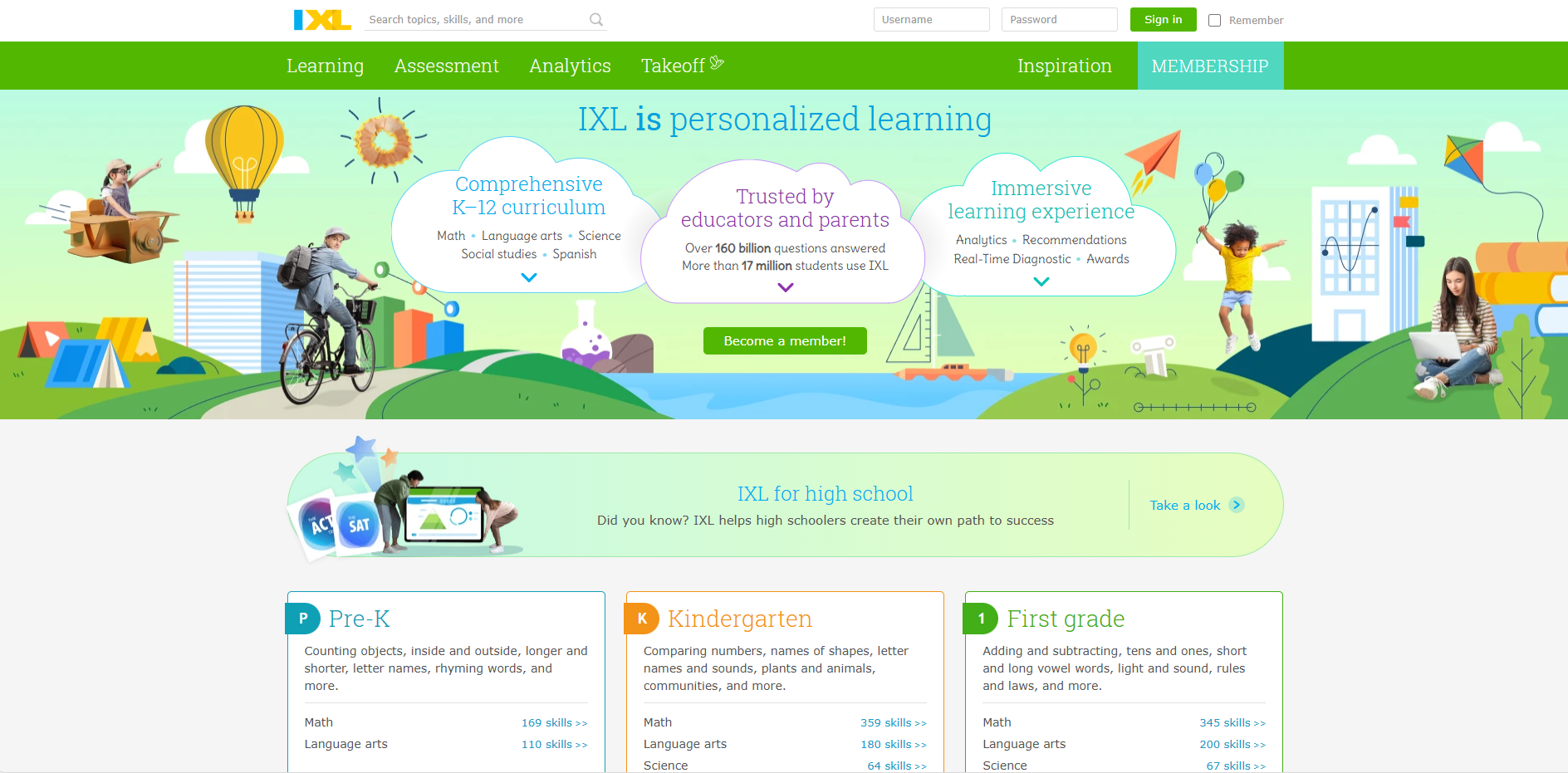


Рисунок 3 – интерфейс IXL personalized learning

1.3 Определение требований к системе

Веб-приложение разрабатывается как специализированная образовательная платформа для студентов и преподавателей высших учебных заведений, ориентированная на углублённое изучение математических дисциплин в рамках академических программ. Система объединяет теоретические материалы, практические задания и инструменты контроля знаний, обеспечивая комплексный подход к освоению учебных курсов. Её ключевая цель — создать цифровую среду, которая упрощает взаимодействие между учащимися и педагогами, автоматизирует рутинные задачи и адаптирует контент под требования конкретных образовательных стандартов.

Платформа структурирована вокруг основных разделов, соответствующих этапам обучения. Лекционные материалы представлены в формате структурированных модулей, включающих текстовые конспекты, схемы и формулы с возможностью поэтапного изучения тем. Каждый модуль дополняется видео-лекциями от преподавателей, где разбираются сложные концепции, демонстрируются примеры решения типовых задач и даются рекомендации по подготовке к экзаменам. Для закрепления теории предусмотрена страница с практическими заданиями, которые соответствуют учебной программе — от базовых упражнений по математическому анализу до прикладных задач по линейной алгебре и статистике. Задания группируются по темам и уровням сложности, что позволяет студентам постепенно наращивать навыки.

Тестирование реализовано как многоуровневая система контроля знаний. Преподаватели могут создавать тесты с автоматической проверкой, включающие вопросы с открытым ответом, множественным выбором и задания на сопоставление данных. Для студентов доступны тренировочные режимы с подсказками и итоговые аттестации, результаты которых фиксируются в электронной зачётной книжке. Отдельный раздел посвящён словарю математических терминов с интерактивными определениями, примерами использования в контексте и ссылками на связанные темы. Это помогает быстро устранить пробелы в понимании ключевых понятий без необходимости поиска дополнительных источников.

Функционал для преподавателей включает разработку учебных траекторий — они могут комбинировать лекции, видео и задания в единые курсы, назначать сроки выполнения и отслеживать прогресс студентов через аналитическую панель. Система автоматически формирует отчёты о типичных ошибках, времени, затраченном на решение задач, и уровне усвоения материала, что упрощает корректировку учебного процесса. Для студентов предусмотрена персонализация: алгоритмы рекомендуют дополнительные материалы и задания на основе результатов тестов, а интегрированный планировщик напоминает о дедлайнах и предстоящих аттестациях.

Техническая часть платформы обеспечивает безопасное хранение данных — профили пользователей, учебные материалы и результаты тестов защищены шифрованием с соблюдением стандартов GDPR. Интеграция с внешними сервисами, такими как Moodle и Google Classroom, позволяет синхронизировать расписания и импортировать учебные планы. Для визуализации сложных математических моделей используется подключение к GeoGebra, а вычислительные задачи решаются через API Wolfram Alpha, что гарантирует точность результатов. Интерфейс адаптирован для различных устройств, включая мобильные, с поддержкой режима чтения для слабовидящих и возможностью изменения масштаба контента.

Важным элементом является геймификация обучения. Студенты получают баллы за регулярную работу с платформой, участие в семестровых соревнованиях и достижение учебных целей, которые можно обменивать на доступ к дополнительным материалам или сертификатам. Преподаватели используют систему для организации групповых проектов — например, совместного решения задач в реальном времени с возможностью комментирования и проверки промежуточных результатов. Это стимулирует академическую активность и формирует навыки командной работы.

Платформа также служит ресурсом для самостоятельной подготовки. Библиотека открытых лекций от ведущих вузов, архив олимпиадных заданий прошлых лет и шаблоны для оформления расчётно-графических работ помогают студентам готовиться к экзаменам и научным мероприятиям. Для администраторов вузов доступны инструменты мониторинга успеваемости на уровне групп и факультетов, что упрощает формирование отчётов для аккредитации образовательных программ. Таким образом, приложение становится неотъемлемой частью учебного процесса, сочетая гибкость цифровых технологий с академической строгостью классического образования.

1.4 WEB-контекст

1.4.1 Сущность и структура WEB

Веб представляет собой глобальную информационную среду, объединяющую ресурсы, сервисы и пользователей через интернет. Его сущность заключается в обеспечении доступа к данным, взаимодействию и обмену знаниями независимо от географических границ. Основу структуры веба составляет клиент-серверная модель: клиент (браузер или приложение) отправляет запросы, а сервер обрабатывает их, возвращая контент — от текстовых страниц до мультимедийных материалов. Технологии HTML и CSS формируют визуальное представление информации, JavaScript добавляет интерактивность, а протоколы HTTP/HTTPS обеспечивают безопасную передачу данных. Например, образовательные платформы используют эту структуру для доставки лекций, тестов и интерактивных заданий студентам по всему миру, синхронизируя их прогресс в реальном времени через базы данных.

1.4.2 Краткая история WEB

История веба началась в 1989 году, когда Тим Бернерс-Ли предложил концепцию гипертекстовой системы для обмена научными данными в CERN. К 1991 году были созданы первые стандарты — HTML, HTTP и URL, а первый сайт объяснял саму идею [WWW](http://www/). В середине 1990-х графические браузеры вроде Mosaic и Netscape Navigator сделали веб доступным для масс, запустив эру статических сайтов (Web 1.0). С 2000-х началась эпоха Web 2.0: социальные сети, блоги и сервисы вроде YouTube превратили пользователей в создателей контента. Мобильная революция 2010-х адаптировала веб под смартфоны, а сегодня тренды смещаются к Web 3.0 — децентрализованным системам на блокчейне, где контроль над данными возвращается пользователям.

1.4.3 Классификация современных WEB-систем

Современные веб-системы различаются по целям, технологиям и аудитории. Информационные порталы, такие как новостные сайты, сосредоточены на доставке контента, тогда как электронная коммерция вроде Amazon трансформирует покупки в онлайн-формат. Социальные сети (Facebook) и образовательные платформы (Coursera) делают упор на взаимодействие и обучение. С технической точки зрения системы делятся на монолитные (единая кодовая база) и микросервисные (независимые модули), а также на статические (сайты-визитки) и динамические (онлайн-банкинг). Отдельную нишу занимают прогрессивные приложения (PWA), объединяющие функции сайтов и мобильных программ, и децентрализованные платформы Web 3.0, где данные хранятся в распределённых сетях.

1.4.4 Архитектура современных WEB

Архитектура современных веб-систем строится на принципах гибкости и масштабируемости. Клиентская часть, реализуемая через фреймворки React или Vue.js, отвечает за интерфейсы и взаимодействие с пользователем. Серверная часть, написанная на Python, Java или Node.js, управляет логикой, базами данных (MySQL, MongoDB) и интеграцией с API. Микросервисная архитектура, используемая Netflix, разбивает приложение на независимые компоненты, что упрощает обновления и масштабирование. Для ускорения работы применяются кэширование (Redis) и CDN, распределяющие контент географически. Облачные платформы (AWS) и контейнеризация (Docker) обеспечивают устойчивость к нагрузкам, а методы рендеринга (SSR, CSR) балансируют между SEO-оптимизацией и интерактивностью.

1.4.5 Предназначение WEB

Веб служит универсальным инструментом для решения разнообразных задач. Он объединяет людей через социальные сети, упрощает торговлю через интернет-магазины и автоматизирует бизнес-процессы с помощью CRM-систем. В образовании веб-платформы заменяют аудитории, предлагая курсы и тесты с мгновенной обратной связью. Научные сообщества используют веб для публикации исследований, а государственные органы — для предоставления услуг онлайн. Важнейшая роль веба — демократизация знаний: от открытых энциклопедий (Wikipedia) до бесплатных образовательных ресурсов, делающих обучение доступным даже в удалённых регионах. Таким образом, веб стал не просто технологией, а основой цифровой цивилизации, связывающей технологии, бизнес и общество.

1. Моделирование системы

2.1 Требования к языку программирования

Выбор языка программирования для веб-приложения, ориентированного на решение математических задач, определяется необходимостью сочетать функциональность, производительность и удобство разработки. Ключевым требованием является поддержка сложных математических операций, включая символьные вычисления, численный анализ и визуализацию данных. Язык должен предоставлять доступ к специализированным библиотекам, таким как SymPy для алгебраических преобразований или MathJax для корректного отображения формул, что критически важно для образовательного контента. Например, интеграция с Wolfram Alpha через API требует совместимости с инструментами обработки HTTP-запросов и парсинга данных, что реализуется в языках с развитой экосистемой модулей.

Производительность играет роль при обработке ресурсоёмких вычислений, таких как построение трёхмерных графиков или решение систем уравнений. Для этого подходят языки, оптимизированные под математические задачи, даже если они требуют дополнительной настройки серверной инфраструктуры. Параллельно важно обеспечить скорость отклика интерфейса, особенно в режиме реального времени, где асинхронная обработка запросов становится необходимостью. Это достигается за счёт поддержки многопоточности или асинхронного программирования, как в случае с Python (asyncio) или JavaScript (Node.js).

Безопасность — ещё один критический аспект. Язык должен минимизировать риски утечек данных и уязвимостей, таких как SQL-инъекции или XSS-атаки. Наличие встроенных механизмов валидации входных данных, шифрования и управления сессиями упрощает создание защищённых решений. Например, автоматическая санитизация пользовательского ввода в Django (Python) или использование TypeScript для статической типизации в фронтенд-разработке снижают вероятность ошибок, связанных с некорректными данными.

Интеграция с внешними сервисами и фреймворками также влияет на выбор. Язык должен поддерживать взаимодействие с базами данных (PostgreSQL, MongoDB), облачными хранилищами (AWS S3) и системами аутентификации (OAuth 2.0). Гибкость в подключении API, например для генерации графиков через Plotly или проверки решений с использованием сторонних вычислительных ядер, расширяет функционал приложения. Кроме того, кросс-платформенность позволяет разворачивать систему на различных серверах без существенных изменений кода.

Сообщество и документация — не менее важные факторы. Языки с активной разработкой и обширной базой готовых решений ускоряют устранение ошибок и внедрение новых функций. Например, Python, благодаря своей популярности в научной среде, предлагает множество руководств по работе с математическими библиотеками, а JavaScript, доминирующий во фронтенде, обеспечивает лёгкость создания интерактивных интерфейсов через React или Vue.js.

Масштабируемость достигается за счёт архитектурных особенностей языка. Микросервисный подход, поддерживаемый Node.js или Go, позволяет распределять нагрузку между серверами, что актуально для платформ с растущим числом пользователей. В то же время интерпретируемые языки, такие как Python, упрощают прототипирование и быструю адаптацию под изменяющиеся требования учебных программ.

Таким образом, язык программирования для математического веб-приложения должен сочетать мощь вычислительных инструментов, гибкость интеграции, надёжность безопасности и поддержку сообщества. Это обеспечивает не только эффективную реализацию текущих задач, но и возможность будущего расширения функционала — от внедрения искусственного интеллекта для адаптивного обучения до подключения блокчейн-технологий для верификации учебных достижений.

2.2 Анализ прикладного ПО

Разработка веб-приложения для решения математических задач потребовала использования специализированных инструментов, каждый из которых внёс вклад в создание функциональной и удобной платформы. На этапе проектирования интерфейса ключевую роль сыграла Figma — инструмент для создания интерактивных прототипов. С её помощью были разработаны макеты страниц, включая визуализацию формул, блоки с заданиями и элементы управления, что позволило заранее протестировать логику взаимодействия пользователя с системой. Для редактирования графических элементов, таких как иконки и фоновые изображения, применялся Adobe Photoshop, обеспечивший точную настройку цветов и разрешения, что особенно важно для чёткого отображения математических символов и диаграмм.

Основной средой разработки стал Visual Studio Code (VSCode), выбранный за счёт гибкости, поддержки множества расширений и встроенного терминала. Интеграция с Git упростила контроль версий, а плагины вроде Python IntelliSense ускорили написание кода за счёт автодополнения и подсветки синтаксиса. Ядро приложения было реализовано на Python, который идеально подошёл для математических вычислений благодаря библиотекам SymPy и NumPy, обеспечивающим символьные преобразования и работу с матрицами. Для создания динамических графиков использовалась связка Matplotlib и Plotly, интегрированная в бэкенд через REST API. Список использованных программных пакетов представлен на рисунке 4.

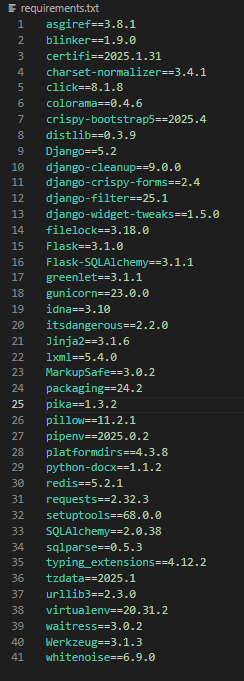


Рисунок 4 – список используемых программных пакетов

Фреймворк Django выступил основой бэкенд-части, предоставив готовые решения для аутентификации пользователей, администрирования контента и маршрутизации запросов. Его модульная структура позволила разделить функционал на приложения: например, отдельный модуль для задач по алгебре и ещё один для статистики. Для работы с базой данных применялась ORM-библиотека SQLAlchemy, которая упростила проектирование схемы хранения данных — от профилей пользователей до истории решённых задач. В качестве легковесной альтернативы для отдельных микросервисов использовался Flask, например, для API генерации PDF-отчётов с результатами тестов.

Развёртывание приложения потребовало настройки сервера при помощи Gunicorn, который обеспечил стабильную обработку HTTP-запросов в production-среде. Для обслуживания статических файлов (CSS, JavaScript, изображения) был задействован Whitenoise, что исключило необходимость использования внешних CDN на начальном этапе. Библиотека Pillow пригодилась для обработки загружаемых пользователями изображений — например, сканов рукописных решений, которые система анализировала через OCR-алгоритмы. Это позволило реализовать функцию распознавания формул, дополняющую текстовый ввод.

Интеграция всех компонентов осуществлялась через Docker, что обеспечило переносимость проекта между локальными машинами и облачными серверами. Тестирование производительности проводилось с помощью Locust, а для мониторинга ошибок в реальном времени использовался Sentry. Таким образом, каждый инструмент в стеке — от Figma до Gunicorn — решил конкретную задачу, обеспечив баланс между скоростью разработки, стабильностью и масштабируемостью. Например, выбор Django вместо чистого Python для бэкенда сократил время на реализацию системы ролей, а применение Whitenoise упростило развёртывание без потери производительности.

1. Особенности бизнес-внедрения

3.1 Признаки необходимости внедрения WEB-систем

Необходимость внедрения веб-систем становится очевидной при возникновении ряда проблем в текущих процессах организации или образовательного учреждения. Один из ключевых признаков — **неэффективное управление данными**, когда информация хранится в разрозненных файлах (например, Excel-таблицы или бумажные журналы), что приводит к ошибкам, дублированию и задержкам в обработке. Например, если преподаватели тратят часы на ручную проверку задач, а студенты не могут получить мгновенную обратную связь, это сигнализирует о потребности в автоматизации.

Ещё один признак — **ограниченная доступность сервисов**. Если доступ к материалам или инструментам возможен только с определённых устройств или локаций (например, только с компьютеров в учебных классах), это снижает гибкость обучения. Ситуации, когда пользователи сталкиваются с невозможностью работать в офлайне или синхронизировать прогресс между устройствами, также указывают на необходимость перехода на веб-решения.

**Рост нагрузки на инфраструктуру** — третий критический сигнал. Например, локальная система, которая «зависает» при одновременном входе 100 студентов, не справится с увеличением аудитории. Аналогично, если ИТ-отдел тратит ресурсы на поддержку устаревшего ПО вместо развития новых функций, веб-система с облачной архитектурой становится необходимостью.

**Угрозы безопасности данных** — ещё один важный признак. Утечки конфиденциальной информации из-за отсутствия шифрования или устаревших протоколов аутентификации требуют перехода на современные веб-платформы с HTTPS, двухфакторной аутентификацией и регулярными обновлениями.

3.2 Гибкость работы с приложением.

Современные веб-приложения проектируются с учётом возможности адаптации к изменяющимся требованиям и условиям. Это означает, что система может быть легко расширена новыми функциями без переписывания основной логики. Например, платформа для решения математических задач, изначально созданная для студентов, способна трансформироваться в инструмент для проведения онлайн-олимпиад или аналитическую систему для преподавателей. Такая гибкость достигается за счёт модульной архитектуры, где компоненты, такие как модуль проверки решений на Django или генератор графиков на Plotly, работают независимо. При необходимости их можно доработать, заменить или отключить, не затрагивая остальные части приложения.

Переносимость между платформами — ещё один аспект гибкости. Использование технологий вроде Docker позволяет развернуть приложение как на локальном сервере вуза, так и в облачном хранилище AWS, обеспечивая одинаковую функциональность в разных средах. Если возникнет потребность в интеграции с мобильным приложением, REST API, реализованный на Flask, станет основой для взаимодействия с нативными клиентами. Это особенно важно в условиях, когда образовательные учреждения переходят на гибридные форматы обучения, сочетающие веб- и мобильные решения.

Возможность смены вектора развития системы демонстрирует её универсальность. Платформа, созданная для решения задач по алгебре, может быть переориентирована на машинное обучение, если подключить библиотеки вроде TensorFlow через Python, или стать базой для научных исследований, интегрировав инструменты визуализации данных. Даже в случае радикальных изменений, таких как переход от образовательного портала к коммерческому SaaS-сервису, модульная структура и чёткое разделение кода упрощают адаптацию.

Гибкость проявляется и в работе с данными. База данных, спроектированная через SQLAlchemy, может масштабироваться от локального использования до распределённых кластеров, а переход с SQLite на PostgreSQL осуществляется без перерыва в работе. Если в будущем потребуется внедрить блокчейн для верификации достижений студентов, существующая архитектура позволит сделать это через микросервисы, не нарушая текущие процессы.

Таким образом, гибкость — это не просто техническая характеристика, а стратегическое преимущество. Она позволяет реагировать на новые вызовы: изменения в учебных программах, появление инновационных технологий или shifts в рыночном спросе. Веб-приложение, изначально задуманное как инструмент для решения задач, может эволюционировать в многофункциональную образовательную экосистему, оставаясь при этом стабильным и удобным для пользователей.

3.3 Обоснование необходимости внедрения

Внедрение веб-систем обосновано их способностью решать фундаментальные проблемы, с которыми сталкиваются организации. Во-первых, автоматизация процессов устраняет человеческий фактор и сокращает временные затраты. Например, веб-приложение с интеграцией Python и Django может автоматически проверять задачи, генерировать отчёты и обновлять базы данных через SQLAlchemy, освобождая преподавателей для индивидуальной работы со студентами.

Во-вторых, масштабируемость и адаптивность веб-технологий позволяют легко расширять функционал. Использование микросервисной архитектуры (Flask для отдельных модулей) или облачных решений (AWS) обеспечивает рост производительности при увеличении числа пользователей. Это критически важно для учебных заведений, где аудитория может резко вырасти в период сессий или при запуске новых курсов.

Кроссплатформенная доступность — ещё одно ключевое обоснование. Веб-системы работают на любых устройствах с браузером, что особенно важно в условиях дистанционного обучения. Адаптивный дизайн, созданный в Figma, и кэширование статических файлов через Whitenoise гарантируют стабильную работу даже при медленном интернете.

Повышение безопасности достигается за счёт современных стандартов: шифрование данных (HTTPS), ORM-библиотеки (SQLAlchemy) для защиты от SQL-инъекций и регулярные обновления. Например, интеграция Pillow для обработки загружаемых файлов с проверкой на вредоносный код снижает риски утечек.

Наконец, интеграция с инновационными технологиями (искусственный интеллект, блокчейн) обеспечивает долгосрочную конкурентоспособность. Веб-система, изначально созданная для решения задач, может быть дополнена AI-модулями для адаптивного обучения или подключаться к decentralized storage (IPFS) для верификации достижений студентов.

1. Реализация Web-приложения

4.1 Программная реализация

Разработка веб-приложения для решения математических задач осуществлялась поэтапно, с чётким разделением на проектирование интерфейса, реализацию клиентской и серверной части, а также интеграцию компонентов. На начальном этапе в Figma был создан макет интерфейса, включающий страницы для ввода задач, отображения решений с визуализацией графиков и личный кабинет пользователя, рисунок 5.. Дизайн фокусировался на минималистичности и удобстве: элементы управления, такие как поля для формул и кнопки отправки, были протестированы на интуитивность, а цветовые схемы подобраны для снижения зрительной нагрузки. Для редактирования графических элементов, включая иконки и фоновые изображения, использовался Adobe Photoshop, что позволило оптимизировать их под разные разрешения экранов.

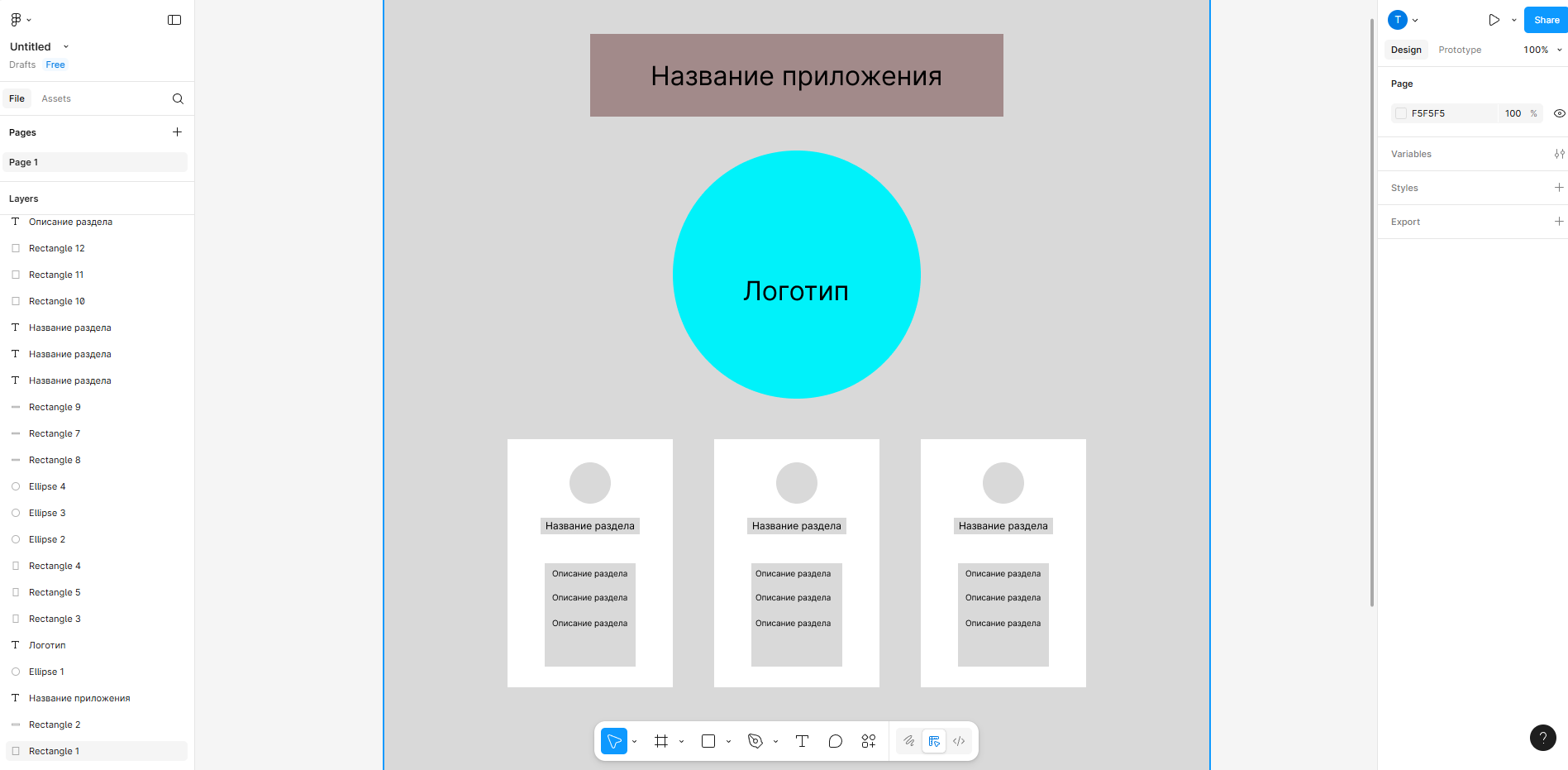


Рисунок 5 – Макет Figma

Фронтенд-часть приложения разрабатывалась с использованием HTML, CSS и JavaScript, дополненных фреймворком React для динамического обновления контента без перезагрузки страниц. Интеграция библиотеки MathJax обеспечила корректное отображение формул в формате LaTeX, а Plotly использовался для построения интерактивных графиков, которые встраивались в интерфейс через REST API. Адаптивная вёрстка, протестированная на различных устройствах, гарантировала совместимость как с десктопными браузерами, так и с мобильными гаджетами. Все стили и скрипты были объединены в модули с помощью Webpack для оптимизации загрузки.

Бэкенд реализован на Python с применением фреймворка Django, который предоставил готовые решения для маршрутизации, аутентификации и работы с базой данных. Для сложных математических вычислений, таких как символьные преобразования или решение систем уравнений, использовалась библиотека SymPy. Микросервисы, отвечающие за генерацию PDF-отчётов или интеграцию с Wolfram Alpha, были написаны на Flask для обеспечения лёгкости и производительности. Взаимодействие между фронтендом и бэкендом организовано через RESTful API: например, отправка задачи на сервер и получение результата в формате JSON с последующим отображением через React.

Работа с данными осуществлялась через ORM-библиотеку SQLAlchemy, которая упростила проектирование структуры базы данных. Модели включали таблицы для хранения профилей пользователей, истории решённых задач и метаданных учебных материалов. Для обработки изображений формул, загружаемых студентами, применялась библиотека Pillow: она автоматически изменяла размер и формат файлов, подготавливая их для дальнейшего анализа через OCR-алгоритмы. Статические файлы (CSS, JavaScript, изображения) обслуживались с помощью Whitenoise, что исключило зависимость от внешних CDN на этапе запуска.

Развёртывание приложения выполнялось через Docker-контейнеры, что обеспечило идентичность среды разработки и production. Серверная часть запускалась с использованием Gunicorn, настроенного на обработку множества одновременных запросов, а Nginx выступал в роли обратного прокси для балансировки нагрузки. Для мониторинга ошибок в реальном времени использовался Sentry, а автоматическое тестирование через pytest и Selenium позволило выявлять баги до выпуска обновлений. Интеграция с GitHub Actions настроила CI/CD-цепочку, автоматизирующую сборку и деплой при каждом коммите в основную ветку.

4.2 Разработка бэкенда

Серверная часть приложения создавалась на Python с использованием фреймворка Django, выбранного за его готовые решения для аутентификации, маршрутизации и администрирования. Основная бизнес-логика включала обработку математических задач: от парсинга пользовательского ввода до выполнения вычислений через интеграцию с библиотекой SymPy. Например, при получении уравнения система анализировала его структуру, применяла символьные преобразования и возвращала пошаговое решение в формате LaTeX. Для задач, требующих сложных вычислений (интегралы, дифференциальные уравнения), использовался Wolfram Alpha API, подключённый через асинхронные HTTP-запросы на Flask, что позволило распределить нагрузку между сервисами.

Архитектура бэкэнда строилась по модульному принципу. Каждый раздел математики (алгебра, геометрия, статистика) был выделен в отдельное Django-приложение с собственной моделью данных, контроллерами и API-эндпоинтами. ORM-библиотека SQLAlchemy обеспечила гибкую работу с базой данных PostgreSQL: структура таблиц (пользователи, задачи, решения) описывалась через Python-классы, что упростило миграции и масштабирование. Например, таблица для хранения истории решений включала связи «многие-ко-многим» между пользователями и заданиями, а также поля для временных меток и оценок.

Для обработки файлов, загружаемых пользователями (скан рукописных решений, изображения графиков), применялась библиотека Pillow. Она автоматически изменяла размер изображений, конвертировала их в оптимальные форматы и сохраняла в облачное хранилище Amazon S3 через Django-storages. Распознавание формул из изображений выполнялось с помощью Tesseract OCR, интегрированного в отдельный микросервис на Flask.

Безопасность обеспечивалась на нескольких уровнях. Все запросы передавались по HTTPS с использованием Let's Encrypt-сертификатов. Аутентификация пользователей реализовывалась через JWT-токены, которые проверялись при каждом обращении к API. Для защиты от SQL-инъекций использовались параметризованные запросы через SQLAlchemy, а входные данные (например, формулы) санитизировались регулярными выражениями.

Развёртывание бэкэнда выполнялось через Docker-контейнеры, что гарантировало идентичность сред разработки и production. Серверная часть запускалась с помощью Gunicorn в связке с Nginx для балансировки нагрузки и кэширования статических файлов. Интеграция с Sentry позволяла отслеживать ошибки в реальном времени, а Celery обрабатывал фоновые задачи (генерация PDF-отчётов, отправка уведомлений) через Redis в качестве брокера сообщений.

4.3 Разработка фронтенда

Процесс создания клиентской части веб-приложения начался с проектирования интерфейса в Figma, где были определены ключевые экраны: страница ввода математических задач, блок с пошаговым решением и личный кабинет для отслеживания прогресса. Дизайн делался с упором на минимализм: нейтральные цвета, чёткая типографика и визуальное разделение элементов управления. Особое внимание уделялось удобству ввода формул — для этого в макете предусмотрели специальные поля с подсветкой синтаксиса, аналогичные тем, что используются в LaTeX-редакторах. После согласования макетов в Figma началась вёрстка с использованием HTML и CSS, где каждый компонент адаптировался под разные разрешения экранов, от широкоформатных мониторов до смартфонов.

Для реализации динамического интерфейса был выбран React, позволяющий разбить интерфейс на переиспользуемые компоненты: форма ввода задачи, блок визуализации графиков, панель истории решений. Библиотека MathJax интегрировалась через React-компоненты, чтобы формулы, поступающие с сервера в формате LaTeX, автоматически преобразовывались в читаемый вид. Интерактивные графики на основе Plotly добавлялись через отдельные модули, которые обновлялись в реальном времени при получении новых данных. Например, при решении уравнения график функции перерисовывался без перезагрузки страницы, что достигалось за счёт асинхронных запросов к API.

Работа с состоянием приложения управлялась через Redux, что упростило синхронизацию данных между компонентами. Например, при отправке задачи на сервер статус её проверки (в процессе, завершено, ошибка) отображался одновременно в нескольких частях интерфейса. Обработка пользовательского ввода, включая валидацию математических выражений, выполнялась на клиентской стороне: некорректные символы блокировались, а подсказки с примерами форматирования появлялись при фокусировке на поле ввода.

Адаптивная вёрстка тестировалась через инструменты разработчика в браузере, а также на реальных устройствах — от планшетов до смартфонов с небольшими экранами. Для оптимизации загрузки ресурсов все стили и скрипты объединялись с помощью Webpack, что сократило количество HTTP-запросов и улучшило производительность. Изображения и шрифты сжимались без потери качества, а ленивая загрузка (lazy loading) графиков и формул уменьшила время первоначального рендеринга.

Тестирование интерфейса включало проверку на совместимость с разными браузерами (Chrome, Firefox, Safari) и обработку крайних сценариев, таких как обрыв соединения во время решения задачи. Для этого использовались инструменты вроде React Testing Library, а также ручные тесты, имитирующие медленное интернет-соединение через Chrome DevTools. В итоге фронтенд стал не просто оболочкой для ввода данных, а интерактивной средой, где каждый элемент работает согласованно — от момента ввода формулы до отображения подробного решения с графиками и пояснениями.

4.4 Инсталляция системы

До начала разработки приложения необходимо определится с текстовым редактором. Самые популярные IDE для разработки: VSCode, Notepad++, Sublime, PyCharm, IntelliJ. Каждый из них имеет плюсы и минусы, какие-то из них специализируются на каком-то определенном языке, например IntelliJ часто используется для android разработки, Pycharm используется для профилировании кода с использованием научных инструментов. В данной разработке использовался Visual Studio Code так как имеет больше всего плагинов, имеет встроенный Ai ассистент, может работать с плагинами Docker, ESLint, TypeScript.

Для начала работы необходимо скачать программу, она распространяется на бесплатной основе на официальном сайте, рисунок 6.

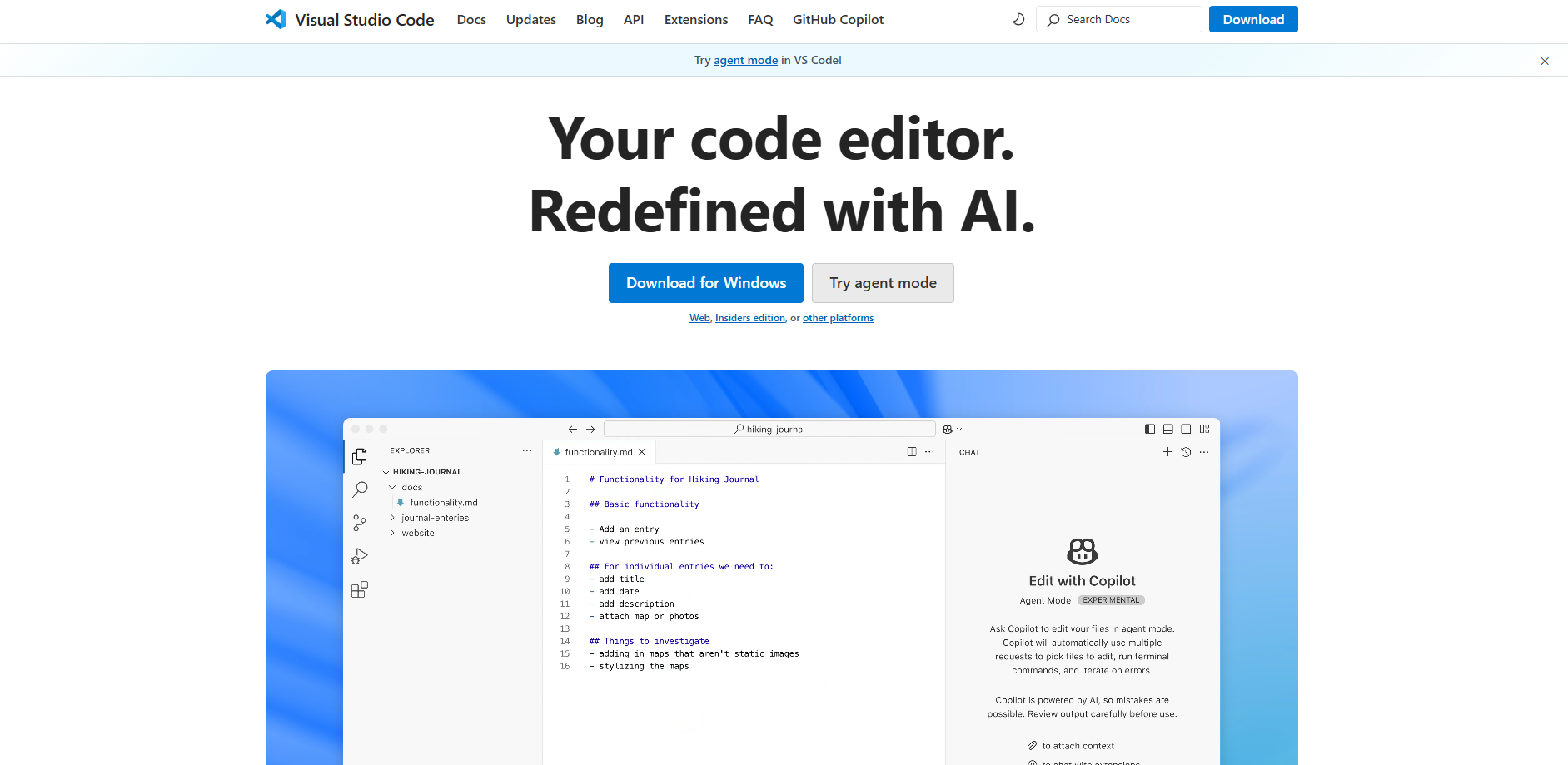


Рисунок 6 – страница скачивания VSCode

Далее следует установить Python. Его также можно скачать с официального сайта, сразу же на главной странице можно ознакомится с последними обновлениями и в чем они заключаются, рисунок 7.

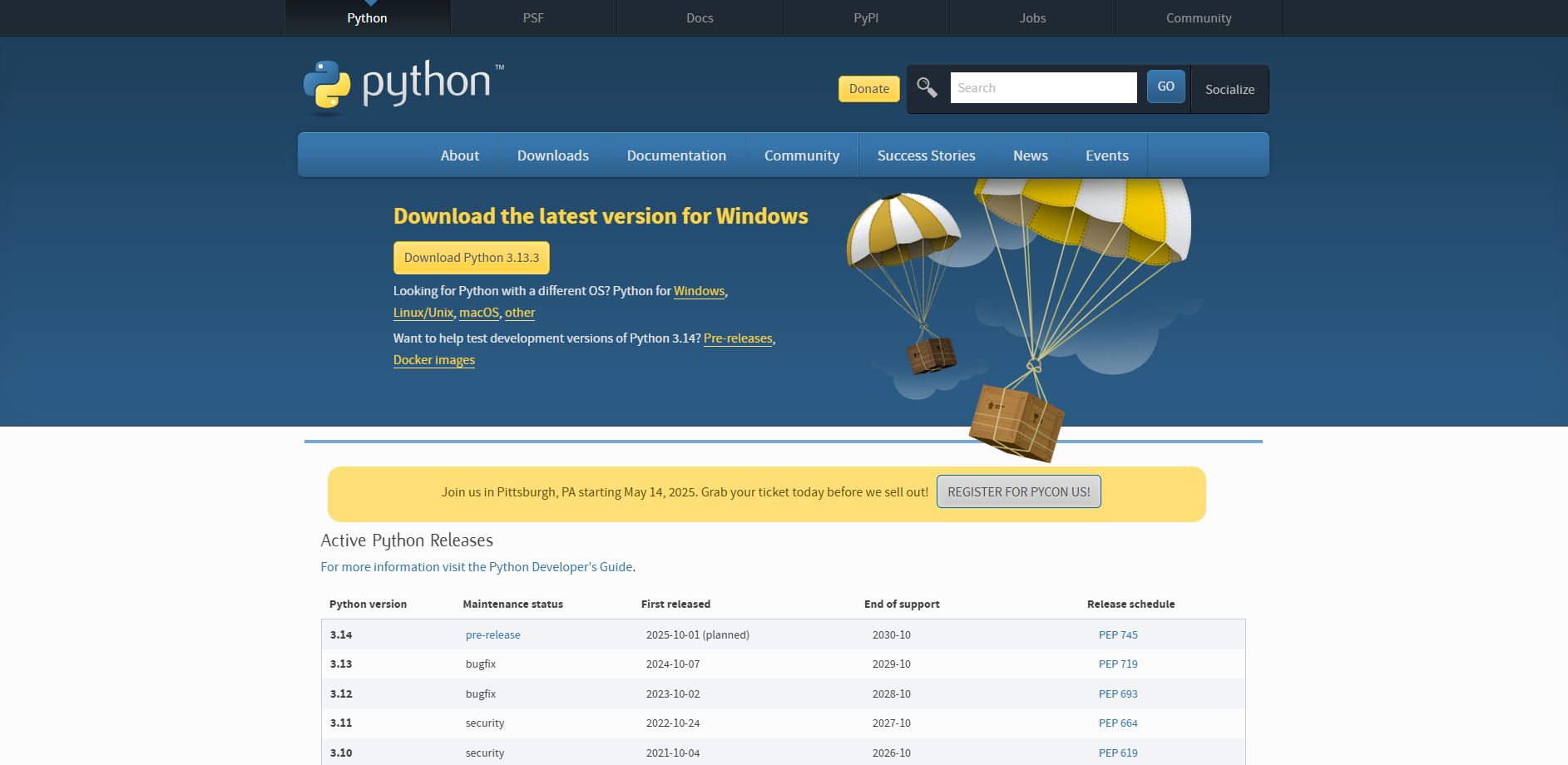


Рисунок 7 – страница скачивания Python.

В дальнейшем приложение будет использоваться и дополнятся, возможно состав программистов сменится, чтобы не возникало разногласий в версиях используемого программного обеспечения создается виртуальное окружение. Далее мы устанавливаем в это окружение Django, создаем там свой проект, его называем Project-site, желательно не использовать кириллицу так как в дальнейшем могут появится ошибки на этом фоне, Рисунок 8.

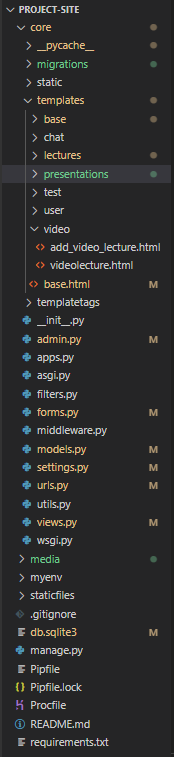


Рисунок 8 – структура проекта Project-site.

Django использует MVT модель, это аббревиатура означающая связь Модель-Вью-Темплейт, все строится на ней. Модель описывает структуру данных и взаимодействие с базой данных, Вью или представление обрабатывает логику приложения, принимает запрос взаимодействует с моделью и возвращает ответ. Темплейт или же шаблон Html файлы с динамическим контентом. Также там существует файл urls для придания путей, он связывает Url с представлением, чтобы отображать содержимое. Работает по такому принципу – Запрос пользователя, обработка и поиск подходящего пути в urls и вызывает соответствующую функцию из представления, представление взаимодействует с моделью, может получать данные из бд как пример, и передает их в шаблон.

Как только разобрались с скелетом системы можно приступать к разработке.

4.5 Интерфейсы пользователей

4.5.1 Запуск приложения

При запуске приложения происходит проверка на аутентификацию пользователя, если пользователь вошел его перебрасывает на главную страницу, на страницу приветствия, в противном случае он будет перенаправлен на страницу регистрации. Страница регистрации представляет из себя обычный регистрационный модуль куда можно ввести Логин, почту, пароль, имя и фамилию, эти данные нужны преподавателю, чтобы отслеживать деятельность студента, рисунок 9.

На главной странице можно увидеть на название проекта, логотип и представленный функционал.

На странице лекционный материал можно увидеть лекции необходимые обучающемуся, если пользователь пошел под аккаунтом администратора ему будет доступны кнопки добавить лекцию, редактировать лекцию, удалить лекцию.

На странице тестирования можно пройти тест, каждый из тестов защищен паролем для защиты от досрочного решения или использования интернета. Таким образом студент получает доступ к тесту только в то время в которое он должен сдавать его.

На странице видеоуроки можно увидеть видео, их можно добавлять с помощью ссылок в форме, приложение создает id для каждого видео и выводит их через тег iframe.

На странице презентации также можно манипулировать с формами при наличии доступа к админ привилегиям.

4.6 Написание функций

Изначально при разработке плана выявилась необходимость в авторизации, так как система не должна быть статичной, должна быть возможность добавлять новый материал и при этом ограничить доступ к этому функционалу рядовым пользователям.

Для этого была разработана базовая аутентификация, логин, регистрация, логаут.

Позже нужно было добавить материал и в следствии этого появилась функции Create.

Этот материал в следствии некоторых ошибок было необходимо изменить, появились функции Update.   
В следствии тестирования выявилась большая нагрузка на систему в следствии загрузки больших объемов данных, а также ошибки в следствии несовместимости версий файлов и из-за этого внеслись некоторые ограничения.

1. Результат