

## СОДЕРЖАНИЕ

<b>СОДЕРЖАНИЕ.....</b>	<b>1</b>
<b>ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....</b>	<b>3</b>
<b>ВВЕДЕНИЕ.....</b>	<b>5</b>
<b>1 Теоретический обзор.....</b>	<b>7</b>
1.1 Обзор предметной области.....	7
1.2 Обзор существующих решений по поиску стажировок.....	10
1.2.1 Headhunter [1].....	10
1.2.2 FutureToday [2].....	11
1.2.3 Jobby.AI [3].....	11
1.2.4 Superjob Students [4].....	12
1.2.5 Выводы по обзору платформ для поиска стажировок.....	12
1.3 Вывод по первой главе.....	14
<b>2 Проектирование приложения.....</b>	<b>15</b>
2.1 Функциональные и нефункциональные требования.....	15
2.1.1 Авторизация и регистрация.....	15
2.1.2 Стажировки.....	15
2.1.3 Тесты.....	16
2.1.4 Компании.....	16
2.1.5 Тренды.....	17
2.1.6 Профиль.....	17
2.2 Проектирование базы данных.....	19
2.2.1 Обзор и выбор базы данных.....	19
2.2.2 Создание структуры и определение таблиц.....	21
2.3 Проектирование серверной части.....	23
2.3.1 Обзор и выбор архитектуры серверной части приложения.....	24
2.3.2 Обзор и выбор языка программирования для реализации серверной части приложения.....	25
2.3.3 Обзор и выбор фреймворка для создания приложения по выбранному языку программирования.....	27
2.4 Проектирование клиентской части.....	29
2.4.1 Обзор и выбор архитектурного подхода для разработки клиентской части приложения.....	29
2.4.2 Обзор и выбор языка программирования для клиентской части приложения.....	32
2.4.3 Обзор и выбор фреймворка для клиентской части приложения...	33

2.4.4 Макет приложения.....	35
2.5 Вывод по второй главе.....	38
<b>3 Реализация приложения.....</b>	<b>39</b>
3.1 Разработка серверной части приложения.....	39
3.1.1 Реализация базы данных.....	39
3.1.2 Реализация функций взаимодействия со стажировками.....	42
3.1.3 Реализация функций взаимодействия с тестированием.....	44
3.1.4 Реализация функций взаимодействия с компаниями.....	47
3.1.5 Реализация функций взаимодействия с профилем.....	49
3.1.6 Реализация функций для аутентификации пользователя.....	50
3.2 Разработка клиентской части приложения.....	51
3.3 Разработка сервиса для сбора и анализа данных с платформы HeadHunter.....	55
3.4 Вывод по третьей главе.....	57
<b>ЗАКЛЮЧЕНИЕ.....</b>	<b>58</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....</b>	<b>59</b>

## ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Аутентификация - это процесс установления подлинности пользователя.

Авторизация - процесс предоставления прав доступа пользователю к определенному ресурсу.

Верифицировать - подтвердить достоверность информации, документа или данных.

CSRF-токен - (Cross-Site Request Forgery token) - специальный токен, используемый для защиты от атак CSRF (межсайтовая подделка запросов).

БД (База данных) - система для хранения и управления данными.

Мониторинг - процесс наблюдения за состоянием системы, приложения или процесса. Используется для выявления проблем и предотвращения сбоев.

Хостинг - предоставление доступа к серверу для размещения сайтов, приложений и других данных.

Хэширование - преобразование данных в набор символов фиксированной длины (хэш). Используется для проверки целостности данных, аутентификации и шифрования.

Ресурс - любой объект, доступный для использования в системе, например файл, страница сайта, база данных.

Даталогическая модель (Data model) - описание структуры данных, их взаимосвязей в базе данных.

Фреймворк (Framework) - структура для разработки программного обеспечения, предоставляющая набор инструментов.

Model-View-Controller (MVC) - архитектурный шаблон, разделяющий разработку веб-приложения на модель (данные в приложении), представление (интерфейс приложения) и контроллер (логика приложения).

HTML (HyperText Markup Language) - язык разметки документов для веб-страниц.

Актуализация - обновление информации, данных или системы до актуального состояния.

Стек технологий - совокупность технологий, используемых для разработки и поддержки программного обеспечения.

Backend - серверная часть приложения, отвечающая за обработку данных, логику и взаимодействие с базами данных.

Frontend - клиентская часть приложения, отвечающая за взаимодействие с пользователем, интерфейс и отображение данных в браузере.

URL-адрес (Uniform Resource Locator) - адрес веб-ресурса в интернете.

Идентификатор - уникальный код, используемый для опознания объекта.

JSON-файл (JavaScript Object Notation) - формат представления данных в виде текста. Используется для обмена между сервером и клиентом.

## **ВВЕДЕНИЕ**

В современном мире студентам необходимо приобретать актуальные навыки и знания о новых технологиях, чтобы быть конкурентоспособными на рынке труда. Одним из способов достичь этого является прохождение стажировки в компаниях, работающих с актуальными технологиями. Однако студенты часто сталкиваются с трудностями в поиске подходящих стажировок. При поиске стажировки студенты часто отправляют заявки во многие компании. Поэтому количество тестовых заданий может быть большим. Из-за этого возникает проблема отслеживания за статусом тестового задания и крайних дат отправки.

Создание веб-приложения для поиска стажировок, в котором будет система для прохождения тестовых заданий, может стать отличным решением для данной проблемы. Веб-сайт должен учитывать профессиональные навыки студента, актуальные технологические требования компаний и содержать широкий спектр программ стажировок. Особое внимание следует уделить процессу выполнения тестового задания для прохождения на стажировку, сделав его максимально прозрачным, чтобы можно было мониторить, какие тестовые задания сейчас у кандидата на стажировку активные, а какие уже прошли проверку от компании. Также система должна иметь конструктор тестов для компании, в котором будет широкий спектр инструментов для разнообразия вариантов выполнения задач.

Такое приложение позволит:

- Обеспечить более точное соответствие между потребностями студентов и ожиданиями работодателей в области стажировок.
- Снизить нагрузку на студентов, связанную с загрузкой своих решений на различные платформы, каждая из которых имеет свой интерфейс, требования и сроки. То есть, система централизует данный процесс, обеспечив мониторинг статуса тестового задания.

Целью моей дипломной работы является централизация процесса прохождения тестовых заданий для стажировок. Для этого будет создано веб-приложение для поиска стажировок в различных компаниях с возможностью прохождения тестовых заданий от компаний. Это позволит студентам проходить стандартизированные тесты, результаты которых будут доступны принимающим компаниям.

Для достижения поставленной цели были сформулированы следующие задачи:

- 1) Исследовать и проанализировать существующие аналоги по подбору стажировок на основе стека технологий.
- 2) Исследовать и проанализировать существующие технологии для создания веб-приложения.
- 3) Спроектировать серверную часть веб-приложения на основе выбранного языка программирования, определить структуру базы данных, описать требования по функционалу веб-приложения.
- 4) Спроектировать клиентскую часть веб-приложения на базе выбранного языка программирования, создать макеты веб-приложения на Figma.
- 5) Разработать серверную часть веб-приложения с использованием выбранного фреймворка.
- 6) Разработать клиентскую часть веб-приложения с использованием выбранного фреймворка.

Данная платформа будет актуальна для людей, которые только начинают путь разработчика, так и для тех, кто хочет поменять сферу деятельности.

## **1 Теоретический обзор**

В данной главе будет проведен обзор существующих сервисов, предоставляющих информацию о стажировках или реализующих схожую функциональность. Также будет проведен обзор алгоритмов, которые могут быть использованы для определения наиболее подходящих стажировок с учетом навыков студента.

### **1.1 Обзор предметной области**

Стажировка – это образовательная программа, предлагаемая компанией для студентов или специалистов, желающих получить практический опыт в определенной сфере деятельности. Стажировка может быть оплачиваемой или неоплачиваемой и обычно имеет ограниченную продолжительность.

Есть еще понятие “испытательный срок”. Испытательный срок – это период, в течение которого работодатель оценивает навыки нового сотрудника занимаемой должности. Испытательный срок устанавливается в рамках трудового договора и предполагает выполнение реальных рабочих задач.

Стажировка и испытательный срок — это разные этапы профессионального пути, каждый из которых имеет свои особенности и преимущества.

Преимущества стажировки относительно испытательного срока:

- Фокус на обучении: Главная цель стажировки — обучение и развитие. Стажеры получают возможность познакомиться с компанией, ее процессами, а также получить практические навыки под руководством специалистов.
- Меньше ответственности и давления: Стажеры не несут полной ответственности за результаты своей работы, как сотрудники на испытательном сроке. Это позволяет им учиться, экспериментировать, задавать вопросы без страха совершения ошибки.

- Возможность попробовать себя в разных ролях: Некоторые программы стажировок предусматривают систему, по которой можно попробовать себя в различных позициях. Это позволяет стажерам получить более широкое представление о разных сферах деятельности.
- Нет жестких обязательств: Стажировка не предполагает обязательств по трудоустройству. Стажер может отказаться от предложения о работе, а компания – не продлевать сотрудничество после окончания стажировки.
- Снижение рисков для обеих сторон: Стажировка позволяет компании оценить навыки кандидата без рисков, связанных с наймом. Стажер, в свою очередь, может "примерить" на себя профессию и компанию, прежде чем принимать решение о дальнейшей работе на постоянной основе.

Испытательный срок, в свою очередь, направлен на оценку соответствия сотрудника занимаемой должности и предполагает выполнение реальных рабочих задач с полной ответственностью за результат.

Таким образом, стажировка - это более гибкий и менее рискованный способ получить практический опыт и определиться с дальнейшим профессиональным путем.

Чтобы пройти на стажировку студенты в большинстве случаев проходят ряд тестирований. Есть множество способов оценки специалистов, которые уникальны для каждой компании. Но для отсева большинства кандидатов используется тестовое задание, чтобы быстро оценить минимальную компетентность, а также для дальнейшего допуска уже на более узконаправленное тестирование.

Тестовое задание — это задача или набор задач, которые предлагаются кандидату на стажировку для оценки его профессиональных навыков, знаний и способностей.



На большинстве агрегаторов, на которых есть стажировки от малых и средних компаний, при отклике пользователя работодатель в большинстве случаев отправляет ссылку с инструкцией и техническим заданием для выполнения тестового задания. Большие компании в качестве тестового задания используют систему для решения алгоритмических задач.

В результате данного отбора решается вопрос о дальнейшем рассмотрении кандидата на прохождение стажировки в компании.

Надо учитывать, что для разных компаний требуются свои формы для проведения тестовых заданий, которые отражают реальные задачи при работе.

При выполнении тестовых заданий кандидаты могут попробовать себя в решении различных задач и при этом в любой момент отказаться от выполнения, если что-то не понравилось.

При рассмотрении тестовых заданий, которые предлагают большие компании, можно заметить, что они используют “контест” в качестве самой первой проверки на компетентность кандидата. Контест – это соревнование, в котором кандидатам предлагается решить набор задач за ограниченное время. Таким образом, компании автоматически отсеивают кандидатов, которые не набрали определенный балл в контесте.

Средние и малые компании, у которых нет большой системы для проверки написанного кода, используют более упрощенный вариант для проведения тестового задания. В первом случае, это обычный тест на знание технологий, то есть, примитивный тест, где нет никакого функционала, а просто выбор из 4 вариантов. Во втором случае, это техническое задание с инструкцией по выполнению, где нужно отправить выполненное кандидатом задание (это может быть код, либо какой-нибудь документ с описанием реализации) на почту, либо в систему хранения данных.

Таким образом, у каждой компании есть своя система для проведения тестового задания без каких-либо стандартизированных условий. Кандидату, для мониторинга всех тестовых заданий, необходимо запоминать что и где находится. Из этого возникает проблема раздробленности процесса выполнения тестовых заданий.

Для решения данной проблемы я предлагаю систему, которая имеет удобный функционал для поиска стажировок, а также, чтобы при выборе стажировки была возможность сразу пройти тестовое задание и мониторить статус проверки выполнения.

## **1.2 Обзор существующих решений по поиску стажировок**

Рассмотрим несколько приложения, которые имеют функционал по поиску стажировок в различных компаниях.

### **1.2.1 Headhunter [1]**

HeadHunter - платформа для поиска работы и подбора персонала. Она предоставляет возможность работодателям эффективно находить потенциальных сотрудников, а также помогает соискателям в поиске подходящих вакансий. У сайта есть функционал по поиску стажировок, то есть веб-сайт не фокусируется на поиске стажировок, а имеет возможность отображать доступные стажировки, которые предлагают компании.

Веб-сайт предлагает хорошую фильтрацию при поиске стажировок. Можно отметить в фильтре пункт “стажировка”, тогда будут показываться только актуальные стажировки, а также система будет показывать в первую очередь те стажировки, которые больше подходят пользователю. Указать профессиональные навыки можно в профиле пользователя.

Что касается прохождения тестового задания, то тут полностью отсутствует функционал прохождения тестовых заданий от компаний, но стоит отметить, что сайт предоставляет чат для общения с менеджерами по отбору специалистов, которые отправляют ссылку на систему, где можно

проходить тестовое задание. В большинстве случаев это обычная файловая система, где нужно отправлять свое выполненное задание.

### 1.2.2 FutureToday [2].

FutureToday - платформа для поиска программ стажировок, а также различных школ для профессиональной подготовки. У сайта есть множество полезных функций для подбора стажировок, а также есть функционал для работодателей:

- Поиск программ стажировок: Пользователи могут искать актуальные стажировки на различные компании
- Подробная информация: На сайте можно выделить оформление программ по стажировкам и различным олимпиадам, каждая страница является своеобразной рекламной брошюрой с подробным описанием всех направлений стажировок.
- Рассылка: В FutureToday можно подписаться на рассылку актуальной информации по различным стажировочным программам.
- Рейтинг работодателей: На платформе есть рейтинг работодателей, который актуализируется на основе голосов студентов старших курсов, тем самым мы можем рассмотреть тенденцию рынка работодателей.

У веб-сайта отсутствует фильтрация по выбранному стеку технологий пользователя. А также нет системы для прохождения тестового задания, а также для большинства стажировок нужно переходить на сайт компании и там подавать заявку на прохождение стажировки.

### 1.2.3 Jobby.AI [3].

Jobby - это платформа для поиска вакансий с учетом навыков, сайт предоставляет широкий спектр программ стажировок в различных

компаниях. Также у веб-сайта есть функционал для поиска стажировок по выбранным технологиям и возможность указать, какими профессиональными навыками обладает пользователь.

Веб-сайт не предоставляет функционала для прохождения тестовых заданий. Но стоит отметить, что есть возможность напрямую отправить заявку для прохождения стажировки.

#### 1.2.4 Superjob Students [4].

Superjob - это платформа для поиска работы и подбора персонала, но также сайт обладает функциями для поиска стажировок. Данный сайт не акцентируется на поиске стажировок, а дает возможность работодателем публиковать доступные их компанией стажировки.

Платформа имеет определенный фильтр для отображения стажировок. Что касается функционала по прохождению тестового задания, то она отсутствует. Пользователи могут напрямую откликнуться по предложениям на стажировку и ждать обратной связи. После этого менеджер по подбору специалистов отправляет ссылку на тестовое задание, которое нужно пройти кандидату.

#### 1.2.5 Выводы по обзору платформ для поиска стажировок.

Я рассмотрел несколько самых популярных агрегаторов вакансий в России, которые имеют функционал для поиска стажировок. Исходя из обзора, можно понять, что все рассмотренные веб-сайты не имеют функционала для прохождения тестового задания от компаний. Каждый из них имеет функционал для общения между менеджерами по подбору персонала и пользователями, чтобы в дальнейшем установить связь и провести отбор на стажировку в своей системе.

Чтобы удобнее было рассматривать характеристику между существующими решениями для поиска стажировок, я приведу сравнительную характеристику, которая представлена на таблице 1.1.

Основные критерии:

- Есть функционал по поиску стажировок
- Функционал для проведения тестового задания
- Фильтрация стажировок по профессиональным навыкам
- Мониторинг статуса выполнения тестового задания

Таблица 1.1 - Сравнительная характеристика аналогов

Критерий	HeadHunter	FutureToday	JobbyAI	SuperJob
Есть функционал по поиску стажировок	Да	Да	Да	Да
Функционал для проведения тестового задания	Нет	Нет	Нет	Нет
Фильтрация стажировок по профессиональным навыкам	Да	Нет	Да	Нет
Мониторинг статуса выполнения тестового задания	Нет	Нет	Нет	Нет

Исходя из данной сравнительной характеристики, можно сделать, что реализация системы, которая будет включать в себя функционал по прохождению тестовых заданий будет конкурентоспособна среди данных аналогов.

### **1.3 Вывод по первой главе**

В данной главе была рассмотрена предметная область, а также обзор существующих решений по поиску стажировок. Был изучен функционал каждой платформы, а также выделен недостаток в виде системы для прохождения тестового задания, которая является индивидуальной для каждой компании. Из этого вытекает проблема при отслеживании кандидатом каждой отправленной заявки на прохождение стажировки. При получении обратной связи компания отправляет ссылку, где находится тестовое задание. Зачастую, при поиске стажировок кандидат отправляет множество заявок в различные компании, поэтому мониторить время на выполнение, а также место куда нужно отправлять выполненное задание нужно вручную запоминать или сохранять. На основе проведенного обзора можно сделать вывод об актуальности создания веб-приложения для поиска стажировок с упором на систему для выполнения тестового задания.

## **2 Проектирование приложения**

В данной главе будут описаны этапы к проектированию приложения. Будет создана структура приложения по которой будет реализована разработка. А также будут описаны функциональные и нефункциональные требования к приложению.

### **2.1 Функциональные и нефункциональные требования**

Для разработки веб-приложения распишем функциональные и нефункциональные требования.

#### **2.1.1 Авторизация и регистрация**

- 1) Система должна авторизовать пользователя при успешном прохождении аутентификации
- 2) Система должна поддерживать аутентификацию по email и паролю
- 3) Система должна поддерживать смену пароля авторизованного пользователя
- 4) Система должна поддерживать восстановление пароля пользователя
- 5) Система должна поддерживать выход из системы
- 6) Система должна поддерживать регистрацию пользователя по email и паролю
- 7) При регистрации система должна верифицировать почту пользователя
- 8) Система должна поддерживать уникальность email пользователей

#### **2.1.2 Стажировки**

- 1) Система должна поддерживать отображение стажировок на платформе
- 2) Система должна поддерживать фильтрацию стажировок
- 3) Система должна поддерживать поиск стажировок по ключевым словам
- 4) Система должна поддерживать отображение дополнительной информации о стажировке.

- 5) Система должна поддерживать отображение подходящих для пользователя стажировок
- 6) Система должна отображать примерное проанализированное соответствие между стажировками и навыками пользователя
- 7) Система должна поддерживать создание/редактирование/удаление стажировок

#### 2.1.3 Тесты

- 1) Система должна поддерживать прохождение тестов от компаний авторизованными пользователями
- 2) Система должна поддерживать корректное отображение тестов
- 3) Система должна сохранять результаты тестирования пользователя
- 4) Система должна поддерживать отправку тестов на проверку
- 5) Система должна поддерживать корректное отображение вопросов для теста
- 6) Система должна поддерживать создание тестов в конструкторе тестов компаниями
- 7) Система должна поддерживать редактирование/удаление тестов компаниями

#### 2.1.4 Компании

- 1) Система должна корректно отображать компании
- 2) Система должна поддерживать отображение дополнительной информации о компании
- 3) Система должна поддерживать отображение всех доступных стажировок компании
- 4) Система должна поддерживать фильтрацию стажировок компании по технологиям
- 5) Система должна поддерживать создание/редактирование/удаление компаний



### 2.1.5 Тренды

- 1) Система должна показывать отчет о самых популярных технологиях
- 2) Система должна корректно отображать анализ количества вакансий по различным ресурсам
- 3) Система должна поддерживать добавление трендов

### 2.1.6 Профиль

- 1) Система должна поддерживать редактирование профиля

Для неавторизованных пользователей будет доступна лишь страница с авторизацией и регистрацией

Для более детального ознакомления была создана диаграмма вариантов использования, рисунок 2.1.

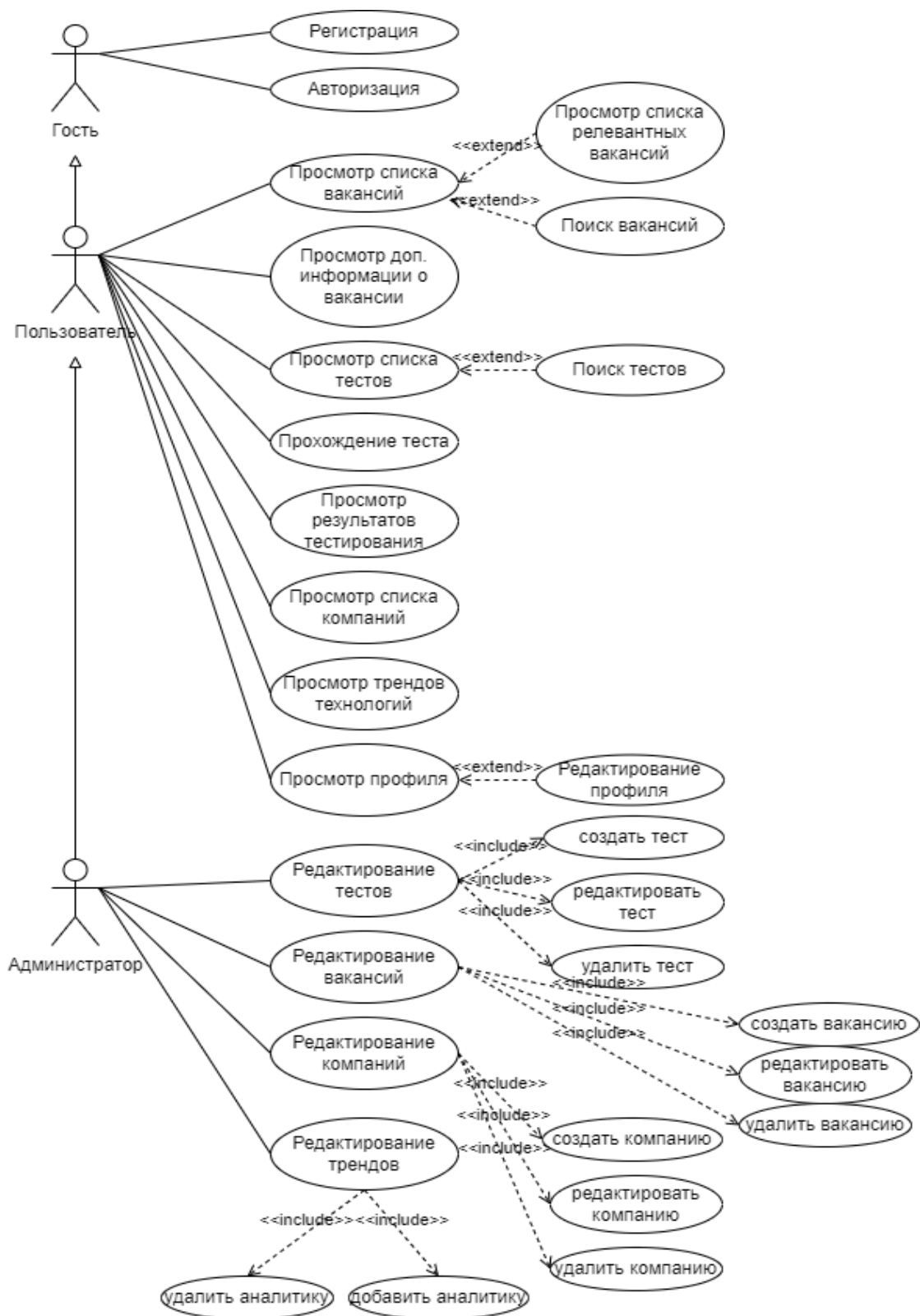


Рисунок 2.1

Также распишем нефункциональные требования для системы в целом. В требованиях будут описания свойства системы:

- **Надежность:** Будет реализован мониторинг приложения для контроля за работоспособностью и производительностью приложения.
- **Безопасность:** Использование безопасных методов аутентификации, такие как хэширование паролей и использование CSRF-токенов, шифрование данных в БД.
- **Условия эксплуатации:** Приложение будет размещено на надежном хостинге с возможностью мониторинга и управления данными.

## **2.2 Проектирование базы данных**

База данных является основным компонентом для хранения данных системы, поэтому выбор технологий для создания стабильного и защищенного хранилища является основным пунктом при проектировании. В данном этапе будет обзор и выбор базы данных, а также будет описана структура и определение таблиц.

### **2.2.1 Обзор и выбор базы данных**

#### **PostgreSQL [5]**

- База данных с открытым исходным кодом.
- **Сильные стороны:** Надежность, расширяемость, строгая поддержка SQL стандартов, богатый набор функций.
- **Слабые стороны:** Может уступать в производительности при операциях чтения.
- **Идеально подходит для:** Проектов, где важна надежность, соответствие стандартам и расширяемость, а также для работы со сложными схемами данных.

#### **MySQL [6]**

- База данных с открытым исходным кодом.

- Сильные стороны: Простота использования, высокая производительность при операциях чтения, широкая распространенность.
- Слабые стороны: Ограниченная функциональность по сравнению с PostgreSQL, некоторые отклонения от SQL стандартов.
- Идеально подходит для: Проекты, где важна скорость чтения данных и простота настройки.

### **MariaDB [7]**

- База данных с открытым исходным кодом.
- Сильные стороны: Высокая совместимость с MySQL, активная разработка, позиционируется как более производительная и функциональная альтернатива MySQL.
- Слабые стороны: Меньшая распространенность по сравнению с MySQL.
- Идеально подходит для: Проектов, использующих MySQL, но нуждающихся в расширенной функциональности или большей производительности.

### **Oracle Database [8]**

- Используется коммерческий код
- Сильные стороны: Широкий набор функций, высокая надежность, масштабируемость, отличная производительность.
- Слабые стороны: Высокая стоимость, сложность в настройке и поддержке.
- Идеально подходит для: Крупных приложений, критически важных систем, где требуется максимальная надежность и производительность.

## Microsoft SQL [9]

- Используется коммерческий код
- Сильные стороны: Широкая функциональность, хорошая интеграция с другими продуктами Microsoft, высокая производительность.
- Слабые стороны: Высокая стоимость, привязка к платформе Microsoft.
- Идеально подходит для: Проектов, разрабатываемых на платформе Microsoft, где важна интеграция с другими продуктами Microsoft.

## SQLite [10]

- База данных с открытым исходным кодом.
- Сильные стороны: Встраиваемая база данных, простота использования, не требует настройки, идеальна для локальных приложений.
- Слабые стороны: Ограниченная функциональность, не подходит для крупных проектов.
- Идеально подходит для: Встраивания в приложения, мобильной разработки, проектов с ограниченными ресурсами.

Исходя из сильных и слабых сторон каждой рассмотренной базы данных, а также из моего личного опыта при использовании некоторых рассмотренных решений, я выбрал базу данных PostgreSQL.

### 2.2.2 Создание структуры и определение таблиц

База данных состоит из 9 таблиц:

- 1) User: На данной таблице будет храниться информация о пользователя - почта пользователя, хешированный пароль

- 2) Skill: Таблица с технологиями, нужен для хранения различных технологий
- 3) Company: На этой таблице будет храниться информация о компаниях - название, адрес, почта и описание
- 4) Vacancy: Будет храниться информация о вакансиях.
- 5) Test: По каждому технологиям будет выделен тест с названием и описанием
- 6) VacancySkill: Таблица для хранения информации об вакансии и какие навыки будут нужны для конкретной вакансии
- 7) Trend: Таблица для хранения аналитики по различным технологиям
- 8) SkillUser: Промежуточная таблица для отношения многие-ко-многим, нужна для хранения информации о результатах тестирования пользователя

Для наглядного отображения была построена даталогическая модель на рисунке 2.2

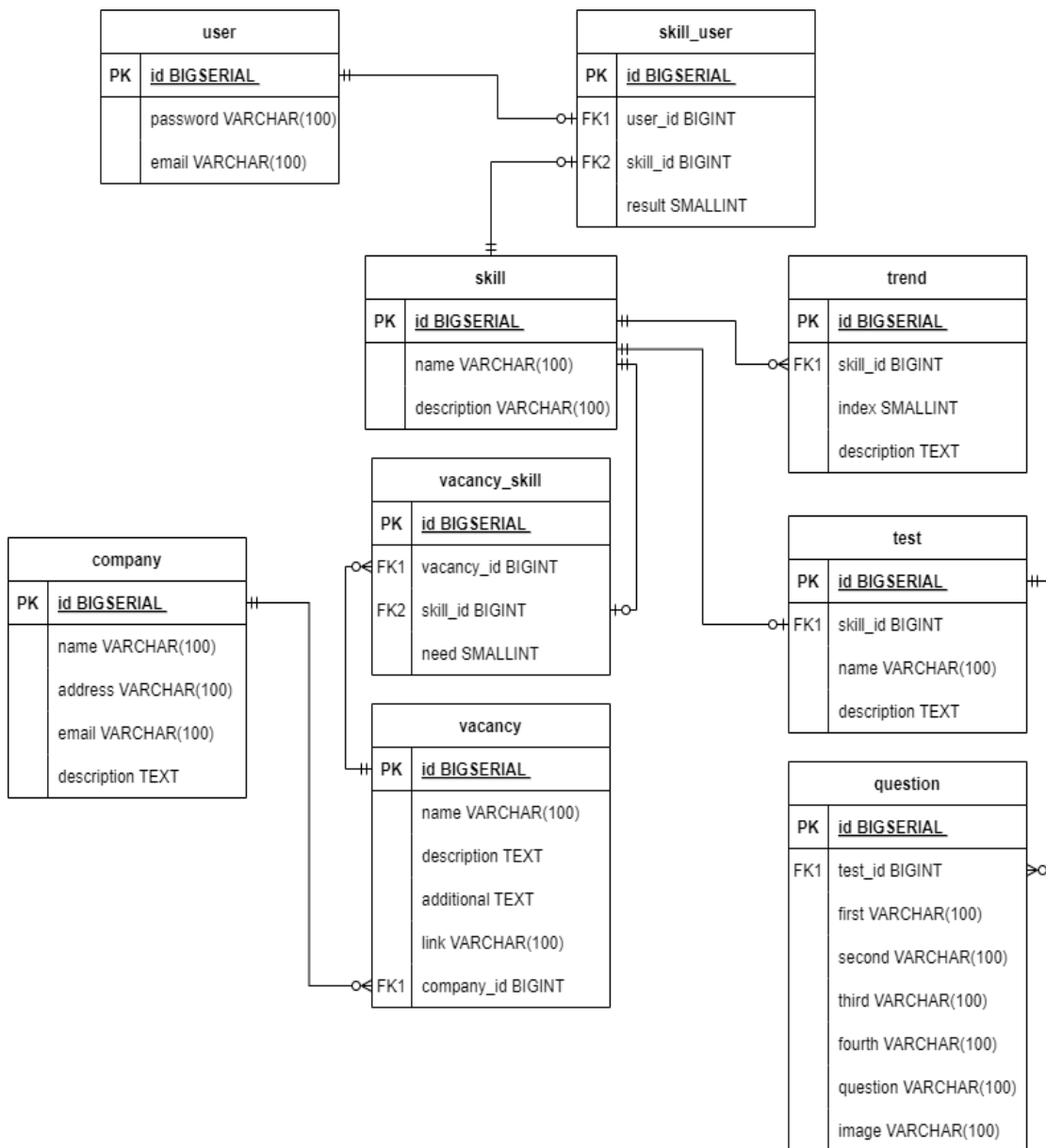


Рисунок 2.2 - даталогическая модель базы данных

### 2.3 Проектирование серверной части

Серверная часть приложения - это сторона приложения которая отвечает за процессы и функционал системы. Пользователь не видит эту часть, а взаимодействует с ним путем отправления различных запросов с клиентской части.

### 2.3.1 Обзор и выбор архитектуры серверной части приложения

1. Монолитная архитектура - весь код серверной части находится в одном приложении.
  - Преимущества: Простая для понимания и разработки. Легко разворачивается.
  - Недостатки: Сложность обслуживания и масштабирования. Изменения в одной части кода могут повлиять на работу всего приложения. Сложно внедрять новые технологии.
  - Пример: Небольшие веб-сайты, приложения с ограниченной функциональностью.
2. Микросервисная архитектура - приложение разбивается на независимые модули (микросервисы), взаимодействующие друг с другом.
  - Преимущества: Легко масштабируется, каждый микросервис может быть развернут независимо. Повышенная отказоустойчивость, сбой одного сервиса не влияет на другие. Возможность использовать разные технологии для разных сервисов.
  - Недостатки: Сложность разработки и управления. Требуется большого опыта. Более сложное тестирование.
  - Пример: Крупные приложения, системы с высокой нагрузкой, платформы электронной коммерции.
3. Serverless архитектура - код запускается на серверах, предоставляемых провайдерами облачных сервисов (AWS Lambda, Google Cloud Functions).
  - Преимущества: Высокая масштабируемость и отказоустойчивость. Оплата только за фактически



использованные ресурсы. Упрощенное развертывание и управление.

- Недостатки: Ограничение по времени выполнения функций. Сложность отладки. Зависимость от провайдера облачных сервисов.
- Пример: Обработка событий, масштабируемые API, backend для мобильных приложений.

4. API-ориентированная архитектура (API-first) - серверная часть строится вокруг API, предоставляющего доступ к функциям и данным.

- Преимущества: Упрощает взаимодействие между различными сервисами и клиентами. Позволяет создавать экосистему вокруг приложения.
- Недостатки: Требуется тщательного проектирования API.
- Пример: Публичные API, платформы для разработчиков, микросервисные приложения.

Исходя из рассмотренных архитектур серверной части приложения была выбрана монолитная архитектура для приложения, основным моментом стало простота разработки, а также ненужность создания маленьких модулей, так как система будет одна и небольшая. То есть основной упор будет сделан на создание системы тестирования, поэтому основной код будет написан для данного раздела приложения.

### 2.3.2 Обзор и выбор языка программирования для реализации серверной части приложения

#### 1. Python [11]

- Преимущества: Простой синтаксис, легко изучить. Обширная экосистема библиотек и фреймворков (Django, Flask). Много доступной документации.
- Недостатки: Может уступать в производительности другим языкам.
- Подходит для: Быстрой разработки, прототипирования, машинного обучения, анализа данных.

## 2. Java [12]

- Преимущества: Стабильность, надежность, масштабируемость. Строгая типизация, хорошо подходит для крупных проектов. Огромное количество библиотек и фреймворков (Spring).
- Недостатки: Более сложный синтаксис, разработка может быть медленнее.
- Подходит для: Enterprise-приложений, сложных систем, проектов, требующих высокой надежности.

## 3. PHP [13]

- Преимущества: Широкая распространенность, большое количество хостингов. Простота в изучении, быстрая разработка. Популярные фреймворки (Laravel, Symfony).
- Недостатки: Менее строгая типизация, что может привести к ошибкам.
- Подходит для: Небольших и средних веб-сайтов, блогов, CMS.

## 4. C# [14]

- Преимущества: Высокая производительность, масштабируемость. Мощная платформа .NET, богатый набор библиотек.
- Недостатки: Привязка к платформе Microsoft Windows.
- Подходит для: Приложений, работающих на Windows.

## 5. Go [15]

- Преимущества: Высокая производительность. Простой синтаксис, легко изучить.
- Недостатки: Неразвитая система библиотек.

- Подходит для: Систем, требующих высокой производительности, сервисов, микросервисной архитектуры.

Исходя из рассмотренных языков программирования самым подходящим под мое приложение является язык Python. Этот выбор обусловлен моим личным опытом в программировании на данном языке, а также обширным количеством полезных библиотек.

### 2.3.3 Обзор и выбор фреймворка для создания приложения по выбранному языку программирования

1. Django [16] - это высокоуровневый веб-фреймворк на Python. Данный фреймворк предоставляет всё необходимое для создания полноценных и безопасных веб-приложений прямо из коробки.

- Преимущества: предоставляет всё необходимое для создания полноценных веб-приложений (ORM, админская панель). Хорошая документация. MVC архитектура, которая способствует структурированию кода. Встроенные средства безопасности.
- Недостатки: Может быть слишком громоздким для небольших проектов.

Фреймворк подходит как для больших веб-приложений, так и для малых.

2. Flask [17] - это микро-фреймворк Python для веб-разработки, который предоставляет базовые инструменты для создания веб-приложений. Фреймворк не навязывает никакую структуру при создании веб-приложений.

- Преимущества: Гибкость и минимализм, дает больше контроля над структурой приложения. Легкий в изучении, хорошо подходит для небольших проектов. Большое количество расширений.

- Недостатки: Требуется больше ручной настройки, чем Django. Меньше встроенных функций, некоторые нужно реализовывать самостоятельно.

Фреймворк подходит для разработки простых веб-приложений, микросервисов.

3. Pyramid [18] - это мощный фреймворк для создания веб-приложений. Предоставляет гибкость и модульность, таким образом, предоставляет разработчику создавать приложения с различными архитектурами.

- Преимущества: Гибкость и модульность, позволяет выбирать нужные компоненты. Хорошая производительность. Подходит для проектов любого размера.
- Недостатки: Более сложный порог входа, чем у Django или Flask.

Фреймворк подходит для разработки проектов, требующих высокой гибкости и производительности.

4. FastAPI [19] - это современный фреймворк Python для создания API, который отличается своей скоростью, простотой и функциональностью.

- Преимущества: Высокая производительность, автоматическая генерация документации. Поддержка асинхронного программирования. Встроенная валидация данных.
- Недостатки: Относительно новый фреймворк, сообщество ещё развивается.

Фреймворк подходит для создания REST API, микросервисов, приложений, где особый упор идет на производительность.

Исходя из рассмотренных фреймворков, а также моего личного опыта программирования я выбрал фреймворк Django. Основным плюсом является большой встроенный функционал, а также внутренняя MVC архитектура.

## 2.4 Проектирование клиентской части

Клиентская часть веб-приложения - это то, что видит пользователь: интерфейс, взаимодействие и т.д.

### 2.4.1 Обзор и выбор архитектурного подхода для разработки клиентской части приложения

1. Традиционная веб-разработка - страницы создаются с использованием чистого HTML [20], CSS [21], JavaScript [22]. При такой разработке запросы на серверную часть отправляются при каждом действии пользователя.

- Преимущества: Простота реализации, совместимость с браузерами.
- Недостатки: Низкая скорость взаимодействия, сложность в реализации динамических компонентов.

2. Одностраничные приложения - весь интерфейс подключается один раз, взаимодействия с пользователем происходят через JavaScript.

- Преимущества: Высокая скорость взаимодействия. Динамические интерфейсы. Подходит для приложений с большим количеством данных.
- Недостатки: Более сложная разработка. Требуется более мощный браузер.

3. Многостраничные приложения - приложение состоит из нескольких HTML-страниц, которые загружаются, исходя из действий пользователя.

- Преимущества: Простота реализации.

- Недостатки: Низкая скорость взаимодействия, не подходит для динамических приложений.

4. Гибридные приложения - комбинирует преимущества одностраничных и многостраничных приложений.

- Преимущества: Гибкая архитектура.
- Недостатки: Сложная разработка.

Исходя из рассмотренных подходов к архитектуре клиентской части приложения я буду использовать гибридное приложение. Так как данный подход сочетает в себе преимущества многостраничного и одностраничного приложения.

Архитектура структуры кода будет на основе методологии FSD - Feature-Sliced Design. [23] Проект на FSD состоит из слоев (layers), каждый слой состоит из слайсов (slices) и каждый слайс состоит из сегментов (segments).

Слои стандартизированы и расположены вертикально. Модули на одном слое могут взаимодействовать лишь с модулями, находящимися на слоях строго ниже.

- shared — переиспользуемый код, не имеющий отношения к специфике приложения/бизнеса.
- entities (сущности) — бизнес-сущности.
- features (фичи) — взаимодействия с пользователем.
- widgets (виджеты) — композиционный слой для соединения сущностей и фич в самостоятельные блоки.
- pages (страницы) — слой для сборки полноценных страниц из сущностей, фич и виджетов.

- app — настройки для всего приложения.

Затем есть слайсы, разделяющие код по предметной области. Они группируют логически связанные модули, что облегчает навигацию по кодовой базе. Слайсы не могут использовать другие слайсы на том же слое, что обеспечивает высокий уровень связности (cohesion) при низком уровне зацепления (coupling).

В свою очередь, каждый слайс состоит из сегментов. Это маленькие модули, главная задача которых — выделить код внутри слайса по техническому назначению. Схема модели FSD приведена на рисунке 2.4

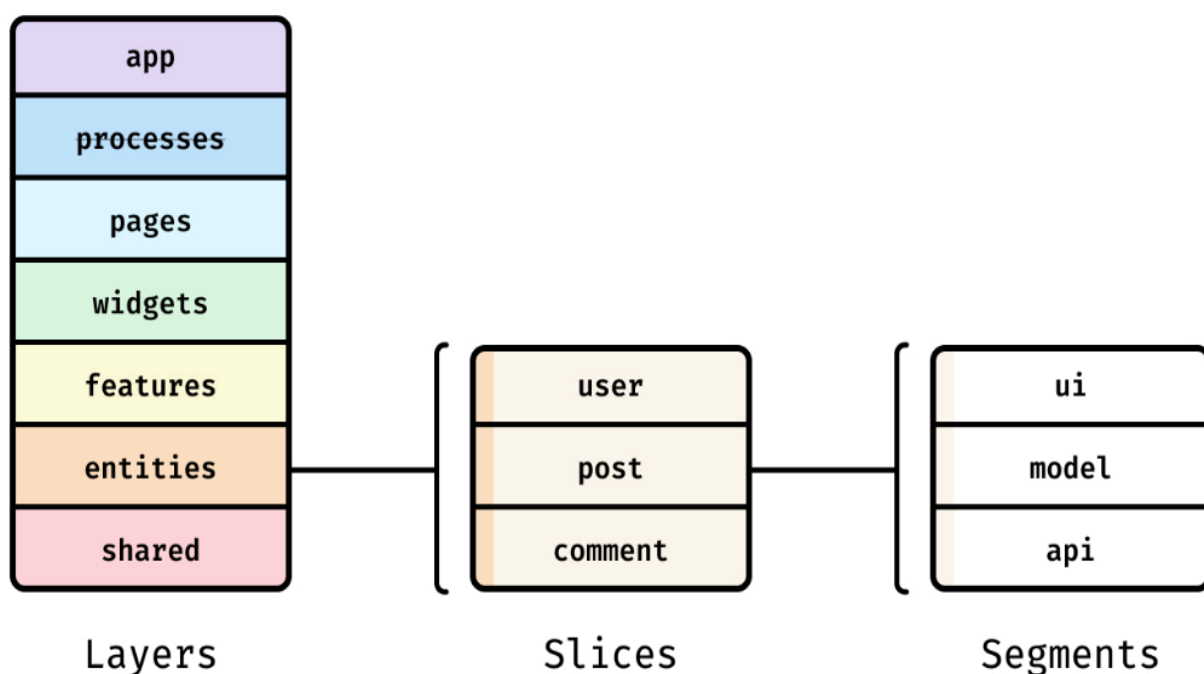


Рисунок 2.4 - методология FSD

Преимущества данной архитектуры:

- Единообразие: Код распределяется по слоям, предметной области и техническому назначению. Благодаря этому архитектура стандартизируется и становится более простой.
- Переиспользование логики: Каждый компонент архитектуры имеет свое назначение и список зависимостей. Благодаря этому появляется возможность адаптировать модуль под разные цели.

- Устойчивость к изменениям: Один модуль не может использовать другой модуль, расположенный на том же слое или на слоях выше. Благодаря этому компоненты изолированы.

## 2.4.2 Обзор и выбор языка программирования для клиентской части приложения

### 1. JavaScript:

- Преимущества: Фактический стандарт для веб-разработки. Широко распространен, большое сообщество, много ресурсов. Возможность создавать как простые, так и сложные интерфейсы. Поддерживает объектно-ориентированное, функциональное программирование. Много библиотек и фреймворков (React, Angular, Vue.js) для упрощения разработки.
- Недостатки: Может быть сложным для новичков. Не всегда легко отлаживать.

Язык подходит для любых веб-приложений.

### 2. TypeScript [24]:

- Преимущества: Статически типизированный язык, что упрощает отладку и делает код более надежным. Совместимость с JavaScript, можно использовать существующие библиотеки. Поддержка ООП, что делает код более структурированным. Большое количество инструментов для разработки и отладки.
- Недостатки: Может быть более сложным для новичков, чем JavaScript. Не все браузеры поддерживают TypeScript непосредственно, требуется компиляция в JavaScript.

Подходит для сложных веб-приложений, где важна надежность кода.

### 3. Dart [25]:



- Преимущества: Статически типизированный язык с поддержкой ООП. Создан Google и используется в Flutter Web. Высокая производительность и эффективность.
- Недостатки: Не так широко распространен, как JavaScript или TypeScript. Меньше библиотек и фреймворков по сравнению с JavaScript.

Подходит для разработки веб-приложений с помощью Flutter.

Исходя из рассмотренных языков программирования для разработки клиентской части, я выбрал TypeScript. Основным моментом для выбора данного языка стала хорошая типизация языка, а также совместимость с библиотеками языка JavaScript.

#### 2.4.3 Обзор и выбор фреймворка для клиентской части приложения

TypeScript — это язык программирования, который добавляет статическую типизацию к JavaScript. Это позволяет создавать более масштабируемые и устойчивые приложения. Существует множество фреймворков, которые используют TypeScript и предлагают различные преимущества:

##### 1. Angular [26]:

- Преимущества: Полностью поддерживает TypeScript. Мощный фреймворк для создания крупных и сложных веб-приложений. Имеет богатую экосистему библиотек и инструментов. Использует компонентный подход для разработки.
- Недостатки: Может быть сложным для начинающих. Довольно большой размер проекта.

##### 2. React [27]:

- Преимущества: Использует TypeScript для написания компонентов. Гибкий и легко настраиваемый. Хорошо подходит для создания интерактивных интерфейсов. Множество библиотек.
- Недостатки: Не является фреймворком, а скорее библиотекой. Может потребоваться больше ручной настройки.

### 3. Vue.js [28]:

- Преимущества: Легко изучить и использовать. Имеет прогрессивный подход к разработке. Гибкий и хорошо подходит для создания небольших и средних приложений.
- Недостатки: Не такой большой выбор библиотек, как у React или Angular.

### 4. NestJS [29]:

- Преимущества: Создан на основе Node.js и Express.js. Обеспечивает архитектурную структуру для создания масштабируемых серверных приложений. Использует концепцию зависимостей и инверсии управления (IoC).
- Недостатки: Может быть сложным для начинающих. Может потребоваться больше усилий для настройки.

### 5. Next.js [30]:

- Преимущества: Использует React для создания серверных приложений. Обеспечивает оптимизацию производительности.
- Недостатки: Может быть сложным для начинающих. Не такой гибкий, как React.

Исходя из рассмотренных фреймворков для разработки веб-приложения, а также моего личного опыта я выбрал фреймворк React, так как он очень гибкий, а также у него есть огромное количество готовых библиотек для разработки.

#### 2.4.4 Макет приложения

Для создания макета веб-приложения я использую платформу Figma. Данный веб-сайт обладает мощным инструментарием для создания прототипов приложений.

Веб-приложение будет состоять из 5 основных страниц и нескольких вспомогательных. Основной страницей является - страница с вакансиями. На нем можно увидеть функции поиска и фильтрации по различным категориям. А также на каждой карточке вакансий можно будет увидеть описание и основной стек технологий. Макет страницы представлен на рисунке 2.5.

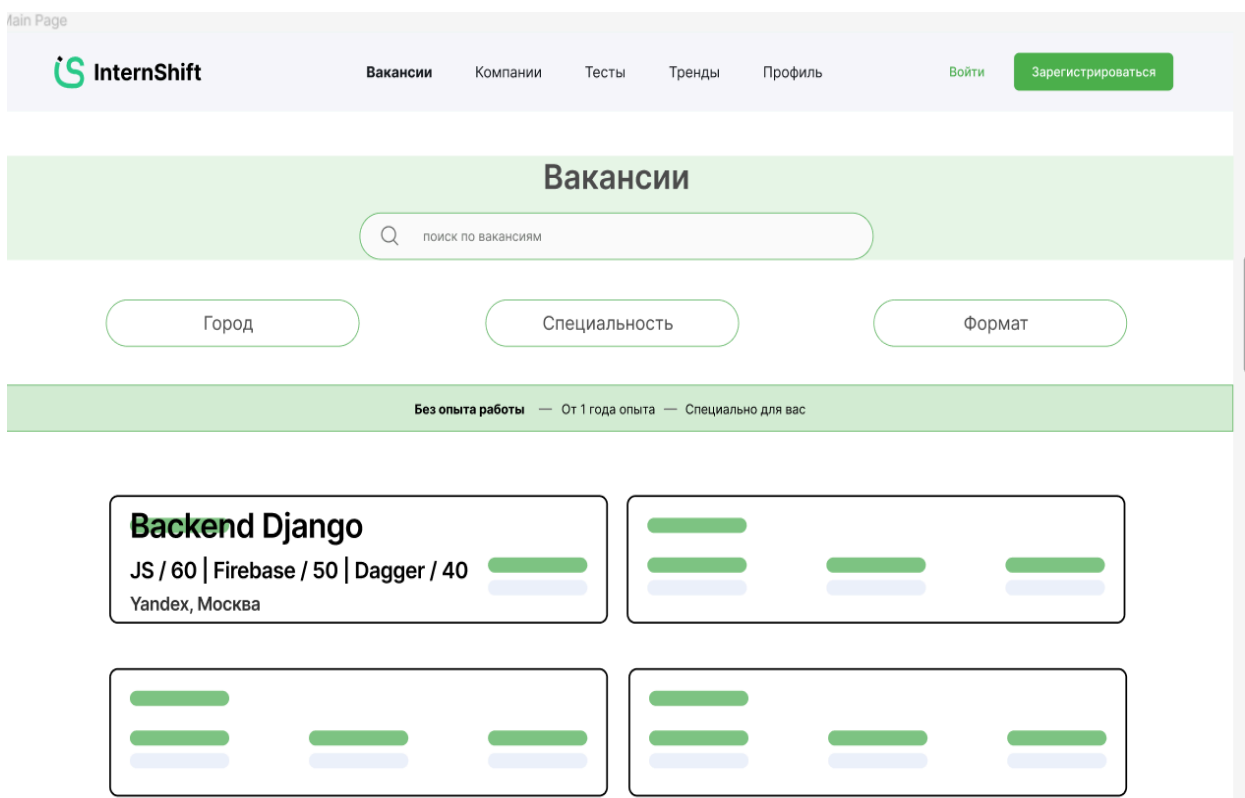


Рисунок 2.5

Страница с компаниями будет хранить информацию о компаниях, сортировка будет проводиться по количеству доступных вакансий, но будет

поиск по названиям компаний и фильтрация по различным категориям. Макет страницы приведен на рисунке 2.6.

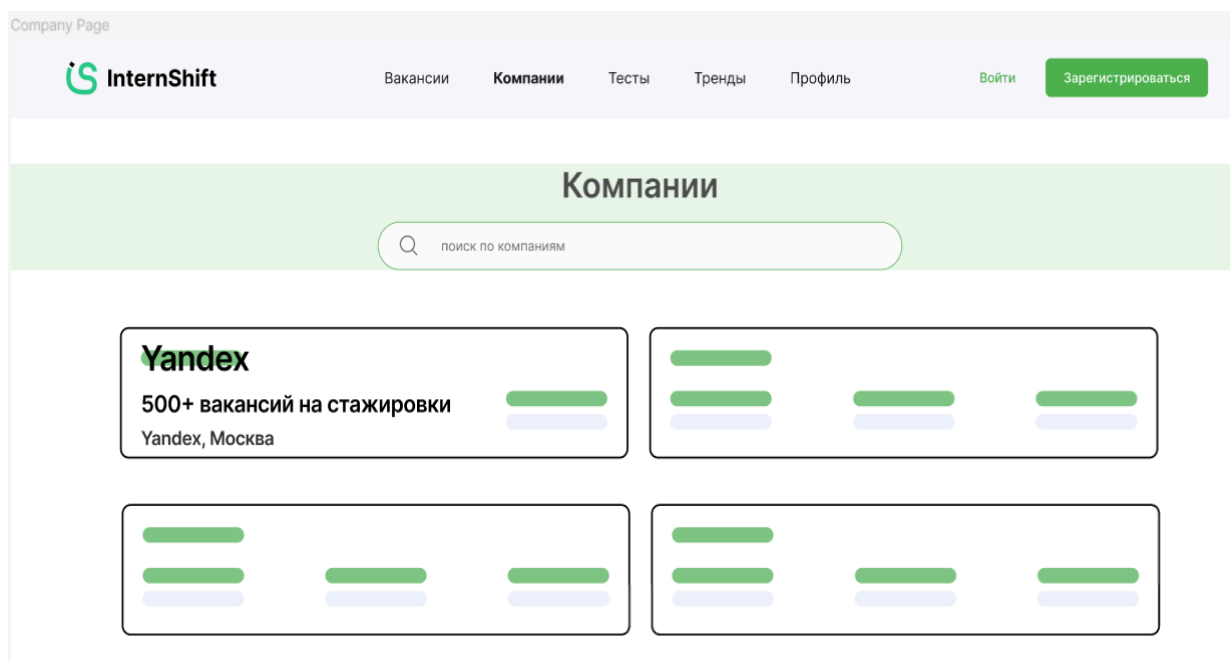


Рисунок 2.6

Страница с тестами будет содержать множество карточки с активными тестовыми заданиями. На карточке будет отражена информация о статусе прохождения теста, а также крайняя дата сдачи тестового задания. Макет страницы приведен на рисунке 2.7.

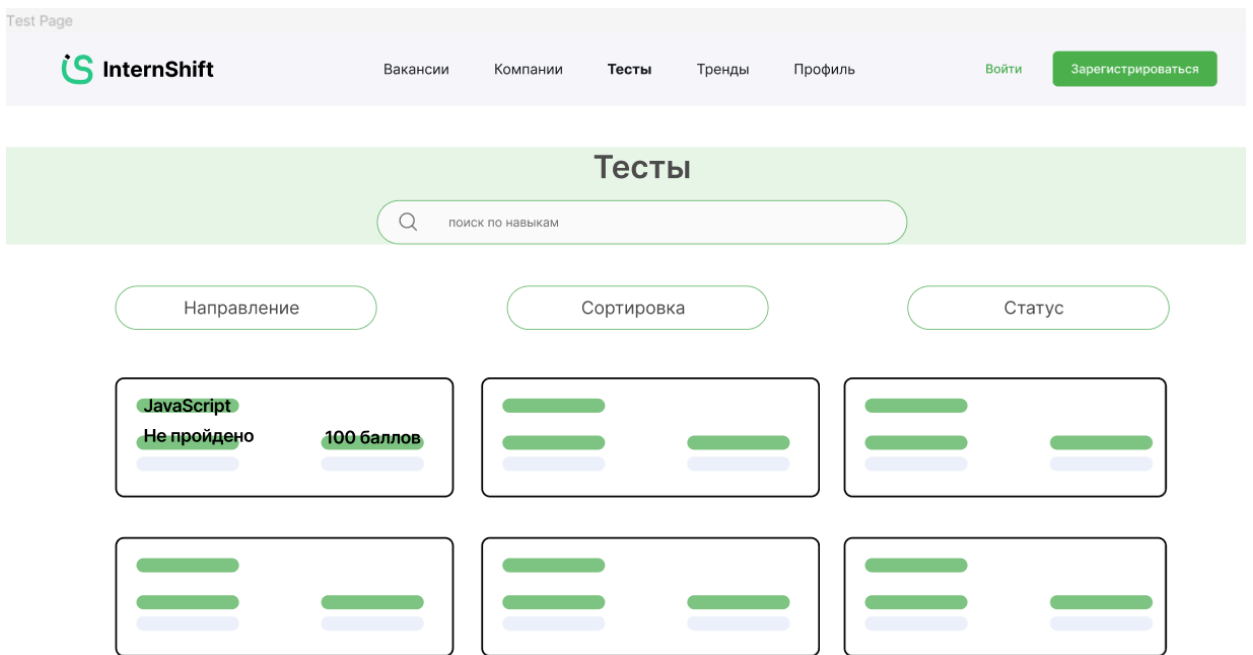


Рисунок 2.7

Страница трендов будет содержать информацию о самых популярных технологиях в различных вакансиях и самых распространенных вакансиях на рынке. Информация будет взята из самых популярных агрегаторов вакансий. Макет страницы представлен на рисунке 2.8.

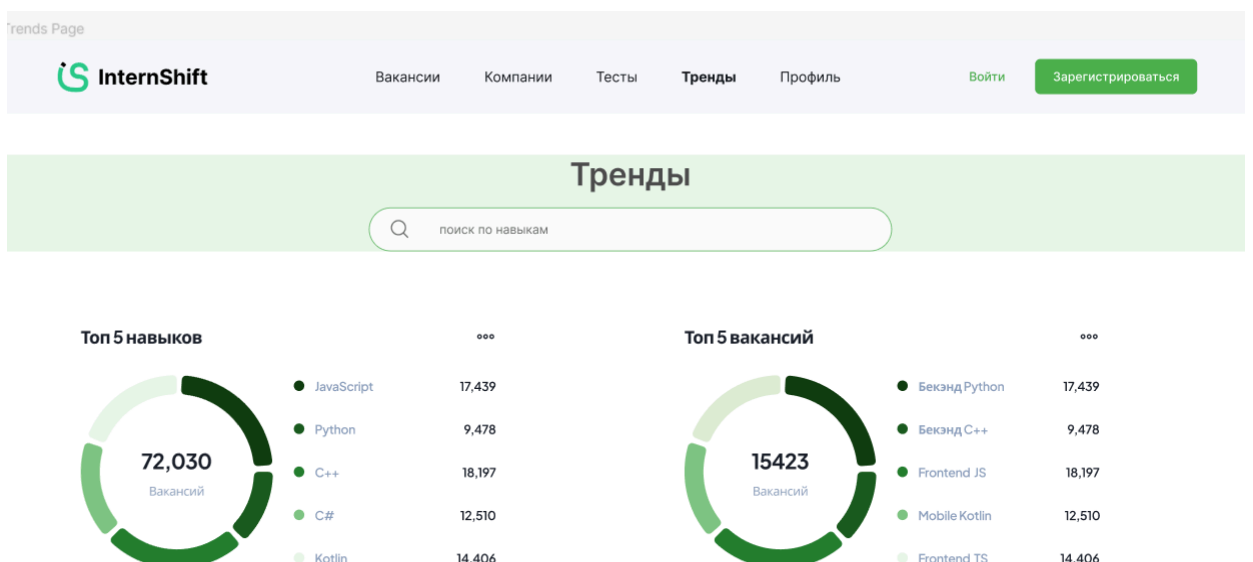


Рисунок 2.8

Страница профиля пользователя будет содержать информацию о пользователе, а также информацию о профессиональных навыках. Макет страницы представлен на рисунке 2.9.

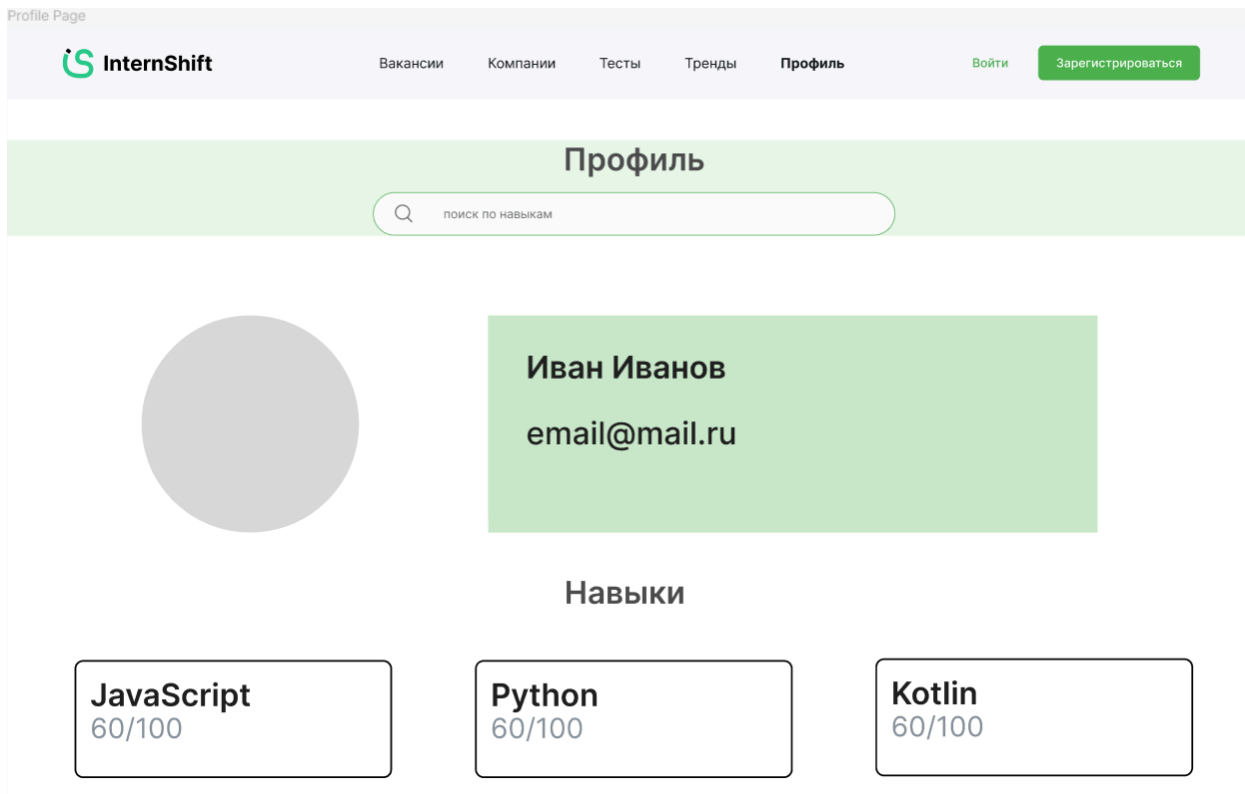


Рисунок 2.9

## 2.5 Вывод по второй главе

В данной главе было проведено проектирование всего веб-приложения. Были описаны функциональные и нефункциональные требования к веб-приложению. Исходя из функциональных требований была создана диаграмма вариантов использования, а также спроектирована структура клиентской и серверной части приложения. Для хранения данных была сформирована даталогическая модель базы данных.

### 3 Реализация приложения

В данной главе будут описаны этапы по разработке всего приложения. Глава будет разделена на 3 основных этапа: Backend, Frontend, Аналитика. По каждому этапу будет подробное описание всех основных компонентов. Рисунок кода будет только по самым важным частям приложения с подробным описанием функции всех компонентов.

#### 3.1 Разработка модуля аутентификации

Для начала реализуем модуль аутентификации. Чтобы была возможность для регистрации как компании, так и пользователя, который ищет стажировки, мы будем использовать систему ролей. Для студента будет роль - candidates, для компании - companies.

Для взаимодействия с базой данных Django предоставляет инструмент, который позволяет взаимодействовать с базой данных с помощью объектов Python вместо написания SQL-запросов напрямую. Вместо создания таблиц SQL я определяю структуру данных с помощью классов Python, называемых моделями. Каждый атрибут модели представляет собой поле в таблице базы данных. Django ORM автоматически генерирует SQL-код для создания таблиц в базе данных на основе моих моделей. Для модуля аутентификации будут использованы готовые классы User и Groups из библиотеки `django.contrib.auth.models`.

Для обеспечения безопасности данных я буду использовать токен для аутентификации пользователя. Токен, в данном контексте - это строка символов, которая подтверждает личность пользователя и предоставляет доступ к ресурсам API без необходимости повторного ввода логина и пароля при каждом запросе. На стороне клиента будет храниться только токен, который будет действителен в течении некоторого времени, при истечении

срока действия токена он будет обновлен при следующем входе. Токен генерируется на этапе регистрации пользователя.

Для отображения бизнес-процесса авторизации я приведу диаграмму последовательности на рисунке 3.1 в нотации UML.

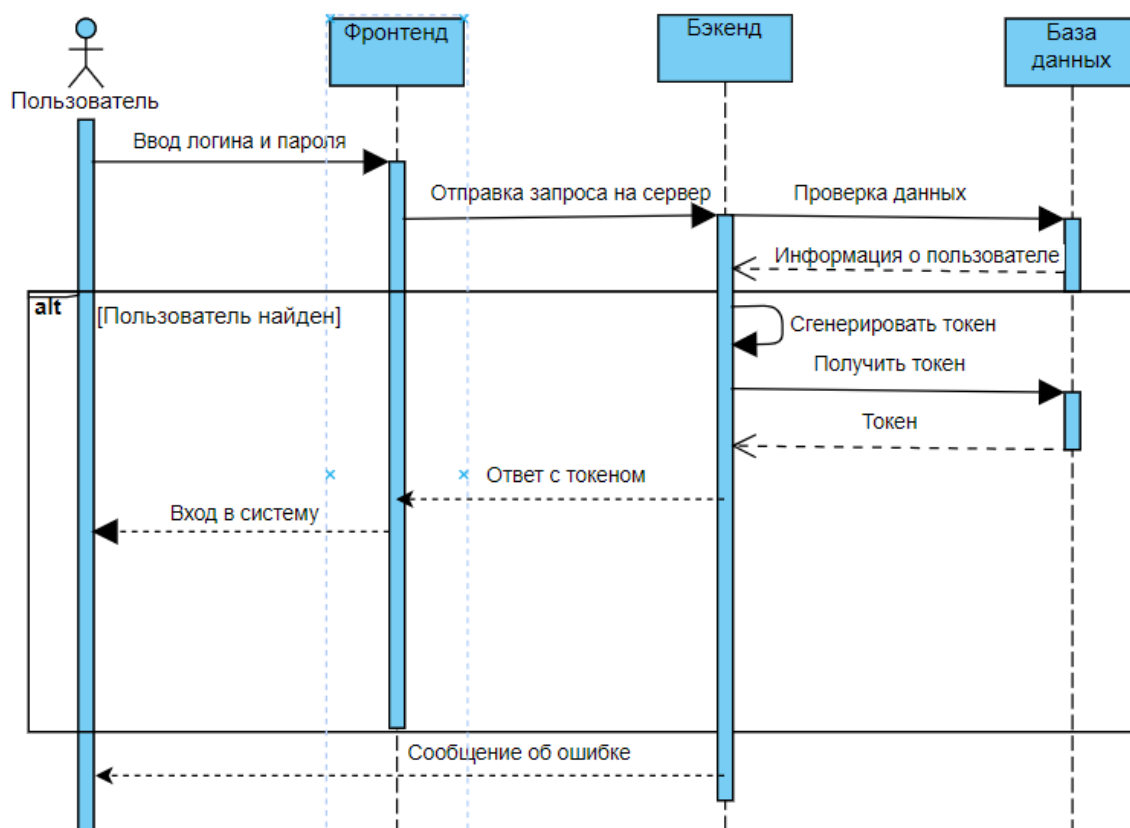
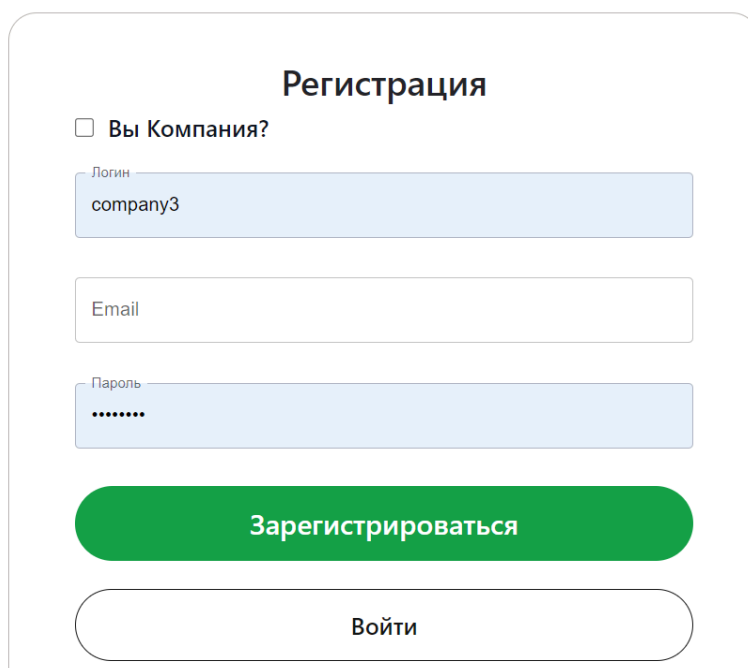


Рисунок 3.1 - Процесс авторизации пользователя

При успешной авторизации пользователя каждый последующий запрос будет отправлен с заголовком `Authorization: Token <Токен пользователя>`. При регистрации у пользователя будет на выбор чекбокс, где пользователь выбирает роль. Если человек выбирает, что он “компания”, то запрос отправиться на endpoint для регистрации компании, иначе для регистрации кандидата. Интерфейс регистрации для с возможностью выбора роли пользователя приведен на рисунке 3.2.

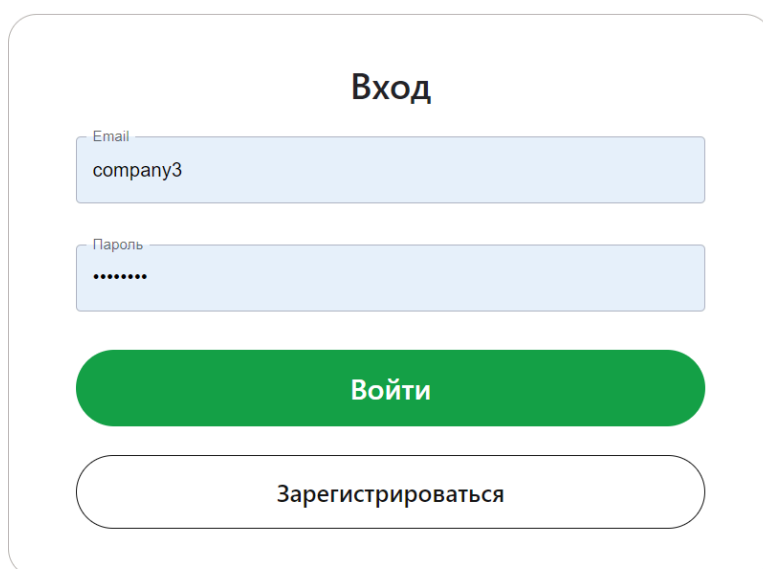




The registration form is titled "Регистрация". It includes a checkbox labeled "Вы Компания?". Below this are three input fields: "Логин" (containing "company3"), "Email", and "Пароль" (containing seven dots). At the bottom are two buttons: a green "Зарегистрироваться" button and a white "Войти" button.

Рисунок 3.2 - страница регистрации

При успешной регистрации пользователь будет переадресован на страницу входа, рисунок 3.3. После заполнения полей и нажатии на кнопку будет отправлен запрос на авторизацию пользователя.



The login form is titled "Вход". It includes two input fields: "Email" (containing "company3") and "Пароль" (containing seven dots). At the bottom are two buttons: a green "Войти" button and a white "Зарегистрироваться" button.

Рисунок 3.3 - Страница входа

Если авторизация прошла успешно, то пользователь будет переадресован на главную страницу. При последующем посещении веб-сайта будет проведена проверка на актуальность токена аутентификации. Если время жизни токена не истекло, то пользователь сразу будет переадресован на главную страницу, иначе пользователь должен пройти повторную авторизацию.

Система ролей нужна для отображения нужных компонентов на клиентской стороне. Для компаний будет функционал для создания тестов и карточек стажировок, а также для мониторинга прохождения тестовых заданий. Для кандидатов будет функционал для поиска стажировок по навыкам и прохождения тестовых заданий с мониторингом статуса выполнения.

### 3.2 Разработка модуля поиска и создания стажировок.

После успешной авторизации пользователя мы переходим на главную страницу приложения. По описанным функциональным требованиям система должна обладать функционалом для поиска, а также фильтрации стажировок по навыкам пользователя. Поэтому реализуем для начала данный функционал.

Для начала создадим модели Skill и SkillUser. В модели Skill будет поле с названием навыка, а в модели SkillUser будет реализовано отношение многие-ко-многим с таблицами User и SkillUser. Данная связь позволяет одному пользователю обладать несколькими навыками и одному навыку быть привязанным к нескольким пользователям.

Навыками в данном случае являются популярные технологии и языки программирования. Чтобы заполнить таблицу Skill языками программирования я взял 50 самых популярных языков программирования,

исходя из рейтинга TIOBE []. TIOBE формирует свой рейтинг на основе количества запросов с “выбранный язык программирования” на различных платформах. Я вручную заполнил таблицу Skill в админской панели от Django. Теперь необходимо заполнить таблицу популярными библиотеками и фреймворками, которые используют в настоящее время. Для выбора веб-фреймворков и технологий я исходил из статистик на сайте [statista.com](https://www.statista.com) [].

После заполнения данными таблицы Skill, я перехожу на разработку функционала для выбора навыков для пользователя.

Изначально при регистрации у пользователя не будет выбрано никаких навыков. При открытии страницы профиля будет проведена проверка роли пользователя, если роль пользователя - кандидат, то будет отображаться форма для выбора профессиональных навыков. В серверной части у функции для обработки запросов пользователя будут декораторы от Django Rest Framework. Декораторы позволяют "обернуть" одну функцию другой, добавляя новый функционал без изменения ее кода. В моем случае я использую 3 декоратора:

- `@api_view` - позволяет выполнять запросы только определенного метода HTTP. (например: GET-запрос, POST-запрос)
- `@authentication_classes` - нужен для указания механизмов аутентификации, которые должны применяться к определенному представлению или набору представлений.
- `@permission_classes` - отвечает за управление доступом к представлениям на основе прав доступа или ролей пользователя. (мне необходима только проверить авторизован ли пользователь на основе заголовка Authentication в HTTP-запросе).

Для обработки запроса на получение списка навыков используем функцию `get_skills`. Она создает список навыков из таблицы Skill, а затем отправляет JSON - файл клиенту. Функция приведена на рисунке 3.4.

```

@api_view(['POST'])
@authentication_classes([SessionAuthentication, TokenAuthentication])
@permission_classes([IsAuthenticated])
def get_skills(request):
    skills = Skill.objects.all()
    skills_json = [{'name': skill.name} for skill in skills]
    return Response({'skills': skills_json})

```

Рисунок 3.4

Теперь на клиентской части сделаем страницу профиля пользователя с формой для редактирования навыков. После выполнения запроса на получение списка навыков мы должны это корректно отобразить в форме. Так как количество навыков большое, то мы будем группировать навыки и отображать их отдельно в dropdown. Реализация страницы приведена на рисунке 3.5:

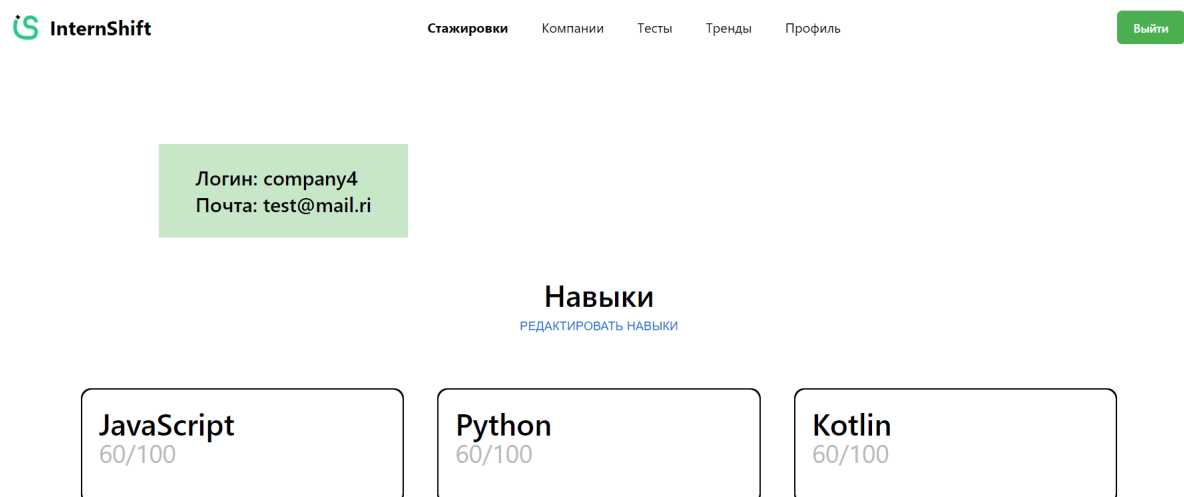


Рисунок 3.5 - Страница профиля пользователя

При нажатии на кнопку редактировать навыки мы попадаем на модальное окно с возможностью выбрать профессиональные навыки. Модальное окно с открытым dropdown приведена на рисунке 3.6.

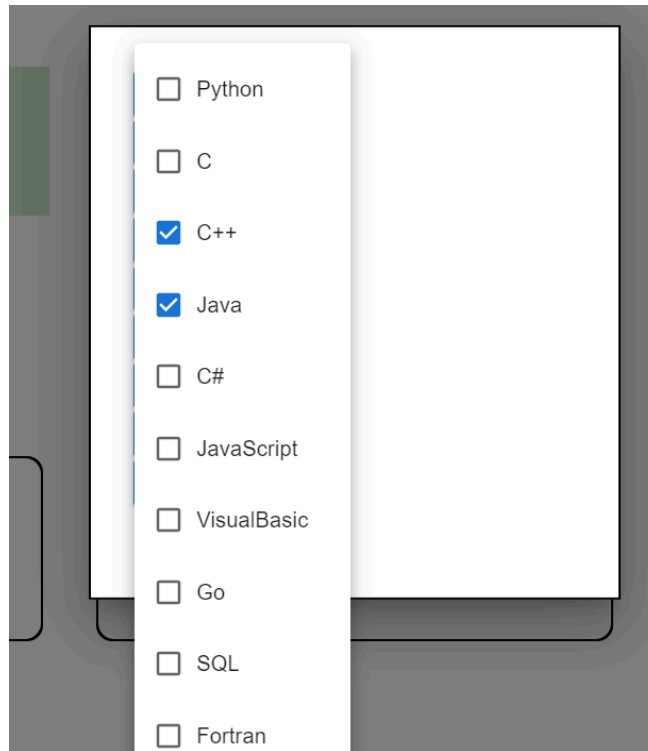


Рисунок 3.6 - Модальное окно с выбором навыков

В конце модального окна я реализовал кнопку “Сохранить”, при нажатии которого на сервер отправляется запрос об изменении таблицы SkillUser. Я показываю в чекбоксах уже выбранными те навыки, которые были изначально выбраны пользователем. При первой регистрации все чекбоксы невыбранные. То есть записей в таблице SkillUser нет. После редактирования навыков, а также при повторном открытии модального окна записи подтягиваются с сервера и уже некоторые чекбоксы становятся закрашенными. Я реализовал функцию на серверной стороне, которая на вход принимает все нажатые пользователем навыки, а затем отправляем запрос на изменение на сервер. Пользователь также может оставить чекбокс пустым, тогда те навыки, которые были изначально привязаны к пользователю будут удалены. Функция приведена на рисунке 3.7

```

@api_view(['POST'])
@authentication_classes([SessionAuthentication, TokenAuthentication])
@permission_classes([IsAuthenticated])
def save_skills(request):
    data = json.loads(request.body)
    submitted_skills = set(data.get('skills', []))
    user_skills = SkillUser.objects.filter(user=request.user)
    existing_skills = set(user_skill.skill.name for user_skill in user_skills)
    skills_to_delete = existing_skills - submitted_skills
    user_skills.filter(skill__name__in=skills_to_delete).delete()
    skills_to_add = submitted_skills - existing_skills
    new_skill_users = []
    for skill_name in skills_to_add:
        skill = Skill.objects.get_or_create(name=skill_name)[0]
        new_skill_users.append(SkillUser(user=request.user, skill=skill))
    SkillUser.objects.bulk_create(new_skill_users)

    return Response({'message': 'Навыки успешно обновлены пользователю'})

```

Рисунок 3.7 - функция для изменения записей SkillUser

Я получаю список навыков от фронтенд стороны, затем ищу совпадения по идентификатору пользователя. Если в списке отсутствуют навыки, которые есть в таблице SkillUser, то я их удаляю. Те навыки, которые отсутствуют в таблице SkillUser, но присутствуют в списке я добавляю. Таким образом происходит процесс обновления навыков пользователя.

Для отображения навыков только для пользователей с ролью “Candidates” используем тернарный оператор, который проверяет привязан ли пользователей к данной роли. Запрос совершается при переадресации пользователя на страницу профиля. С сервера я прошу информацию о привязанных ролях. Если это компания, то навыки не будут отображаться на странице пользователя. Тернарный оператор для вывода функционала приведен на рисунке 3.8.

```

{userInfo?.group == "companies" ? <div></div> :
  <div>
    <div className="mt-12 text-4xl font-semibold leading-10 text-center">
      Навыки
    </div>
    <div>
      <Button onClick={handleOpen}>Редактировать навыки</Button>
      <Modal

```

Рисунок 3.8 - условие для отображения функционала навыков

Теперь реализуем стажировки, которые будут иметь систему тегов. Теги, в данном контексте будут навыками привязанными к стажировке. По ним будет производиться фильтрация стажировок. Если у пользователя не будет выбранных навыков, то первыми будут показаны последние созданные стажировки. Если у пользователя будут выбраны определенные навыки, то фильтрация будет проводиться на основе количества совпадений по привязанным навыкам. Функционал по привязке навыков я сохраню с прошлой функции, то есть к каждой стажировке будет форма для заполнения тегов, он будет идентичен к форме для заполнения навыков пользователя.

Для кандидатов будет видно список доступных стажировок, а для компаний будет реализована форма для создания карточек стажировок. Внутри стажировки будет уже реализована функция для создания тестового задания привязанному к выбранной стажировке.

Начинаем с создания таблиц Internship и SkillInternship. Internship - таблица которая будет хранить информацию о стажировке: дата отбора, краткая информация, ссылки, компания, которая проводит стажировку. Так как при регистрации был выбор между компанией и кандидатом, то компаниями, в данном контексте, будут служить пользователи с ролью “companies”.

### 3.1.1 Реализация базы данных

При разработке базы данных во фреймворке Django используем готовую систему моделей данных. Модель данных представляет собой класс Python, который определяет структуру и типы данных полей базы данных.

Сначала определим модели данных. Для каждой таблицы создаем отдельный класс и определяем нужные нам поля, а также отношения между таблицами. Для таблицы User я буду использовать готовую таблицу из библиотеки `django.contrib.auth.models`. Модели данных приведены на рисунках 3.1.1 - 3.1.3

```
internshift > models.py > Company
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  class Company(models.Model):
5      name = models.CharField(max_length=100)
6      address = models.CharField(max_length=200)
7      email = models.EmailField()
8      description = models.TextField(max_length=300)
9
10     def __str__(self):
11         return self.name
12
13     class Vacancy(models.Model):
14         name = models.CharField(max_length=100)
15         description = models.TextField()
16         additional = models.TextField(blank=True)
17         link = models.URLField()
18         company = models.ForeignKey(Company, on_delete=models.CASCADE)
19
20     def __str__(self):
21         return self.name
22
```



Рисунок 3.1.1

```

23 class Skill(models.Model):
24     name = models.CharField(max_length=100)
25     description = models.TextField()
26
27     def __str__(self):
28         return self.name
29
30 class SkillUser(models.Model):
31     skill = models.ForeignKey(Skill, on_delete=models.CASCADE)
32     user = models.ForeignKey(User, on_delete=models.CASCADE)
33     result = models.CharField(max_length=100)
34
35     def __str__(self):
36         return f"{self.user.username}'s test for {self.skill.name}"
37
38 class SkillVacancy(models.Model):
39     skill = models.ForeignKey(Skill, on_delete=models.CASCADE)
40     vacancy = models.ForeignKey(Vacancy, on_delete=models.CASCADE)
41     need = models.IntegerField()
42
43     def __str__(self):
44         return str(self.id)

```

Рисунок 3.1.2

```

46 class Trend(models.Model):
47     skill = models.ForeignKey(Skill, on_delete=models.CASCADE)
48     index = models.IntegerField()
49     description = models.TextField()
50
51     def __str__(self):
52         return f"Trend for {self.skill.name}"
53
54 class Test(models.Model):
55     skill = models.ForeignKey(Skill, on_delete=models.CASCADE)
56     name = models.CharField(max_length=100)
57     description = models.TextField(max_length=300)
58
59     def __str__(self):
60         return f"Test for {self.skill.name}"
61
62 class Question(models.Model):
63     test = models.ForeignKey(Test, on_delete=models.CASCADE)
64     first = models.CharField(max_length=100)
65     second = models.CharField(max_length=100)
66     third = models.CharField(max_length=100)
67     fourth = models.CharField(max_length=100)
68     question = models.CharField(max_length=100)
69     image = models.CharField(max_length=200)

```

### Рисунок 3.1.3

После реализации моделей данных используем команду makemigrations и migrate для того, чтобы создать таблицы в подключенной базе данных.

#### 3.1.2 Реализация функций взаимодействия со стажировками

Реализуем основные функции для взаимодействия с таблицей Vacancy, это будет: получение списка стажировок, получение стажировки по идентификатору изменение, удаление. Реализация функций приведена на рисунках 3.2.1 - 3.2.3.

```
#Работа с вакансиями

# Получение списка всех вакансий
def vacancy_list(request):
    vacancies= Vacancy.objects.all()
    vacancy_json = [{ 'id': vacancy.id, 'name': vacancy.name, 'description': vacancy.description,
                      'additional': vacancy.additional, 'link': vacancy.link,
                      'company': vacancy.company.name} for vacancy in vacancies]
    return JsonResponse({'vacancies': vacancy_json})

# Получение вакансии по ID или удалить по ID
def get_or_delete_vacancy(request, vacancy_id):
    try:
        vacancy = Vacancy.objects.get(id=vacancy_id)
        if request.method == 'GET':
            vacancy_json = [{ 'id': vacancy.id, 'name': vacancy.name, 'description': vacancy.description,
                              'additional': vacancy.additional, 'link': vacancy.link,
                              'company': vacancy.company.name}]
            return JsonResponse(vacancy_json)
        elif request.method == 'DELETE':
            vacancy.delete()
            return JsonResponse({'message': 'Vacancy deleted successfully'})
    except Skill.DoesNotExist:
        return JsonResponse({'error': 'Vacancy not found'}, status=404)
```

### Рисунок 3.2.1

```

#Добавление вакансии
@require_POST
def add_vacancy(request):
    # Получение данных JSON из запроса
    data = json.loads(request.body)
    name = data.get('name')
    description = data.get('description')
    additional = data.get('additional')
    link = data.get('link')
    company_id = data.get('company')
    if company_id:
        try:
            company = Company.objects.get(id=company_id)
            vacancy = Vacancy.objects.create(name=name, description=description, additional=additional, link=link, company=company)
            return JsonResponse({'message': 'Vacancy created successfully'})
        except Company.DoesNotExist:
            return JsonResponse({'error': 'Company with specified id does not exist'}, status=400)
    else:
        return JsonResponse({'error': 'No company id provided in JSON'}, status=400)

```

Рисунок 3.2.2

```

#Обновление вакансии
def update_vacancy(request, vacancy_id):
    data = json.loads(request.body)
    try:
        vacancy = Vacancy.objects.get(id=vacancy_id)
        if 'name' in data:
            vacancy.name = data['name']
        if 'description' in data:
            vacancy.description = data['description']
        if 'additional' in data:
            vacancy.additional = data['additional']
        if 'link' in data:
            vacancy.link = data['link']
        if 'company' in data:
            company_id = data['company']
            company = Company.objects.get(id=company_id)
            vacancy.company = company
        vacancy.save()
        return JsonResponse({'message': 'Vacancy updated successfully'})
    except Vacancy.DoesNotExist:
        # Если вакансия с указанным идентификатором не найдена, возвращаем ошибку
        return JsonResponse({'error': 'Vacancy with specified id does not exist'}, status=404)
    except Company.DoesNotExist:
        # Если компания с указанным id не найдена, возвращаем ошибку
        return JsonResponse({'error': 'Company with specified id does not exist'}, status=400)

```

Рисунок 3.2.3

Для создания зависимостей между стажировками и необходимыми навыками, реализуем функцию, которая принимает на вход идентификатор навыка и стажировки. Реализация функции приведена на рисунке 3.2.4

#Работа по соединению вакансии с навыками

```
def create_vacancy_skill(request):
    data = json.loads(request.body)
    vacancy = data.get("vacancy_id")
    skill = data.get("skill_id")
    need = data.get("need")
    try:
        vacancySkill = SkillVacancy.objects.get(vacancy=vacancy, skill=skill)
        return JsonResponse({'message': 'VacancySkill already exists'})
    except SkillVacancy.DoesNotExist:
        try:
            newVacancy = Vacancy.objects.get(id=vacancy)
            newSkill = Skill.objects.get(id=skill)
            vacancySkill = SkillVacancy.objects.create(vacancy=newVacancy, skill=newSkill, need=need)
            return JsonResponse({'vacancy': newVacancy.name, 'skill': newSkill.name, 'need': need})
        except Skill.DoesNotExist:
            return JsonResponse({'error': 'Skill not found'}, status=404)
        except Vacancy.DoesNotExist:
            return JsonResponse({'error': 'Vacancy not found'}, status=404)
```

Рисунок 3.2.4

### 3.1.3 Реализация функций взаимодействия с тестированием

Сначала реализуем основные функции работы с таблицей Skill. Нам понадобятся методы для создания, изменения, удаления, а также получения списка, либо конкретного навыка по идентификатору. А также сделаем функцию для фильтрации навыков по названию. Реализация функций приведена на рисунках 3.3.1

```
59 # Получение списка всех навыков
60 def skill_list(request):
61     skills = Skill.objects.all()
62     skills_json = [{'id': skill.id, 'name': skill.name, 'description': skill.description} for skill in skills]
63     return JsonResponse({'skills': skills_json})
64
65 # Получение навыка по ID или удалить по ID
66 def get_or_delete_skill(request, skill_id):
67     try:
68         skill = Skill.objects.get(id=skill_id)
69         if request.method == 'GET':
70             skill_json = {'id': skill.id, 'name': skill.name, 'description': skill.description}
71             return JsonResponse(skill_json)
72         elif request.method == 'DELETE':
73             skill.delete()
74             return JsonResponse({'message': 'Skill deleted successfully'})
75
76     except Skill.DoesNotExist:
77         return JsonResponse({'error': 'Skill not found'}, status=404)
78
```

Рисунок 3.3.1

```

#Добавление навыка
@require_POST
def add_skill(request):
    # Получение данных JSON из запроса
    data = json.loads(request.body)
    name = data.get('name')
    description = data.get('description')
    skill = Skill.objects.create(name=name, description=description)
    return JsonResponse({'id': skill.id, 'name': skill.name, 'description': skill.description})

#Фильтрация навыков
def filter_skills(request):
    json_data = json.loads(request.body)

    if 'name' in json_data:
        name_to_filter = json_data['name']
        filtered_skills = Skill.objects.filter(name=name_to_filter)
        filtered_skills_list = [{'name': skill.name, 'description': skill.description} for skill in filtered_skills]
        return JsonResponse({'skills': filtered_skills_list})
    else:
        return JsonResponse({'error': 'Key "name" is missing in JSON'}, status=400)

```

Рисунок 3.3.2

Для проверки и сохранения результатов тестирования мы будем получать на вход данные о пользователе и навыке, а также список выбранных ответов по каждому вопросу. Реализация функции для проверки тестирования приведена на рисунке 3.3.3. В дальнейшем будут добавлены новые варианты тестового задания от компании.

```

@require_POST
def check_test(request):
    json_data = json.loads(request.body)
    questions = json_data.get('questions', [])
    user_id = json_data.get('user_id')
    user, created = User.objects.get_or_create(username=user_id)
    if questions:
        results = []
        for question_data in questions:
            answer_to_check = question_data.get('answer')
            question_matches = Question.objects.get(id=question_data['question_id'])
            if question_matches.answer == answer_to_check:
                results.append({'question_id': question_data['question_id'], 'match': True})
            else:
                results.append({'question_id': question_data['question_id'], 'match': False})
        total = 0
        correct = 0
        for i in results:
            total += 1
            if i['match'] == True:
                correct += 1
        percent_correct = (correct / total) * 100
        normalized_score = int((percent_correct * 100) // 100)
        skill_id = json_data.get('skill_id')
        skill, _ = Skill.objects.get_or_create(id=skill_id)
        skill_user, _ = SkillUser.objects.get_or_create(skill=skill, user=user)
        skill_user.result = str(normalized_score)
        skill_user.save()

        return JsonResponse({'result': normalized_score})
    else:
        return JsonResponse({'error': 'No questions provided in JSON'}, status=400)

```

Рисунок 3.3.3

Пример JSON-файла для отправки на тестирование приведен на рисунке 3.3.4.

```

1  {
2  |   "user_id": 2,
3  |   "skill_id": 3,
4  |   "questions": [
5  |       {
6  |           "question_id": 1,
7  |           "answer": "Second"
8  |       },
9  |       {
10 |           "question_id": 2,
11 |           "answer": "Second"
12 |       }
13 |   ]
14 }

```

Рисунок 3.3.4

После проведения тестирования результаты будут создаваться, либо перезаписываться в таблице SkillUser. Таким образом мы будем хранить результаты о проведенном тестировании. Это только начальный этап при тестовом задании. То есть пока есть функционал для тестирования знаний, но позже будет добавлен функционал для обработки кода, написанного пользователем.

#### 3.1.4 Реализация функций взаимодействия с компаниями

Реализуем основные функции для работы с таблицей компаний: получение списка компаний, получение конкретной компании по идентификатору, удаление, обновление. На рисунках приведены реализации функций 3.4.1 - 3.4.3.

```

# Получение списка всех компаний
def company_list(request):
    companies= Company.objects.all()
    companies_json = [{ 'id': company.id, 'name': company.name, 'description': company.description,
                        'address': company.address, 'email': company.email} for company in companies]
    return JsonResponse({'companies': companies_json})

# Получение вакансии по ID или удалить по ID
def get_or_delete_company(request, company_id):
    try:
        company = Company.objects.get(id=company_id)
        if request.method == 'GET':
            company_json = { 'id': company.id, 'name': company.name, 'description': company.description,
                            'address': company.address, 'email': company.email}
            return JsonResponse(company_json)
        elif request.method == 'DELETE':
            company.delete()
            return JsonResponse({'message': 'Company deleted successfully'})
    except Company.DoesNotExist:
        return JsonResponse({'error': 'Company not found'}, status=404)

```

Рисунок 3.4.1

```

#Добавление компании
@require_POST
def add_company(request):
    # Получение данных JSON из запроса
    data = json.loads(request.body)
    name = data.get('name')
    description = data.get('description')
    address = data.get('address')
    email = data.get('email')
    try:
        company = Company.objects.get(name=name)
        return JsonResponse({'message': 'Company already exists'})
    except Company.DoesNotExist:
        company = Company.objects.create(name=name, description=description, address=address, email=email)
        return JsonResponse({'id': company.id, 'name': company.name,
                            'description': company.description, 'address': company.address, 'email': company.email})

```

Рисунок 3.4.2

```

#Обновление компании
def update_company(request, company_id):
    data = json.loads(request.body)
    try:
        company = Company.objects.get(id=company_id)
        if 'name' in data:
            company.name = data['name']
        if 'description' in data:
            company.description = data['description']
        if 'address' in data:
            company.address = data['address']
        if 'email' in data:
            company.email = data['email']
        company.save()
        return JsonResponse({'message': 'Company updated successfully'})
    except Company.DoesNotExist:
        return JsonResponse({'error': 'Company with specified id does not exist'}, status=400)

```

Рисунок 3.4.3



### 3.1.5 Реализация функций взаимодействия с профилем

Для работы на странице профиля нам понадобятся две функции: функция для вывода информации о пользователе, функция для вывода списка навыков пользователя, а также редактирование навыков пользователя. Реализации функций приведена на рисунке 3.5.1.

```
#Работа с профилем
def get_user(request, user_id):
    try:
        data = User.objects.get(id=user_id)
        user_json = {
            "email": data.email,
        }
        return JsonResponse(user_json)
    except User.DoesNotExist:
        return JsonResponse({'error': 'User with specified id does not exist'}, status=404)

def get_user_skills(request, user_id):
    try:
        user = User.objects.get(id=user_id)
        skill_users = SkillUser.objects.filter(user=user)
        skill_users_list = [[skill_user.skill.name, skill_user.result] for skill_user in skill_users]
        # Возвращаем список в формате JSON
        return JsonResponse({'skill_users': skill_users_list})
    except User.DoesNotExist:
        return JsonResponse({'error': 'User with specified id does not exist'}, status=404)
```

Рисунок 3.5.1

Пример вывода списка навыков пользователя приведен на рисунке 3.5.2.

```
1  {
2      "skill_users": [
3          [
4              "Java",
5              "50"
6          ],
7          [
8              "Python",
9              "0"
10         ]
11     ]
12 }
```

Рисунок 3.5.2

### 3.1.6 Реализация функций для аутентификации пользователя

Для реализации аутентификации пользователя используем готовые функции фреймворка Django. Для этого нам понадобится две функции , для регистрации и авторизации. При авторизации мы отправляем к клиенту CSRF-токен для того, чтобы потом определять пользователя по нему. При регистрации мы будем сохранять данные пользователя в таблице User. Реализации функций приведены на рисунках 3.6.1 - 3.6.2

```
def login_view(request):
    if request.method == 'POST':
        # Получить данные JSON из запроса
        data = json.loads(request.body)
        username = data.get('login')
        password = data.get('password')
        # Аутентификация пользователя
        user = authenticate(request, username=username, password=password)
        if user is not None:
            # Вход пользователя
            login(request, user)
            return JsonResponse({'message': 'Login successful'})
        else:
            return JsonResponse({'message': 'Invalid credentials'}, status=401)
    else:
        return JsonResponse({'message': 'Method not allowed'}, status=405)
```

Рисунок 3.6.1

```
def register_view(request):
    if request.method == 'POST':
        # Получить данные JSON из запроса
        data = json.loads(request.body)
        username = data.get('login')
        password = data.get('password')
        email = data.get('email')
        # Проверка, что все данные присутствуют
        if not (username and password and email):
            return JsonResponse({'message': 'Missing required fields'}, status=400)
        # Создание нового пользователя
        user = User.objects.create_user(username=username, password=password, email=email)
        return JsonResponse({'message': 'Registration successful'})
    else:
        return JsonResponse({'message': 'Method not allowed'}, status=405)

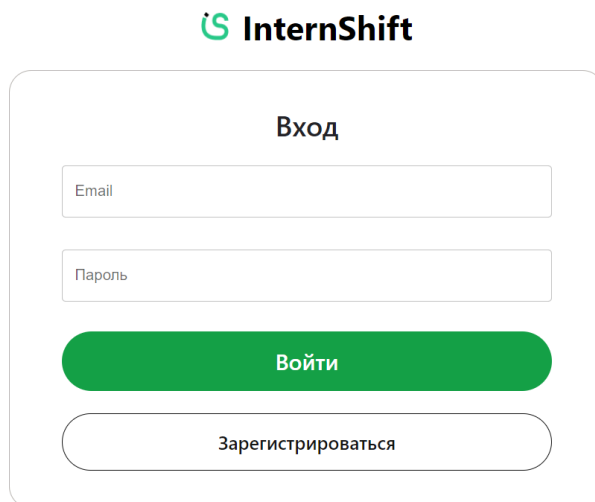
def logout_view(request):
    logout(request)
    return JsonResponse({'message': 'Logged out successfully'})
```

## Рисунок 3.6.2

### 3.2 Разработка клиентской части приложения

Для разработки клиентской части приложения использовался фреймворк React на основе языка TypeScript, а также CSS библиотеки для дизайна интерфейса: TailwindCSS [31], Ant Design

Для реализации страницы входа были использованы компоненты из библиотеки Tailwind. А также каждое значение поля, которую изменил пользователь динамически меняется. Страница входа и регистрации приведены на рисунках 3.7 - 3.8.



IS InternShift

Вход

Email

Пароль

Войти

Зарегистрироваться

Рисунок 3.7 - Страница входа пользователя

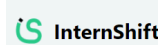
### Регистрация

Зарегистрироваться

Войти

Рисунок 3.8 - Страница регистрации пользователя

На странице стажировок компании могут публиковать свои стажировки, а также при нажатии на карточку стажировки пользователь будет отправлен на страницу с дополнительной информацией и с возможностью пройти тестовое задание. Страница со стажировками приведена на рисунке 3.9.



СтажировкиКомпанииТестыТрендыПрофиль

Выйти

## Стажировки

поиск по стажировкам

ГородСпециальностьФормат

без опыта работы — От 1 года опыта — Специально для вас

### Linux Archlinux Backend

без опыта работы

МоскваУдаленно

Рисунок 3.9 - Страница со стажировками

Страница с компаниями хранит информацию о компании, в которой можно просмотреть доступные стажировки в этой компании. Страница с компаниями приведена на рисунке 3.10.

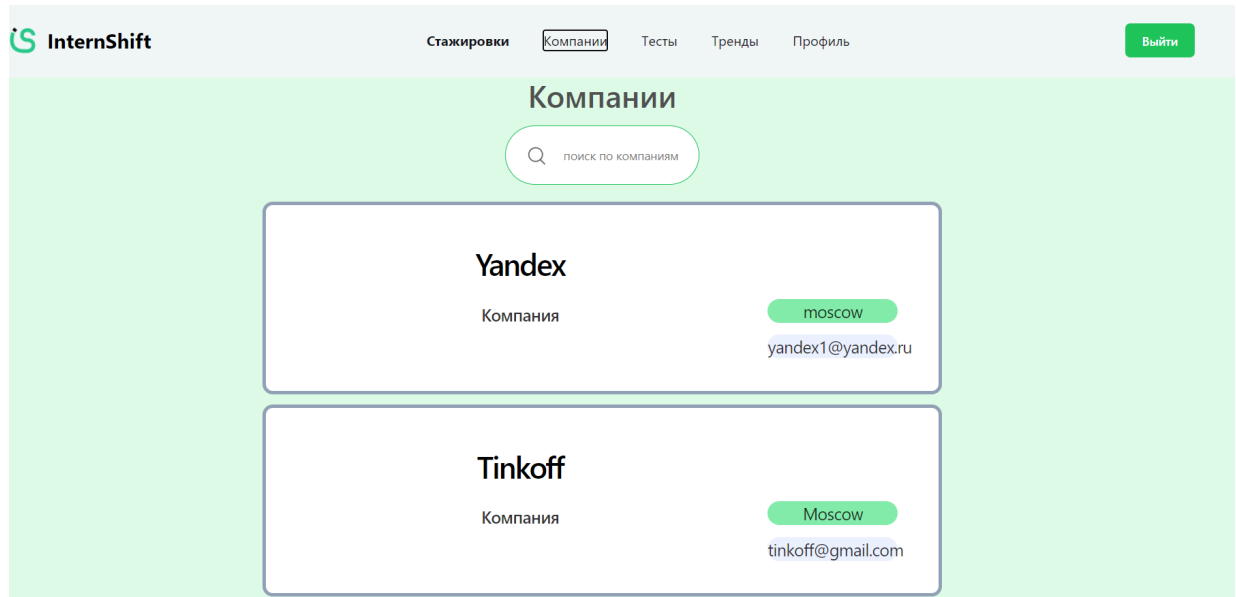


Рисунок 3.10 - Страница с компаниями

Страница с тестами будет заполнена таким образом, пользователь будет отслеживать в какую компанию по какому направлению он выполняет тестовое задание, а также при нажатии на любую карточку пользователь переместиться на страницу, где есть система по прохождению тестового задания. Страница с тестами приведена на рисунке 3.11.

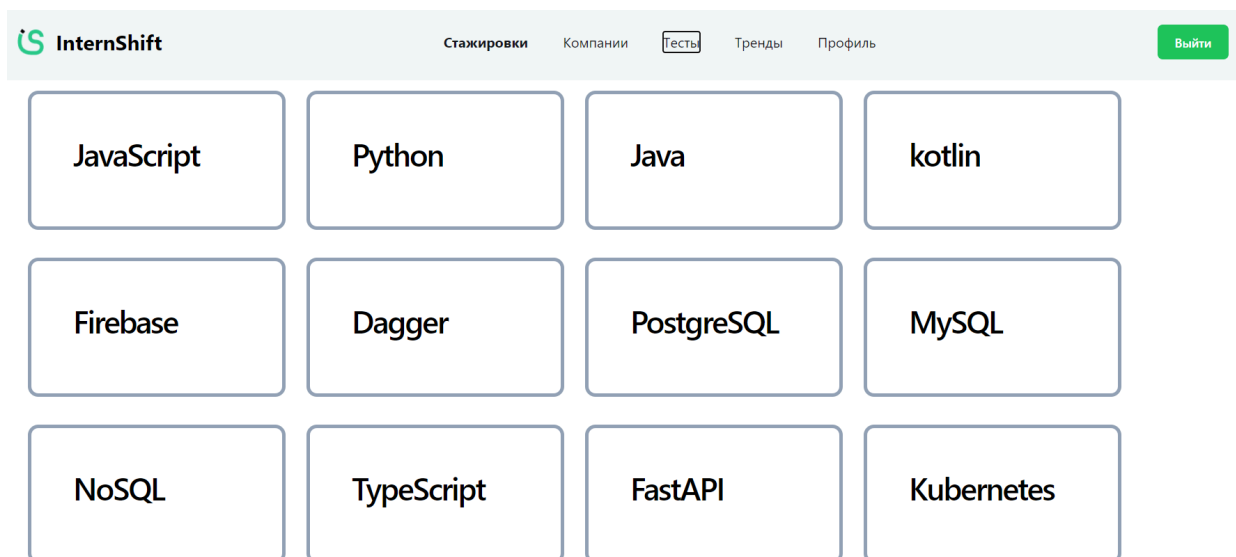


Рисунок 3.11 - Страница с тестами

Страница с трендами компаний где отображены стажировки, а также есть топ навыков, которые требуют в различных стажировках. Страница с трендами приведена на рисунке 3.12.

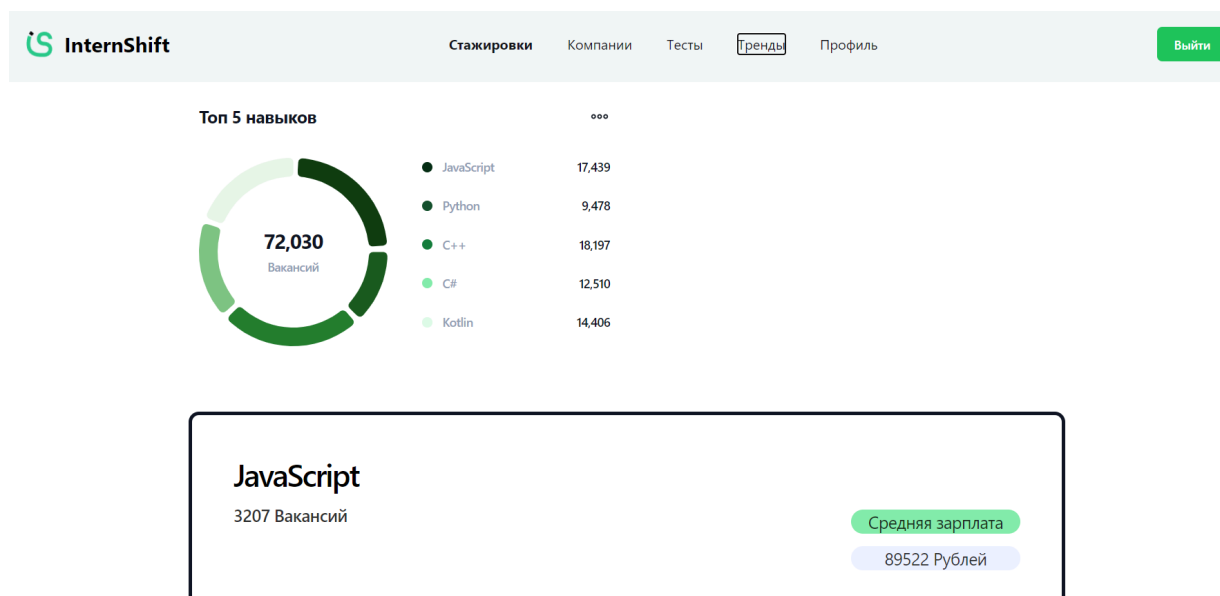


Рисунок 3.12 - Страница с трендами по стажировкам

Профиль пользователя будет отражать личную информацию и данные о профессиональных навыках. Страница с профилем приведена на рисунке 3.13.

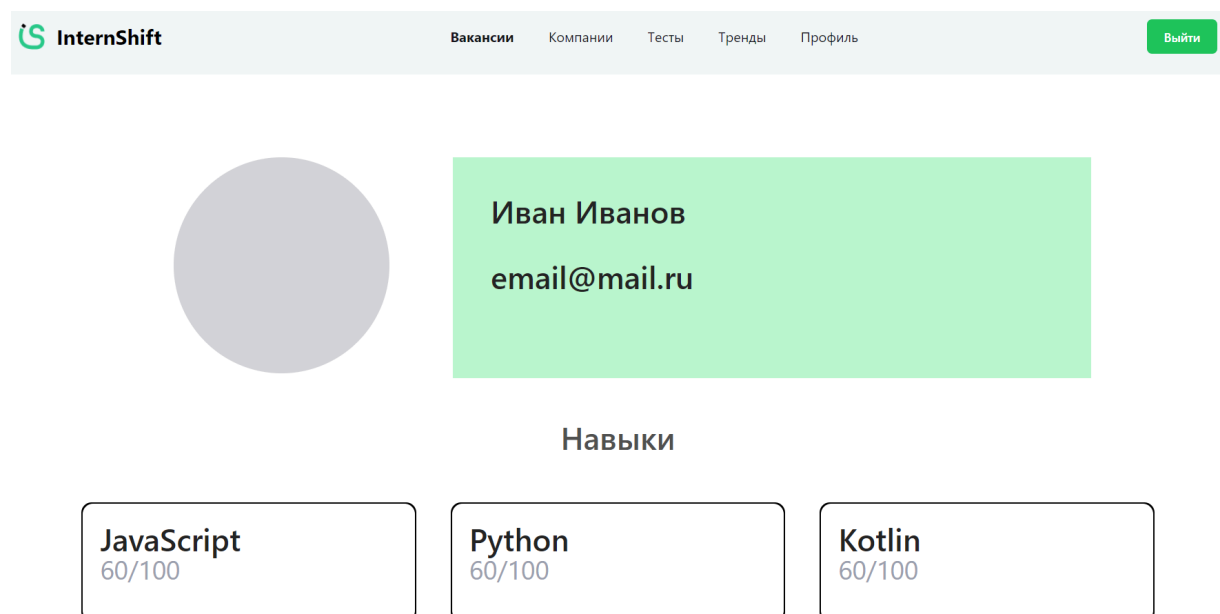


Рисунок 3.13 - Страница профиля пользователя

На данном этапе есть один вариант составления теста, это обычный тест на знание технологий. Компании могут как сделать данный тест как раздел всего тестового задания. Страница с тестом на знание приведена на рисунке 3.12.

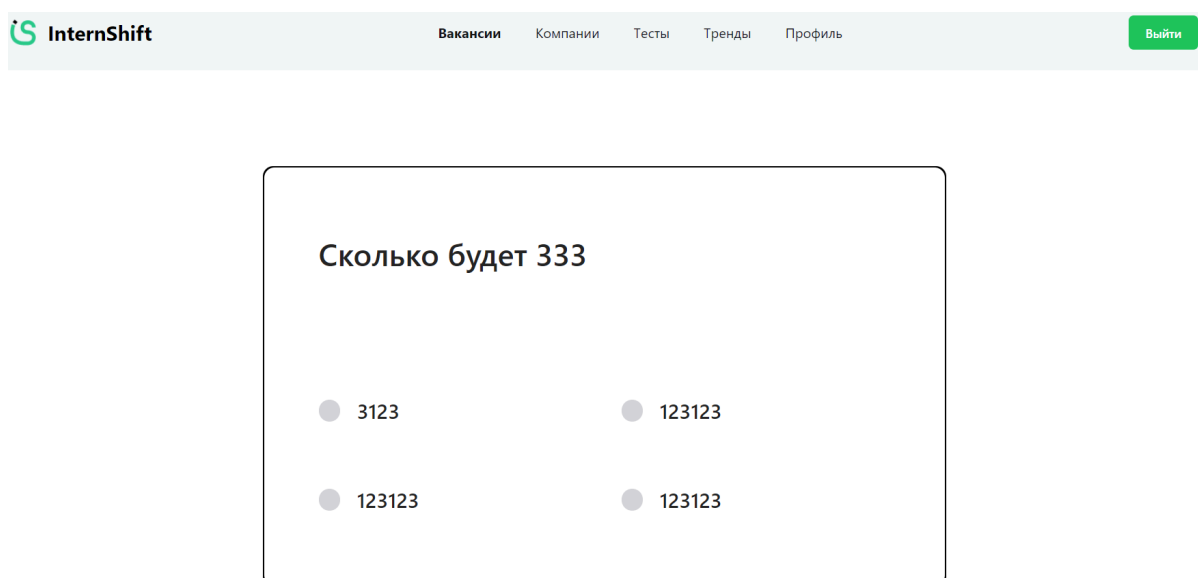


Рисунок 3.14 - Страница теста на знание технологии

### 3.3 Разработка сервиса для сбора и анализа данных с платформы HeadHunter

Для сбора данных и автоматического сбора стажировок с платформы HeadHunter был разработан скрипт для загрузки всех данных с API агрегатора. Далее была реализована фильтрация и отбор только нужных вакансий с ключевым словом “Стажировка”. Данные будут использованы для вывода тренда технологий, чтобы пользователь мог рассматривать, какие сейчас технологии популярны исходя от количества стажировок и средней зарплате. Реализация функций приведена на рисунках 3.15 - 3.17.

```

def get_page(request: str, area_id: int, period,
              only_with_salary=False, order_by='relevance', page=0) -> tuple:

    params = {'text': request,
              'area': area_id,
              'page': page,
              'per_page': 100,
              'only_with_salary': only_with_salary,
              'order_by': order_by,
              'period': period}

    with get('https://api.hh.ru/vacancies', params=params) as r:
        json_object = r.json()
        return json_object['items'], json_object['found']

```

Рисунок 3.15

```

def get_job_info(job):
    data, count = get_page(job, region, last_day)
    salary_sum = 0.0
    salary_count = 0
    for elem in data:
        if "salary" not in elem:
            continue
        if not elem["salary"]:
            continue
        salary = elem["salary"]["from"] or elem["salary"]["to"]
        try:
            salary_sum += salary * rates[elem["salary"]["currency"]]
            salary_count += 1
        except Exception:
            continue

    salary_avg = 0.0
    if salary_count:
        salary_avg = round(salary_sum / salary_count)
    return {"hh_salary": salary_avg, "hh_count": count}

```

Рисунок 3.16



```

def get_vacancies_for(job, reg):
    data, count = get_page(job, reg, last_day)
    jobs_json = []
    for elem in data:
        temp = {}
        if elem["name"]:
            temp["name"] = elem["name"]
        if elem["snippet"]["requirement"]:
            temp["description"] = elem["snippet"]["requirement"]
        if elem["schedule"]:
            temp["format"] = elem["schedule"]["name"]
        if elem["area"]:
            temp["town"] = elem["area"]["name"]
        if elem["experience"]["name"]:
            temp["additional"] = elem["experience"]["name"]
        if elem["employer"]["name"]:
            temp["company"] = elem["employer"]["name"]
        if elem["alternate_url"]:
            temp["link"] = elem["alternate_url"]
        jobs_json.append(temp)
    return jobs_json

```

Рисунок 3.17

### 3.4 Вывод по третьей главе

В данной главе была проведена разработка веб-приложения. Описаны основные функции серверной части приложения и приведены рисунки интерфейса клиентской части. Также был разработан скрипт для вывода статистики вакансий с веб-сайта hh.ru и автоматического заполнения таблиц базы данных с вакансиями и компаниями. Централизация системы была реализована в виде системы тестов на отдельной странице, таким образом, пользователи могут смотреть когда и что им отправлять на проверку, а также проходить тесты на знание и в дальнейшем проходить задачи, где нужно будет писать код.

## **ЗАКЛЮЧЕНИЕ**

В данной дипломной работе была разработана веб-платформа InternShift, предназначенная для централизации системы тестовых заданий. Платформа предлагает пользователям удобный поиск стажировок по технологиям. А также система предлагает возможность пройти тестовое задание от компании на знание выбранных технологий, а также в дальнейшем и на написание кода.

В ходе работы над данным веб-приложением была рассмотрена предметная область и рассмотрены существующие решения, где был функционал по поиску стажировок. Обзор показал, что у решений полностью отсутствует система для прохождения тестовых заданий на их платформе, таким образом, есть актуальность в реализации собственной системы с дополнительным функционалом.

Такое приложение можно внедрить в различные платформы, как отдельный сервис, только ориентированное под систему для прохождения тестовых заданий.

Что можно добавить в будущем:

- Интеграция с социальными сетями: Можно сделать интеграции для авторизации с помощью социальных сетей.
- Разработка мобильного приложения: Можно создать мобильное приложение для более удобного мониторинга статуса отправленного заявления, а также для возможности проходить тестирование с мобильного устройства.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. HeadHunter [Электронный ресурс]. – URL: <https://dev.hh.ru/> (дата обращения – 12.04.2024).
2. FutureToday [Электронный ресурс]. – URL: <https://fut.ru/> (дата обращения – 24.04.2024).
3. JobbyAI [Электронный ресурс]. – URL: <https://jobby.ai/> (дата обращения – 24.04.2024).
4. SuperJob [Электронный ресурс]. – URL: <https://www.superjob.ru/> (дата обращения – 24.04.2024).
5. PostgreSQL [Электронный ресурс]. – URL: <https://www.postgresql.org/docs/16/> (дата обращения – 03.04.2024).
6. MySQL [Электронный ресурс]. – URL: <https://dev.mysql.com/doc/> (дата обращения – 03.04.2024).
7. MariaDB [Электронный ресурс]. – URL: <https://mariadb.com/kb/en/documentation/> (дата обращения – 03.04.2024).
8. Oracle [Электронный ресурс]. – URL: <https://docs.oracle.com/en/database/> (дата обращения – 03.04.2024).
9. Microsoft SQL [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/sql/> (дата обращения – 03.04.2024).
10. SQLite [Электронный ресурс]. – URL: <https://www.sqlite.org/docs.html> (дата обращения – 03.04.2024).
11. Python [Электронный ресурс]. – URL: <https://docs.python.org/3/> (дата обращения – 22.03.2024).
12. Java [Электронный ресурс]. – URL: <https://docs.oracle.com/en/java/> (дата обращения – 22.03.2024).

- 13.PHP [Электронный ресурс]. – URL: <https://www.php.net/docs.php> (дата обращения – 22.03.2024).
- 14.C# [Электронный ресурс]. – URL: <https://learn.microsoft.com/en-us/dotnet/csharp/> (дата обращения – 22.03.2024).
- 15.Go [Электронный ресурс]. – URL: <https://go.dev/doc/> (дата обращения – 22.03.2024).
- 16.Django [Электронный ресурс]. – URL: <https://docs.djangoproject.com/en/5.0/> (дата обращения – 03.04.2024).
- 17.Flask [Электронный ресурс]. – URL: <https://flask.palletsprojects.com/en/3.0.x/> (дата обращения – 03.04.2024).
- 18.Pyramid [Электронный ресурс]. – URL: <https://docs.pylonsproject.org/projects/pyramid/en/latest/> (дата обращения – 03.04.2024).
- 19.FastAPI [Электронный ресурс]. – URL: <https://fastapi.tiangolo.com/> (дата обращения – 03.04.2024).
- 20.HTML [Электронный ресурс]. – URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (дата обращения – 18.04.2024).
- 21.CSS [Электронный ресурс]. – URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (дата обращения – 18.04.2024).
- 22.JavaScript [Электронный ресурс]. – URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (дата обращения – 18.04.2024).

- 23.Методология FSD [Электронный ресурс]. – URL: <https://feature-sliced.design/docs/get-started/overview> (дата обращения – 12.04.2024).
- 24.TypeScript [Электронный ресурс]. – URL: <https://www.typescriptlang.org/docs/> (дата обращения – 01.04.2024).
- 25.Dart [Электронный ресурс]. – URL: <https://dart.dev/> (дата обращения – 01.04.2024).
- 26.Angular [Электронный ресурс]. – URL: <https://angular.io/docs> (дата обращения – 03.04.2024).
- 27.React [Электронный ресурс]. – URL: <https://react.dev/> (дата обращения – 03.04.2024).
- 28.Vue.js [Электронный ресурс]. – URL: <https://vuejs.org/guide/introduction.html> (дата обращения – 03.04.2024).
- 29.NestJS [Электронный ресурс]. – URL: <https://docs.nestjs.com/> (дата обращения – 03.04.2024).
- 30.Next.js [Электронный ресурс]. – URL: <https://nextjs.org/docs> (дата обращения – 03.04.2024).
- 31.Tailwind CSS [Электронный ресурс]. – URL: <https://v2.tailwindcss.com/docs> (дата обращения – 15.04.2024).