**Содержание**

Введение 6

1 Аналитический обзор предметной области 8

1.1 Обзор предметной области 8

1.2 Обзор существующих аналогов 9

1.3 Постановка задачи 10

1.4 Выводы по главе 11

2 Алгоритмическое конструирование 13

2.1 Алгоритм клиент-серверного взаимодействия 13

2.2 Алгоритм регистрации пользователя 14

2.3 Алгоритм аутентификация пользователя 16

2.4 Алгоритм отправки сообщений о заказах и вакансиях 17

2.5 Алгоритм HMAC с использованием SHA–256 19

2.6 Алгоритм хеширования (SHA–256) 21

2.7 Вывод по главе 23

3 Программное конструирование 24

3.1 Обоснование выбора средств разработки 24

3.2 Объектно-ориентированная модель программного средства 27

3.3 Описание структуры баз данных 33

3.4 Вывод по главе 35

4 Тестирование программного средства 36

4.1 Описание процесса тестирования 36

4.2 Выводы по главе 42

Перечень использованных информационных ресурсов 54

Приложение А Техническое задание 56

Приложение Б Листинг программы 64

Введение

Современный мир образования стремительно меняется, и онлайн-платформы для обучения становятся неотъемлемой частью учебного процесса. Существует множество образовательных ресурсов, таких как Coursera, Udemy и другие, однако создание специализированной платформы, адаптированной под конкретные нужды пользователей, остается актуальной задачей. Процесс организации обучения, управления учебным контентом и взаимодействия между участниками образовательного процесса может быть сложным и требует значительных усилий от преподавателей и студентов.

Для решения этой проблемы возможно использование комплексных образовательных платформ, которые предоставляют все необходимые инструменты в одном месте. Такие платформы могут значительно упростить процесс обучения, позволяя преподавателям и студентам сосредоточиться на учебном материале и взаимодействии, а не на организационных вопросах.

Целью данной работы является создание обучающей платформы-сайта, которая будет включать в себя широкий спектр функциональных возможностей для эффективного обучения. Этот сервис позволит пользователям взаимодействовать в реальном времени, обмениваться материалами, проходить тесты и выполнять домашние задания, а также управлять своими учебными процессами через личный кабинет.

.

Выделены следующие задачи:

* Анализ существующих образовательных платформ и их функциональных возможностей;
* выбор подходящих веб-технологий и средств разработки;
* разработка функционала интерактивной доски с отображением изменений в реальном времени;
* реализация чата для общения пользователей на вкладке дэшборда;
* создание системы управления методическими материалами и архивом PDF;
* разработка модуля для создания и прохождения тестов;
* реализация функционала для управления домашними заданиями;
* разработка личного кабинета, системы регистрации и авторизации;
* обеспечение безопасности и удобства использования платформы.

Первая глава работы содержит обзор предметной области и формулирование условий задачи.

Вторая глава посвящена описанию алгоритмов, используемых для решения поставленной задачи.

В третьей главе дано обоснование выбора технологий и инструментов для реализации программного средства.

Четвертая глава представляет созданное программное средство и описывает его функциональные возможности.

Разработанный прототип обучающей платформы может быть использован в качестве основы для дальнейших исследований в области онлайн-образования, а также в качестве реального инструмента для образовательных учреждений, репетиторов и корпоративных тренеров.

# 1 Аналитический обзор предметной области

В данном разделе проводится анализ существующих программных решений в области образовательных платформ. Рассматриваются особенности их функциональности, а также оцениваются преимущества и недостатки.

## 1.1 Обзор предметной области

Образовательные платформы стали неотъемлемой частью современного образования, обеспечивая инструменты для обучения, управления учебным процессом и взаимодействия между преподавателями и студентами. Эти платформы объединяют функции управления курсами, тестирования, коммуникации и доставки контента, поддерживая как синхронное, так и асинхронное обучение. С развитием технологий платформы включают интерактивные элементы, такие как доски для совместной работы, повышающие вовлеченность и эффективность обучения.

Основные функции образовательных платформ включают:

* Доски для совместной работы в реальном времени, визуализация учебных материалов;
* чаты, форумы, системы обмена сообщениями для взаимодействия;
* загрузка, организация и распространение материалов, таких как PDF-документы;
* создание и проведение тестов, викторин и заданий;
* регистрация, авторизация, личные кабинеты для отслеживания прогресса.

Эти функции создают комплексную среду для обучения, однако их реализация и удобство использования варьируются в зависимости от платформы.

## 1.2 Обзор существующих аналогов

В данном подразделе рассматриваются популярные решения для образовательных платформ. Описываются следующие аналоги: Moodle, Canvas и Google Classroom.

## 1.2.1 Moodle

Moodle — это система управления обучением (LMS) с открытым исходным кодом, предоставляющая гибкую и настраиваемую платформу для образовательных учреждений (Moodle.org). Она поддерживает множество плагинов, включая интеграции для интерактивных досок, таких как MIRO или Conceptboard (Moodle Plugins).

Функциональность:

* Интерактивная доска: Через плагины или внешние инструменты;
* чат: Встроенные функции чата и обмена сообщениями;
* методические материалы: Поддержка загрузки и организации файлов, включая PDF;
* тесты: Модуль викторин для создания и проведения тестов;
* домашние задания: Модуль заданий для управления домашними работами;
* личный кабинет: Профили пользователей и дашборды;
* регистрация и авторизация: Встроенное управление пользователями.

Преимущества:

* Бесплатное использование с активной поддержкой сообщества;
* высокая степень кастомизации благодаря плагинам;
* широкий набор функций для управления курсами и оценкой.

Недостатки:

* Требует технических навыков для установки и обслуживания;
* интерфейс может быть сложным для новичков;
* интерактивные доски требуют дополнительных настроек.

## 1.2.2 Google Classroom

Google Classroom — бесплатный веб-сервис для школ, упрощающий создание, распространение и оценку заданий (Google Classroom). Интегрируется с инструментами Google Workspace, включая Jamboard для интерактивных досок (Explain Everything Google)

Функциональность:

* Интерактивная доска: Интеграция с Jamboard;
* чат: Комментарии и объявления;
* методические материалы: Интеграция с Google Drive для файлов;
* тесты: Викторины через Google Forms;
* домашние задания: Управление заданиями;
* личный кабинет: Привязка к учетной записи Google;
* регистрация и авторизация: Через учетные записи Google;

Преимущества:

* Бесплатно для образовательных учреждений;
* простота настройки и использования;
* знакомый интерфейс для пользователей Google.

Недостатки:

* Ограниченный набор функций по сравнению с Moodle и Canvas;
* ориентирован на школьное образование, менее подходит для вузов;
* требует учетных записей Google.

## 1.2.3 Canvas

Canvas — облачная LMS, известная своим интуитивным интерфейсом и надежной функциональностью, широко используемая в школах и вузах (Canvas LMS). Поддерживает интеграции с инструментами, такими как Explain Everything для интерактивных досок (Explain Everything).

Функциональность:

* Интерактивная доска: Интеграция с Explain Everything для совместной работы;
* чат: Форумы обсуждений и система обмена сообщениями;
* методические материалы: Загрузка и организация файлов;
* тесты: Модуль викторин и оценок;
* домашние задания: Управление заданиями;
* личный кабинет: Профили пользователей;
* регистрация и авторизация: Управляется учреждением.

Преимущества:

* Простота и интуитивность интерфейса.
* надежная интеграция с внешними инструментами.
* высокая производительность и поддержка.

Недостатки:

* Платная модель подписки, что может быть дорого.
* меньшая гибкость по сравнению с Moodle.
* зависимость от интернет-соединения.

## 1.3 Постановка задачи

Анализ аналогов показывает, что существующие платформы обладают широкими возможностями, но имеют ограничения. Moodle требует технических навыков, Canvas дорог, а Google Classroom ограничен в функционале. Ни одна из платформ не предлагает встроенной интерактивной доски с изменениями в реальном времени или чата, ограниченного вкладкой дашборда, что может быть важным для определенных сценариев обучения.

Целью проекта является создание современной образовательной платформы, которая объединит лучшие функции аналогов, устраняя их недостатки. Платформа будет интуитивной, с акцентом на интерактивные инструменты, такие как доска с реальным временем и чат на дашборде, для улучшения учебного опыта.

Для достижения поставленной цели необходимо выполнить следующие задачи:

* Разработать интуитивный интерфейс, минимизирующий информационную перегрузку;
* реализовать алгоритмы для обеспечения безопасности, целостности и эффективности работы платформы;
* спроектировать структуру базы данных для хранения данных пользователей, материалов и взаимодействий;
* провести тестирование для подтверждения функциональности, безопасности и соответствия требованиям.

## 1.4 Выводы по главе

В данном разделе рассмотрены основные концепции и цели, связанные с образовательными платформами, а также проведен обзор существующих решений. Особое внимание уделено интерфейсу, интерактивным инструментам, управлению контентом и безопасности. На основе анализа определена потребность в новой платформе, которая обеспечит удобство, функциональность и доступность. Последующие главы будут посвящены разработке платформы, включая выбор технологий и реализацию.

На основе проведенного обзора программных средств, была сформулирована задача по реализации образовательной платформы. Основными задачами становятся разработка интуитивно понятного и простого интерфейса, который не будет перегружен информацией, и предоставление широкого функционала для обучения.

Таким образом, на основании проведенного анализа и определения задач, ставится цель разработки программного средства, которое должна стать универсальным решением, подходящим как для школ, так и для вузов, с акцентом на современные образовательные технологии

.

# 2 Алгоритмическое конструирование

В данном разделе рассматриваются алгоритмы, обеспечивающие функционирование обучающей платформы-сайта. Описаны процессы клиент-серверного взаимодействия, регистрации и аутентификации пользователей, работы интерактивной доски в реальном времени, чата на дашборде, управления методическими материалами, создания и прохождения тестов, а также обработки домашних заданий.

## 2.1 Алгоритм клиент-серверного взаимодействия

Алгоритм клиент-серверного взаимодействия реализует обмен данными между клиентом (браузером пользователя) и сервером, обеспечивая функциональность платформы, включая интерактивные функции в реальном времени. Используются асинхронные технологии, такие как FastAPI для REST API и Websockets для постоянного соединения. Когда пользователь взаимодействует с платформой (например, открывает доску, отправляет сообщение в чат или загружает файл), клиент отправляет запросы, которые сервер обрабатывает и возвращает результаты.

Основные этапы:

1. Установление соединения: клиент (браузер) инициирует подключение к серверу через HTTP/HTTPS для загрузки интерфейса или через Websockets для интерактивных функций (например, интерактивной доски или чата). Websockets позволяют поддерживать постоянное соединение для передачи данных в реальном времени.
2. Аутентификация и авторизация: клиент предоставляет учетные данные (логин и пароль) через защищённое соединение (HTTPS). Сервер проверяет данные в базе данных PostgreSQL с использованием SQLAlchemy. При успешной аутентификации клиенту выдаётся токен (JWT) для авторизации доступа к функциям платформы (личный кабинет, тесты, задания).
3. Обмен сообщениями: клиент отправляет запросы к серверу через REST API (FastAPI) или Websockets. Примеры запросов: загрузка PDF-материалов, отправка сообщений в чат, обновление данных на интерактивной доске, создание теста или отправка домашнего задания. RabbitMQ используется для асинхронной обработки задач, таких как уведомления или обработка больших файлов.
4. Обработка запросов сервером: сервер выполняет операции: сохранение данных (например, результатов тестов или файлов) в базе данных PostgreSQL; обновление состояния интерактивной доски через Websockets для синхронизации изменений между пользователями; обработка чата (доступного только на вкладке дашборда) с фильтрацией сообщений по контексту. Alembic управляет миграциями базы данных для обеспечения целостности структуры.
5. Отправка ответов клиенту: сервер возвращает результаты операций (например, список материалов, обновлённое состояние доски или сообщения чата) через REST API или Websockets. Ответы форматируются в JSON для REST-запросов или как поток данных для Websockets.
6. Завершение соединения: при завершении работы клиент закрывает соединение (например, при выходе из платформы). Websockets-соединение разрывается, освобождая ресурсы сервера.

Схема алгоритма представлена на рисунке 2.1.

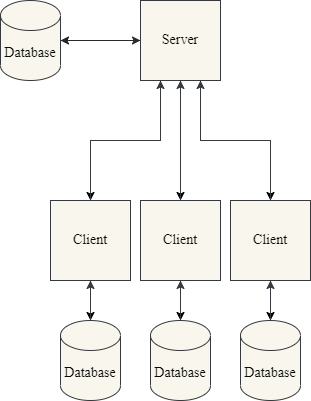


Рисунок 2.1 — Cхема клиент-серверного взаимодействия

## 2.2 Алгоритм регистрации пользователя

Алгоритм регистрации реализует создание учётной записи для доступа к платформе, обеспечивая безопасное и удобное добавление новых пользователей. Когда пользователь хочет зарегистрироваться, он заполняет форму, данные которой проверяются и сохраняются на сервере.

Алгоритм включает следующие шаги:

1. Пользователь заполняет форму на сайте в специальной для этого форме авторизации или регистрации. Данные валидируются на стороне сервера.
2. Данные отправляются на сервер через защищённый POST-запрос. Пароль хешируется перед сохранением.
3. Сервер проверяет уникальность email и username в базе данных PostgreSQL. При отсутствии конфликтов данные сохраняются с использованием SQLAlchemy. Создаётся запись в таблице пользователей с ролями.
4. Пользователю отправляется письмо с подтверждением через брокер сообщений. После подтверждения учётная запись активируется.
5. Сервер возвращает сообщение об успешной регистрации или ошибке.

Схема алгоритма представлена на рисунке 2.2.

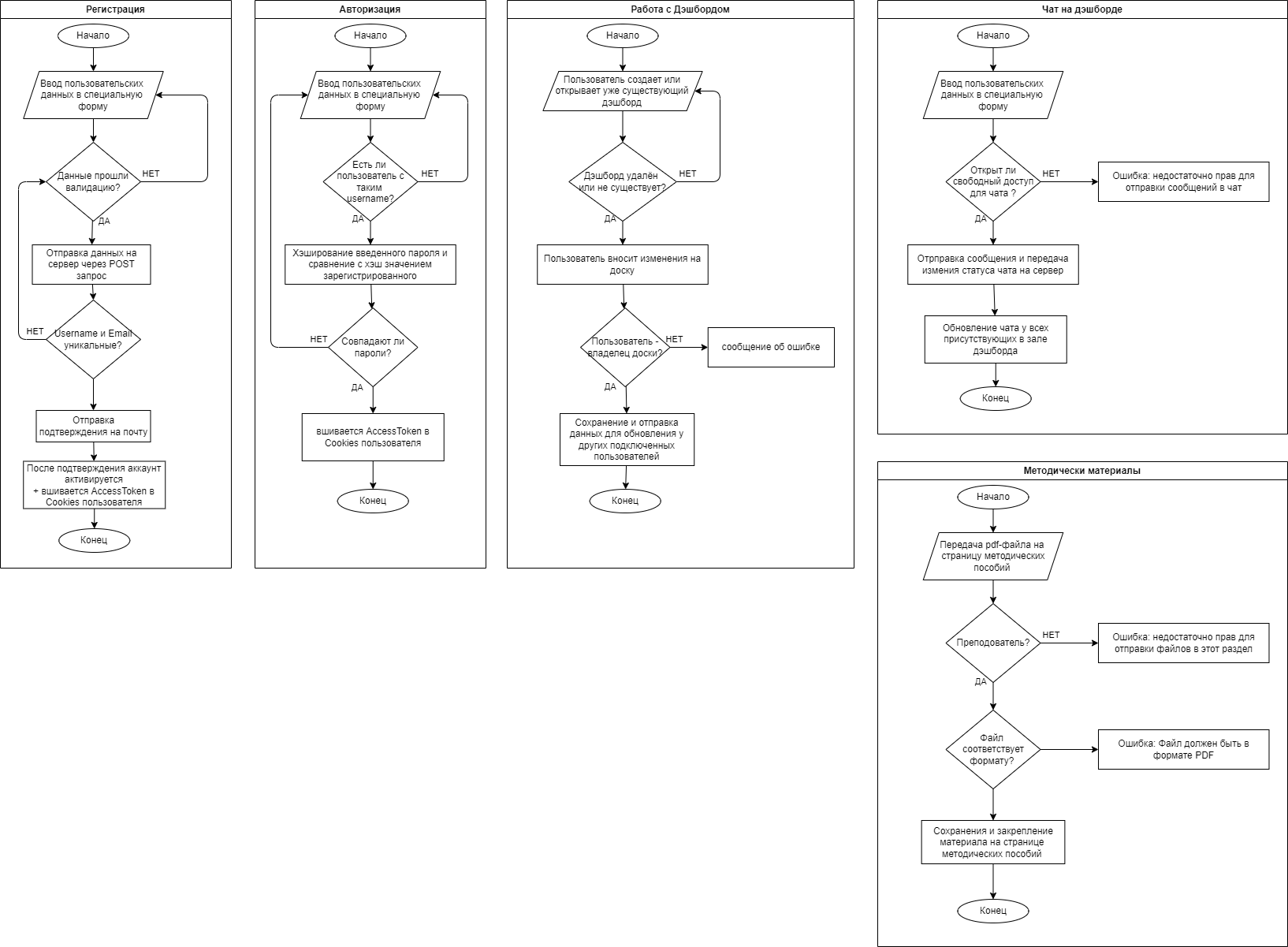


Рисунок 2.2 — Схема алгоритма регистрации

## 2.3 Алгоритм аутентификация пользователя

Алгоритм аутентификации реализует безопасный вход пользователя на платформу, проверяя его учетные данные и предоставляя доступ к функционалу. Когда пользователь хочет войти, он вводит свои данные, которые сервер проверяет, выдавая токен для дальнейшей авторизации.

Шаги алгоритма аутентификации пользователя:

1. Ввод учетных данных: пользователь вводит email и пароль в форму на фронтенде.
2. Данные отправляются через защищённый POST-запрос к FastAPI.
3. Сервер проверяет username и хеш пароля в PostgreSQL с использованием SQLAlchemy. При совпадении создаётся JWT-токен с информацией о пользователе и сроком действия.
4. Токен отправляется клиенту и сохраняется в браузере. Токен используется для авторизации последующих запросов.
5. При неверных данных возвращается сообщение об ошибке.

Схема алгоритма представлена на рисунке 2.3.

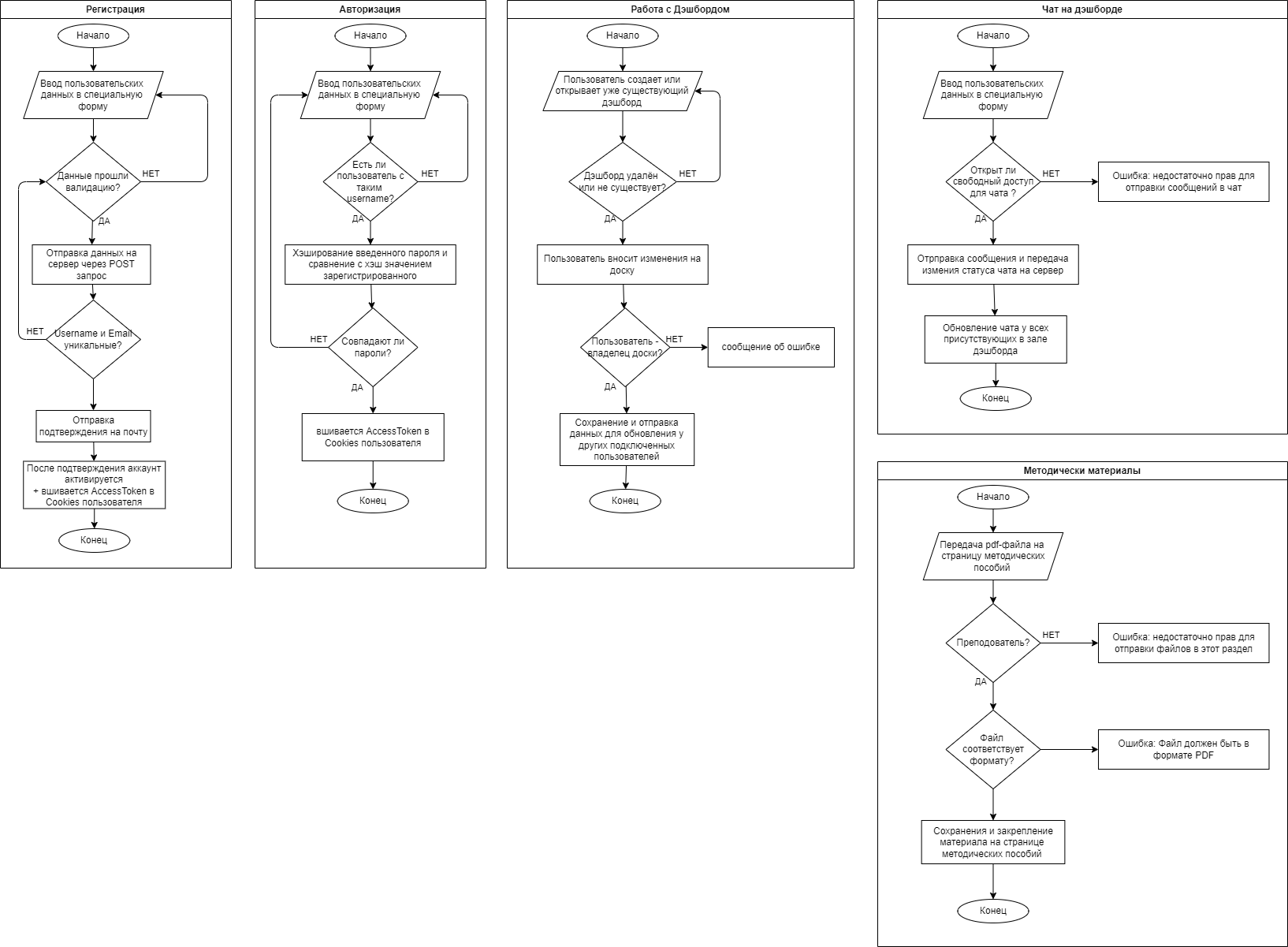


Рисунок 2.3 — Схема алгоритма аутентификация пользователя

## 2.4 Алгоритм работы интерактивной доски

Алгоритм работы интерактивной доски реализует совместную работу пользователей в реальном времени, позволяя рисовать, добавлять текст и фигуры на виртуальной доске. Когда пользователь открывает доску, устанавливается соединение, обеспечивающее синхронизацию изменений между всеми участниками.

Шаги алгоритма создания отклика на заказ:

1. Пользователь открывает или создаёт доску на дашборде, устанавливая Websockets-соединение. Сервер загружает текущее состояние доски из PostgreSQL.
2. Пользователь вносит изменения через JS-интерфейс. Изменения отправляются через Websockets на сервер.
3. Сервер рассылает обновления всем подключённым клиентам через Websockets. Данные сохраняются в базе данных для последующего доступа.
4. Алгоритм синхронизации предотвращает конфликты при одновременных изменениях.
5. При закрытии доски соединение разрывается, а данные сохраняются.

Схема алгоритма представлена на рисунке 2.4.

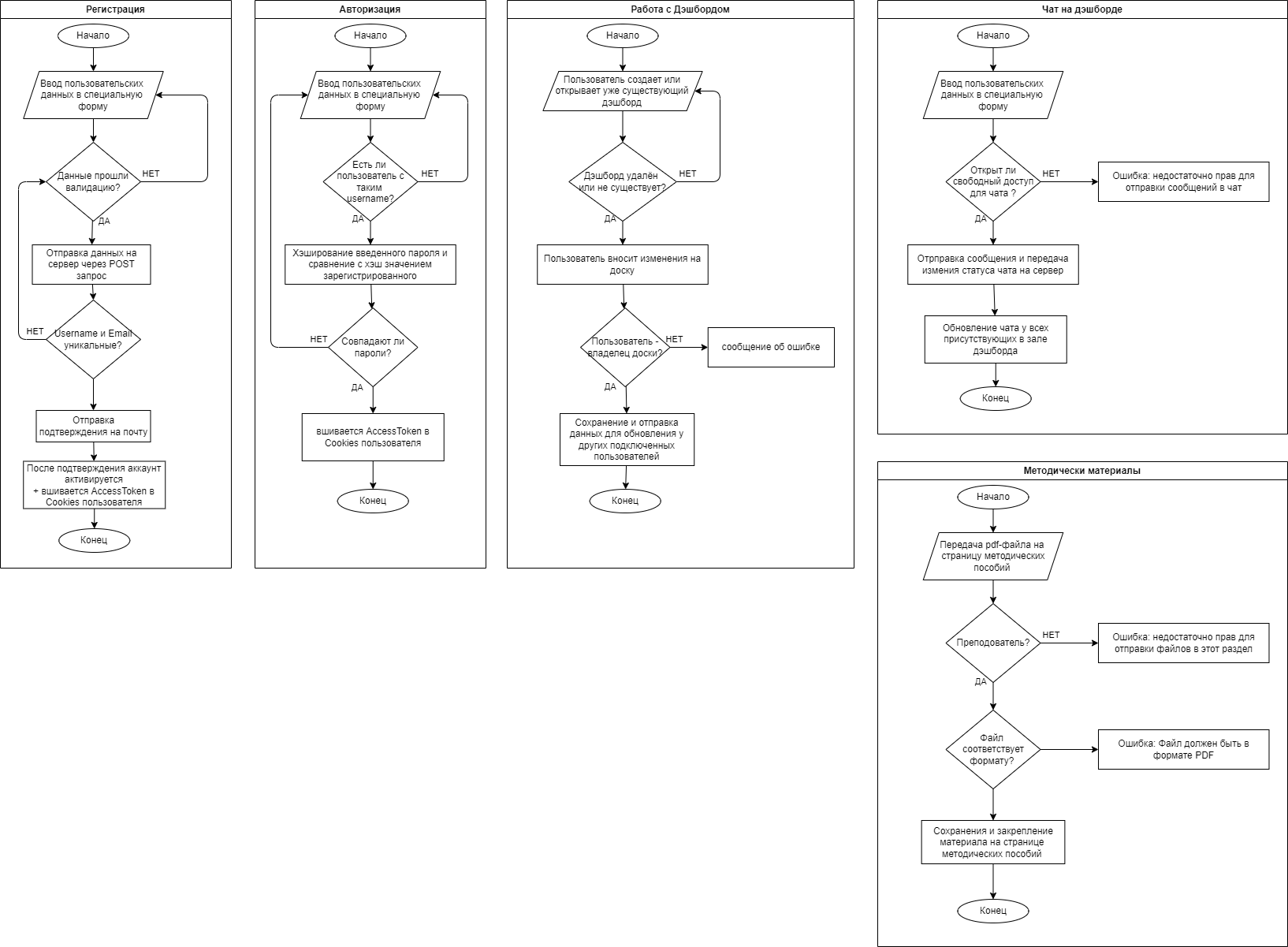


Рисунок 2.4 — Схема алгоритма работы интерактивной доски

## 2.5 Алгоритм чата на дашборде

Алгоритм работы чата реализует обмен сообщениями между пользователями, доступный только на вкладке дашборда, обеспечивая удобное общение в контексте учебного процесса. Когда пользователь открывает дашборд, устанавливается соединение для передачи сообщений в реальном времени.

Алгоритм:

1. Инициализация: при открытии дашборда клиент устанавливает Websockets-соединение для чата. Сервер загружает историю сообщений из PostgreSQL.
2. Отправка сообщений: пользователь вводит сообщение, которое отправляется через Websockets. Сервер проверяет права доступа и сохраняет сообщение в базе данных.
3. Рассылка: сообщение передаётся всем пользователям, подключённым к чату, через Websockets. Сообщения отображаются в реальном времени.
4. Ограничение доступа: чат отключается при переходе на другие вкладки.

Схема алгоритма представлена на рисунке 2.5.

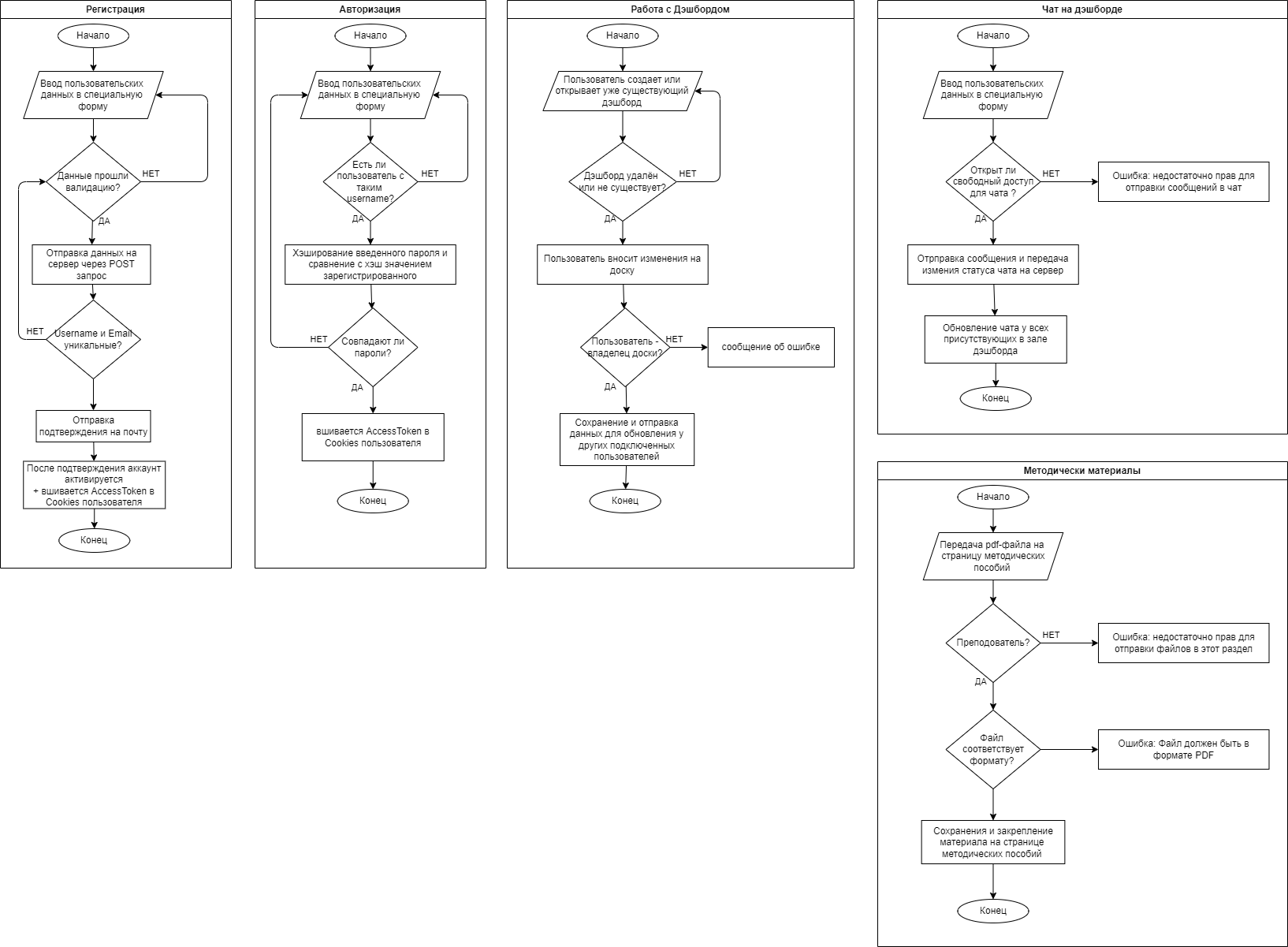


Рисунок 2.5 — Схема алгоритма работы интерактивной доски

## 2.6 Алгоритм управления методическими материалами

Алгоритм управления методическими материалами реализует загрузку, хранение и предоставление доступа к PDF-файлам, используемым в учебном процессе. Когда преподаватель загружает материал, он сохраняется на сервере, а студенты могут получить к нему доступ. Алгоритм:

1. Преподаватель загружает PDF через страницу методических материалов. Файл отправляется через POST-запрос к FastAPI.
2. Сервер проверяет формат и размер файла. Файл сохраняется в хранилище, а метаданные — в PostgreSQL.
3. Студенты запрашивают список материалов через GET-запрос. Сервер возвращает ссылки на файлы для скачивания.

Схема алгоритма представлена на рисунке 2.6.

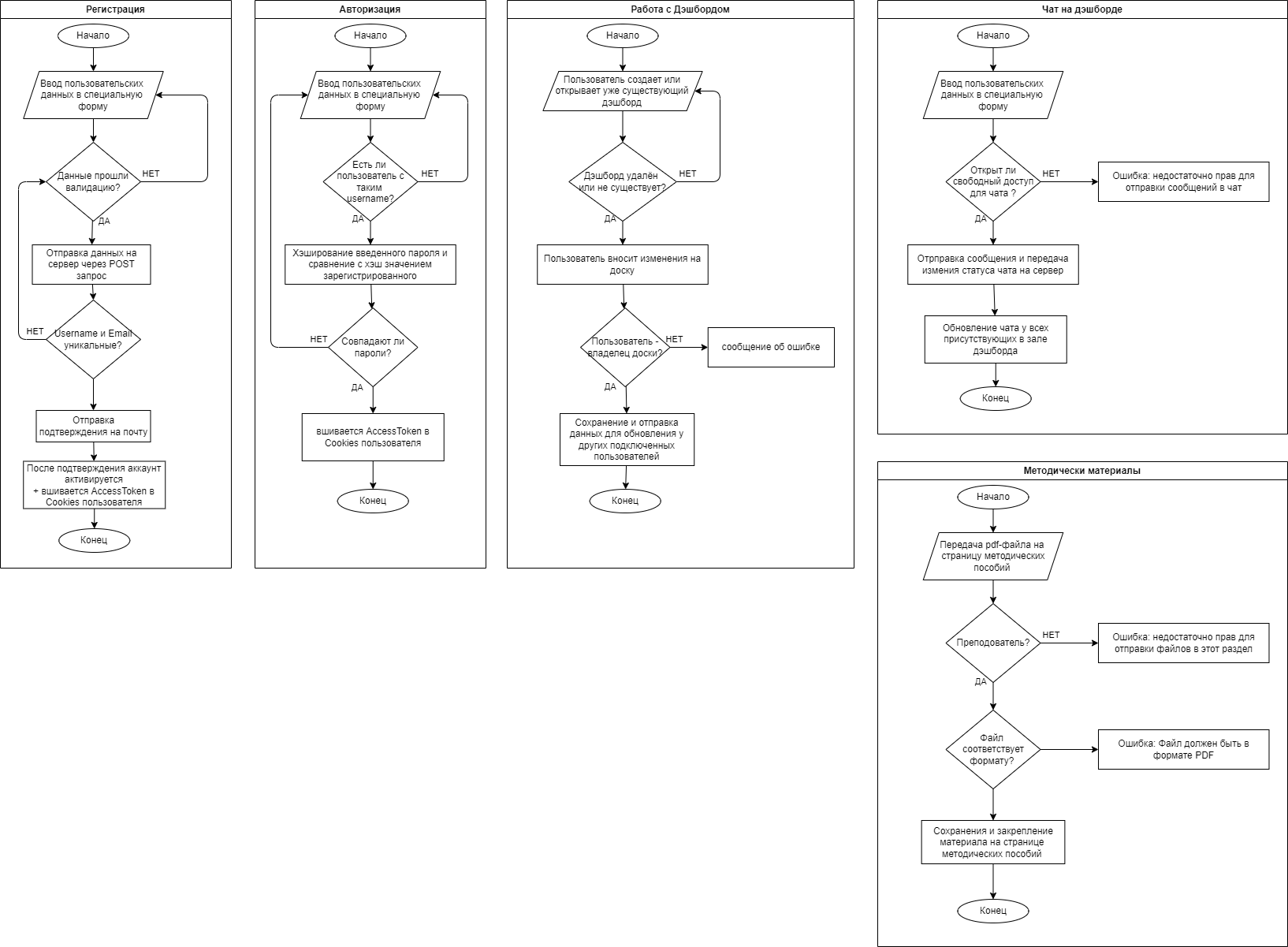


Рисунок 2.6 — Схема алгоритма работы интерактивной доски

## 2.7 Алгоритм создания и прохождения тестов

Алгоритм создания и прохождения тестов реализует функционал для преподавателей по созданию тестов и для студентов по их прохождению. Когда преподаватель создаёт тест, он сохраняется на сервере, а студенты могут пройти его и получить результаты. Алгоритм:

1. Создание теста: преподаватель вводит вопросы, варианты ответа и правильные ответы через интерфейс на фронтенде. Данные отправляются через POST-запрос к FastAPI и сохраняются в PostgreSQL.
2. Прохождение теста: студент открывает тест, получая вопросы через GET-запрос. После выбора ответов данные отправляются через POST-запрос на сервер.
3. Оценка: сервер сравнивает ответы с правильными, вычисляет результат и сохраняет его в PostgreSQL. Результат отображается студенту и сохраняется в его личном кабинете.

Рисунок 2.7 — Схема алгоритма работы интерактивной доски

## 2.8 Алгоритм обработки домашних заданий

Алгоритм Алгоритм обработки домашних заданий реализует управление заданиями, включая их создание, отправку решений и оценку. Когда преподаватель создаёт задание, оно становится доступным для студентов, которые могут отправить свои решения. Алгоритм:

1. Создание задания: преподаватель вводит текст задания, дедлайн и прикрепляет файлы (если необходимо) через интерфейс. Данные отправляются через POST-запрос и сохраняются в PostgreSQL.
2. Отправка решения: студент загружает решение (текст или файлы) через HTML-форму, отправляя данные через POST-запрос. Сервер сохраняет решение и отправляет уведомление преподавателю через RabbitMQ.
3. Проверка: преподаватель оценивает решение через интерфейс, вводя оценку и комментарии. Оценка сохраняется в PostgreSQL и отображается в личном кабинете студента.

Рисунок 2.8 — Схема алгоритма обработки домашних заданий

## 2.9 Вывод по главе

В данном разделе описаны ключевые алгоритмы, обеспечивающие работу обучающей платформы. Рассмотрены процессы клиент-серверного взаимодействия, регистрации, аутентификации, работы интерактивной доски, чата, управления материалами, тестами и заданиями. Использование асинхронных технологий FastAPI, Websockets, RabbitMQ и PostgreSQL обеспечивает высокую производительность и масштабируемость. Эти алгоритмы станут основой для реализации платформы, описанной в последующих главах.

.

# 3 Программное конструирование

В данном разделе приведено обоснование выбора средств разработки и СУБД (системы управления базами данных), описывается объектно-ориентированная модель разрабатываемой платформы и создание баз данных.

## 3.1 Обоснование выбора средств разработки

Для программной реализации программного средства были выбраны среды разработки VS Code (Visual Studio Code): интегрированная среда разработки программного обеспечения, разработанная компанией Microsoft. Она поддерживает множество языков программирования и предлагает широкие возможности для расширения функциональности через плагины[3].

К достоинствам VS Code можно отнести:

* эргономичная среда: среда разработки обеспечивает удобный и интуитивно понятный интерфейс, который легко настраивается под нужды разработчика;
* легкий старт и сочетания клавиш для любых действий: VS Code предоставляет множество сочетаний клавиш, которые ускоряют процесс разработки;
* широкий спектр расширений и плагинов: программа поддерживает множество расширений, которые добавляют функциональность для различных языков программирования и инструментов разработки;
* глубокий анализ кода: Visual Studio Code включает встроенные инструменты анализа кода, которые помогают выявлять и исправлять ошибки на ранних стадиях разработки;
* предоставляет контекстные подсказки и действия, которые упрощают редактирование и рефакторинг кода;
* умное автодополнение кода: поддерживает интеллектуальное автодополнение, что ускоряет написание кода и уменьшает количество ошибок.

Для разработки образовательной платформы были выбраны следующие инструменты и технологии:

* Язык программирования Python выбран как основной благодаря его простоте, поддержке объектно-орицентированного программирования и наличию большого количества библиотек для различных задач [4].
* Фреймворк FastAPI для создания асинхронного REST API, обеспечивающего высокую производительность и поддержку функций в реальном времени [5].
* Websockets для реализации интерактивных функций, таких как интерактивная доска и чат, требующих обмена данными в реальном времени [6].
* SQLAlchemy ORM-библиотека для работы с базами данных в Python, упрощающая взаимодействие с СУБД и поддерживающая объектно-ориентированное программирование [7].
* Alembic для управления миграциями базы данных, обеспечивающего целостность и актуальность структуры данных [8].
* RabbitMQ для асинхронной обработки задач, таких как отправка уведомлений или обработка больших файлов, что повышает масштабируемость системы [9].

Для фронтенд части платформы были выбраны следующие инструменты:

* HTML, JavaScript (JS), CSS стандартный набор технологий для создания динамичного и отзывчивого пользовательского интерфейса.
* Фреймворк React для построения интерактивного интерфейса с компонентным подходом, обеспечивающим удобство разработки и поддержку состояний [10].

Для выбираемого языка программирования важным фактором являлась поддержка строгой типизации, объектно-ориентированность, простота и безопасность. Все вышеперечисленные достоинства послужили весомой причиной для того, чтобы выбрать Python в качестве языка программирования для серверной части.

Для развёртывания базы данных была выбрана программа PostgreSQL надёжная и масштабируемая СУБД с открытым исходным кодом, поддерживающая сложные запросы и транзакции, что идеально подходит для образовательной платформы с большим количеством пользователей и данных [11].

Для реализации обеспечения изоляции и портативности среды выполнения платформы используется Docker. Это позволяет нам легко разворачивать и тестировать приложение в различных окружениях, минимизируя проблемы, связанные с несовместимостью зависимостей и конфигураций.

Docker – это платформа для разработки, доставки и эксплуатации приложений, основанная на контейнеризации. Контейнеры позволяют разработчикам упаковывать приложение и его зависимости в единую портативную единицу, которая может быть запущена в любом окружении. Это значительно упрощает процесс разработки, тестирования и развертывания приложений, обеспечивая консистентность и предсказуемость их поведения на различных платформах.

Основные компоненты Docker включают:

1. Docker Engine: основная часть Docker, отвечающая за создание и управление контейнерами.
2. Docker Images: файлы, содержащие все необходимые компоненты для запуска приложения, включая код, зависимости, библиотеки и конфигурации.
3. Docker Containers: запущенные экземпляры Docker Images, которые изолированы друг от друга и от хоста.
4. Docker Hub: облачный сервис для хранения и обмена Docker Images.

Эти технологии обеспечивают надёжную, масштабируемую и удобную в разработке платформу, соответствующую требованиям проекта, включая интерактивную доску с отображением изменений в реальном времени, чат на вкладке дэшборда, хранение методических материалов в виде PDF-архивов, создание и выкладывание тестов, управление домашними заданиями, а также реализацию личного кабинета, регистрации и авторизации.

## 3.2 Объектно-ориентированная модель программного средства

Программное средство разработано с использованием объектно-ориентированной парадигмы программирования и состоит из множества классов, которые взаимодействуют между собой для обеспечения функциональности различных компонентов. Ниже представлена более детальная структура основных компонентов программы:

Классы серверной части:

* класс Dashboard: представляет собой центральный компонент для взаимодействия с интерактивной доской, осуществляя взаимодействие в реальном времени между преподавателем и учениками;
* класс User: представляет модель пользователя и содержит поля и методы для работы с данными пользователя в базе данных;
* класс Quiz: представляет модель тестов составляемых преподавателем для конкретных групп, содержит поля и методы для работы с данными пользователя в базе данных;
* класс Manual: представляет модель методических материалов, также содержит поля и методы для работы с базой данных;
* класс Homework: представляет модель для домашних заданий, содержит необходимые поля и методы для работы с базой данных.

Каждый класс представляет определенный функциональный блок и содержит методы и свойства, необходимые для выполнения своих задач. Взаимодействие между классами происходит посредством вызова методов, передачи параметров и обмена данными.

Такая объектно-ориентированная модель программного средства позволяет легко расширять и модифицировать функциональность, а также обеспечивает более понятную и структурированную разработку и поддержку кода.

### 3.2.1 Класс Dashboard

Класс Dashboard используется для хранения информации о интерактивной доске и чатах в базе данных.

Поля класса:

* поле id (тип: Integer): уникальный идентификатор доски;
* поле dashboard\_id(тип: String): строковый уникальный идентификатор дашборда, генерируемый как UUID;;
* поле title (тип: String): название дашборда, задаваемое пользователем;
* поле user\_id (тип: String): идентификатор пользователя, создавшего дашборд;
* поле group\_id (тип: String): идентификатор группы, к которой привязан дашборд;;
* поле data (тип: JSON): данные интерактивной доски (координаты, фигуры) и сообщений чата;
* поле created\_at (тип: Datetime): дата и время создания дашборда;
* поле updated\_at (тип: Datetime): данные интерактивной доски дата и время последнего обновления дашборда.

В таблице 3.1 представлены методы класса Dashboard.

Таблица 3.1 — Методы класса Dashboard

| Название | Входные параметры | Описание метода | Возвращаемое значение | |
| --- | --- | --- | --- | --- |
| add | session: AsyncSession, \*\*data | Создаёт новый дашборд с указанными данными (название, пользователь, группа, состояние доски). | Dashboard | |
| get\_per\_id | id: int, session: AsyncSession | Возвращает дашборд по его уникальному идентификатору (id). | Dashboard или None | |
| get | session: AsyncSession, \*\*creterias | Возвращает список дашбордов, соответствующих заданным критериям (например, dashboard\_id). | List[Dashboard] | |
| get\_all\_per\_id | id: int, session: AsyncSession, \*\*kwargs | Возвращает все дашборды, связанные с указанным пользователем (user\_id). | | List[Dashboard] |
| get\_all | session: AsyncSession | Возвращает все существующие дашборды в базе данных. | | List[Dashboard] |
| update | id: int, session: AsyncSession, \*\*updateting\_data | Обновляет данные дашборда (например, состояние доски или сообщения чата). | | Dashboard, str |

### 3.2.2 Класс User

Класс User представляет собой модель пользователя и используется для хранения и изменения информации о пользователях в базе данных.

Поля класса:

* поле id (тип: Integer): уникальный идентификатор пользователя;
* поле username (тип: String): имя пользователя, уникальное;
* поле email (тип: String): электронная почта пользователя, уникальная;
* поле password (тип: String): хешированный пароль пользователя;
* поле role (тип: Enum): роль пользователя (STUDENT или TEACHER);
* поле created\_at (тип: Datetime): дата и время создания учетной записи;
* поле updated\_at (тип: Datetime): дата и время последнего обновления учетной записи.

В таблице 3.2 представлены методы класса User.

Таблица 3.2 — Методы класса User

| Название | Входные параметры | Описание метода | Возвращаемое значение |
| --- | --- | --- | --- |
| add | session: AsyncSession, \*\*data | Создаёт нового пользователя с указанными данными (имя, email, пароль, роль). | User |
| get\_per\_id | id: int, session: AsyncSession | Возвращает пользователя по его уникальному идентификатору (id). | User или None |
| get | session: AsyncSession, \*\*creterias | Возвращает список пользователей, соответствующих критериям (например, username). | List[User] |
| get\_all\_per\_id | id: int, session: AsyncSession, \*\*kwargs | Возвращает всех пользователей, связанных с указанным идентификатором (не используется). | List[User] |
| get\_all | session: AsyncSession | Возвращает всех пользователей в базе данных. | List[User] |
| update | id: int, session: AsyncSession, \*\*updateting\_data | Обновляет данные пользователя (например, email или роль). | User, str |

### 3.2.3 Класс Quiz

Класс Quiz представляет модель теста, используемого для создания и прохождения тестов преподавателями и студентами.

Поля класса:

* поле id (тип: Integer): уникальный идентификатор теста;
* поле user\_id (тип: Integer): идентификатор пользователя, создавшего тест;
* поле title (тип: String): название теста;
* поле description (тип: String): описание теста;
* поле timer (тип: Integer): время на выполнение теста в секундах;
* поле quiz\_description (тип: JSON): вопросы теста в формате JSON;
* поле quiz\_answers (тип: JSON): правильные ответы теста в формате JSON;
* поле created\_at (тип: Datetime): дата и время создания теста;
* поле updated\_at (тип: Datetime): дата и время последнего обновления теста.

Методы класса представлены в таблице 3.3.

Таблица 3.3 — Методы класса Quiz

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| add | session: AsyncSession, \*\*data | Создаёт новый тест с указанными данными (вопросы, ответы, время). | Quiz |
| get\_per\_id | id: int, session: AsyncSession | Возвращает тест по его уникальному идентификатору (id). | Quiz или None |
| get | session: AsyncSession, \*\*creterias | Возвращает список тестов, соответствующих критериям (например, user\_id). | List[Quiz] |
| get\_all\_per\_id | id: int, session: AsyncSession, \*\*kwargs | Возвращает все тесты, созданные указанным пользователем (user\_id). | List[Quiz] |
| get\_all | session: AsyncSession | Возвращает все тесты в базе данных. | List[Quiz] |

### 3.2.4 Класс Manual

Класс Manual представляет модель методического материала, хранящего PDF-файлы, доступные для групп.

Поля класса:

* поле id (тип: Integer): уникальный идентификатор материала;
* поле group\_id (тип: Integer): идентификатор группы, к которой привязан материал;
* поле title (тип: String): название материала;
* поле description (тип: String): описание материала;
* поле filepath (тип: String): путь к PDF-файлу на сервере;
* поле created\_at (тип: Datetime): дата и время загрузки материала;
* поле updated\_at (тип: Datetime): дата и время последнего обновления материала.

Методы класса представлены в таблице 3.4.

Таблица 3.4 — Методы класса Manual

| Название | Входные параметры | Описание метода | Возвращаемое значение |
| --- | --- | --- | --- |
| add | session: AsyncSession, \*\*data | Создаёт новый методический материал с указанными данными (файл, название). | Manual |
| get\_per\_id | id: int, session: AsyncSession | Возвращает материал по его уникальному идентификатору (id). | Manual или None |
| get | session: AsyncSession, \*\*creterias | Возвращает список материалов, соответствующих критериям (например, group\_id). | List[Manual] |
| get\_all\_per\_id | id: int, session: AsyncSession, \*\*kwargs | Возвращает все материалы, связанные с указанной группой (group\_id). | List[Manual] |
| get\_all | session: AsyncSession | Возвращает все материалы в базе данных. | List[Manual] |
| update | id: int, session: AsyncSession, \*\*updateting\_data | Обновляет данные материала (например, описание или путь к файлу). | Manual, str |

## 3.2.5 Класс Homework

Класс Homework представляет модель домашнего задания, отправленного студентом и привязанного к тесту или материалу.

Поля класса:

* поле id (тип: Integer): уникальный идентификатор задания;
* поле user\_id (тип: Integer): идентификатор студента, отправившего задание;
* поле quiz\_id (тип: Integer): идентификатор теста, к которому относится задание;
* поле created\_at (тип: Datetime): дата и время отправки задания;
* поле updated\_at (тип: Datetime): дата и время последнего обновления задания.

Методы класса представлены в таблице 3.5.

Таблица 3.5 — Методы класса

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| add | session: AsyncSession, \*\*data | Создаёт новое домашнее задание с указанными данными (студент, тест). | Homework |
| get\_per\_id | id: int, session: AsyncSession | Возвращает задание по его уникальному идентификатору (id). | Homework или None |
| get | session: AsyncSession, \*\*creterias | Возвращает список заданий, соответствующих критериям (например, user\_id). | List[Homework] |
| get\_all\_per\_id | id: int, session: AsyncSession, \*\*kwargs | Возвращает все задания, отправленные указанным студентом (user\_id). | List[Homework] |
| get\_all | session: AsyncSession | Возвращает все задания в базе данных. | List[Homework] |
| update | id: int, session: AsyncSession, \*\*updateting\_data | Обновляет данные задания (например, статус или оценка). | Homework, str |

## 3.3 Описание структуры баз данных

В данном разделе описаны таблицы базы данных образовательной платформы, реализованной с использованием PostgreSQL и ORM SQLAlchemy. Каждая таблица соответствует сущности, необходимой для реализации функционала: интерактивной доски с отображением изменений в реальном времени, чата на вкладке дашборда, хранения методических материалов в формате PDF, создания и прохождения тестов, управления домашними заданиями, личного кабинета, регистрации и авторизации пользователей. Для каждой таблицы приведены атрибуты, их типы данных, домены и возвращаемые значения в контексте использования в приложении.

### 3.3.1 База данных «dashboards.db»

Объект «dashboard» представляет собой базу данных дашбордов. Данный объект хранит в себе следующие атрибуты:

* атрибут «id» — уникальный идентификатор дашборда;
* атрибут «dashboard\_id» — строковый идентификатор дашборда;
* атрибут «title» — название дашборда;
* атрибут «user\_id» — идентификатор пользователя, создавшего дашборд;
* атрибут «group\_id» — идентификатор группы, к которой привязан дашборд;
* атрибут «data» — данные интерактивной доски и чата;
* атрибут «created\_at» — дата и время создания дашборда;
* атрибут «updated\_at» — дата и время последнего обновления дашборда.

Таблица 3.6 — Объект «dashboards»

| Имя атрибута | Домен атрибута |
| --- | --- |
| id | int not null auto\_increment |
| dashboard\_id | int not null |
| title | text not null |
| user\_id | text |
| group\_id | text |
| data | text |
| created\_at | text |
| updated\_at | datetime |

Ограничения:

* поле id — первичный ключ (уникальные значения);
* поле dashboard\_id — уникальное значение;
* поле user\_id — внешний ключ, ссылается на id из таблицы users;
* поле group\_id — внешний ключ, ссылается на id из таблицы groups.

### 3.3.2 База данных «users.db»

Объект «users» представляющий собой базу данных пользователей.

Данный объект хранит в себе следующий атрибуты:

* атрибут «id» — уникальный идентификатор пользователя;
* атрибут «username» — имя пользователя;
* атрибут «email» — электронная почта пользователя;
* атрибут «password» — хешированный пароль пользователя;
* атрибут «role» — роль пользователя;
* атрибут «created\_at» — дата и время создания учетной записи;
* атрибут «updated\_at» — дата и время последнего обновления учетной записи.

Основные поля базы данных представлены в таблицe 3.7.

Таблица 3.7 — Объект «users»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | int not null auto\_increment |
| username | text not null unique |
| email | text not null unique |
| password | text not null |
| role | text not null |
| created\_at | datetime |
| updated\_at | datetime |

Ограничения:

* поле «id» — первичный ключ (уникальные значения);
* поля «username», «email» — уникальные значения;
* поле «role» — ограничено значениями (STUDENT, TEACHER).

## 3.3.3 База данных «quizzes»

Объект «quiz» представляет собой базу данных тестов. Данный объект хранит в себе следующие атрибуты:

* атрибут «id» — уникальный идентификатор теста;
* атрибут «user\_id» — идентификатор пользователя, создавшего тест;
* атрибут «title» — название теста;
* атрибут «description» — описание теста;
* атрибут «timer» — время на выполнение теста;
* атрибут «quiz\_description» — вопросы теста;
* атрибут «quiz\_answers» — правильные ответы теста;
* атрибут «created\_at» — дата и время создания теста;
* атрибут «updated\_at» — дата и время последнего обновления теста.

Основные поля базы данных представлены в таблице 3.3.

Таблица 3.8 — Объект «quiz»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| Id | int not null auto\_increment |
| user\_id | int not null |
| Title | text |
| Description | text |
| Timer | int |
| quiz\_description | json |
| quiz\_answers | json |
| created\_at | datetime |
| updated\_at | datetime |

**Ограничения:**

* поле «id» — первичный ключ (уникальные значения);
* поле «user\_id» — внешний ключ, ссылается на «id» из таблицы «users».

## 3.3.4 База данных «manuals»

Объект «manual» представляет собой базу данных методических материалов. Данный объект хранит в себе следующие атрибуты:

* атрибут «id» — уникальный идентификатор материала;
* атрибут «group\_id» — идентификатор группы, к которой привязан материал;
* атрибут «title» — название материала;
* атрибут «description» — описание материала;
* атрибут «filepath» — путь к PDF-файлу;
* атрибут «created\_at» — дата и время загрузки материала;
* атрибут «updated\_at» — дата и время последнего обновления материала.

Основные поля базы данных представлены в таблице 3.4.

Таблица 3.9 — Объект «manual»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | int not null auto\_increment |
| group\_id | int not null |
| title | text |
| description | text |
| filepath | text |
| created\_at | datetime |
| updated\_at | datetime |

**Ограничения:**

* поле «id» — первичный ключ (уникальные значения);
* поле «group\_id» — внешний ключ, ссылается на «id» из таблицы «groups».

## 3.3.5 База данных «homeworks»

Объект «homework» представляет собой базу данных домашних заданий. Данный объект хранит в себе следующие атрибуты:

* атрибут «id» — уникальный идентификатор домашнего задания;
* атрибут «user\_id» — идентификатор студента, отправившего задание;
* атрибут «quiz\_id» — идентификатор теста, к которому относится задание;
* атрибут «created\_at» — дата и время отправки задания;
* атрибут «updated\_at» — дата и время последнего обновления задания.

Основные поля базы данных представлены в таблице 3.10.

Таблица 3.10 — Объект «homework»

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | int not null auto\_increment |
| user\_id | int not null |
| quiz\_id | int not null |
| created\_at | datetime |
| updated\_at | datetime |

Ограничения:

* поле «id» — первичный ключ (уникальные значения);
* поле «user\_id» — внешний ключ, ссылается на «id» из таблицы «users»;
* поле «quiz\_id» — внешний ключ, ссылается на «id» из таблицы «quizzes».

## **3.5 Вывод по главе**

В данной главе были обоснованы выбор языка программирования и среды разработки для создаваемого программного средства, выделены их преимущества применительно к конкретному случаю, обоснована общая структура работы программного средства. Были описаны основные функциональные компоненты и классы приложения. Для классов были приведены поля и методы с указанием параметров. Для каждого из классов и функциональных компонентов, было описано его предназначение. Был обоснован выбор СУБД и описаны структуры используемых баз данных.

# 4 Тестирование программного средства

В данном разделе продемонстрированы основные этапы тестирования образовательной платформы, реализованной с использованием FastAPI, PostgreSQL и React. Описаны сценарии взаимодействия пользователя с интерфейсом, включая регистрацию, авторизацию, работу с тестами, дашбордом (интерактивная доска и чат), методическими материалами, домашними заданиями и личным кабинетом..

## 4.1 Описание процесса тестирования

При первом входе на платформу пользователь сталкивается с главной страницей, содержащей кнопки «Войти» и «Зарегистрироваться», изображенной на рисунке 4.1. Нажатие на кнопку «Зарегистрироваться» перенаправляет пользователя на форму регистрации, где требуется ввести имя пользователя, электронную почту и пароль, как показано на рисунке 4.2. Поля необязательные (например, имя или возраст) можно пропустить, чтобы заполнить их позже в личном кабинете.

Рисунок 4.1 — Начальное сообщение

Рисунок 4.2 — Прохождение регистрации

После успешной регистрации пользователь перенаправляется на страницу авторизации, где вводит имя пользователя и пароль, изображенную на рисунке 4.3. При успешной авторизации открывается личный кабинет, содержащий меню с доступными функциями: просмотр дашбордов, тестов, домашних заданий, методических материалов и профиля, как показано на рисунке 4.4.

Рисунок 4.3 — Новое начальное меню после регистрации

Рисунок 4.4 — Детали профиля

При выборе раздела «Тесты» пользователь попадает на страницу со списком доступных тестов, изображенную на рисунке 4.5. Для преподавателей доступна кнопка «Создать тест», которая открывает форму для ввода названия, описания, времени выполнения и вопросов теста, как показано на рисунке 4.6. После создания теста он становится доступным для студентов.

Рисунок 4.5 — Меню действий одной из платформ

Рисунок 4.6 — Поступившие отклики

Студент может выбрать тест из списка и нажать кнопку «Решать», что открывает страницу прохождения теста, изображенную на рисунке 4.7. На этой странице отображаются вопросы, варианты ответа и таймер, если он задан. После завершения теста результаты отправляются на сервер, и пользователь получает уведомление об успешной отправке, как показано на рисунке 4.8. В случае ошибки (например, истекло время) отображается сообщение об ошибке, изображенное на рисунке 4.9.

Рисунок 4.7 — Сгенерированный ответ

При переходе в раздел «Дашборды» пользователь видит список доступных дашбордов, изображенный на рисунке 4.10. Выбор дашборда открывает интерактивную доску и чат, реализованные через WebSockets, как показано на рисунке 4.11. Пользователь может рисовать на доске, видеть изменения в реальном времени и отправлять сообщения в чат, которые отображаются всем участникам группы.

Рисунок 4.8 — Введенный вручную ответ

В разделе «Методические материалы» преподаватель может загрузить PDF-файл, указав название и описание, как изображено на рисунке 4.12. Студенты видят список материалов, доступных для их группы, и могут скачать файлы, как показано на рисунке 4.13.

Рисунок 4.9 — Введенный вручную ответ

Раздел «Домашние задания» позволяет студентам отправлять выполненные задания, привязанные к тестам, как изображено на рисунке 4.14. Преподаватель может просмотреть отправленные работы и выставить оценки, как показано на рисунке 4.15.

Рисунок 4.10 — Введенный вручную ответ

В личном кабинете пользователь может редактировать профиль, включая имя, возраст и статус, как изображено на рисунке 4.16. После сохранения изменений отображается уведомление об успешном обновлении, как показано на рисунке 4.17.

Рисунок 4.11 — Успешная отправка отклика

Рисунок 4.12 — Ошибка при отправке

## 4.2 Выводы по главе

В данной главе было произведено последовательное знакомство с функционалом образовательной платформы с последующим его тестированием. Были протестированы механизмы регистрации и авторизации, создание и прохождение тестов, работа с интерактивной доской и чатом, загрузка и просмотр методических материалов, отправка и проверка домашних заданий, а также редактирование профиля в личном кабинете. На основании проведенного тестирования можно сделать вывод, что основные функции реализованы в соответствии с поставленными требованиями.

# 5. Безопасность и экологичность

В данной работе представлена разработка сервиса для автоматических откликов на фриланс заказы.

В разработке программного продукта уделяется приоритетное внимание соблюдению требований и нормативов в области безопасности труда. Подробное изложение мероприятий, направленных на выявление и нейтрализацию возможных угроз безопасности пользователей, представлено в данной главе.

Основополагающим аспектом безопасности на всех этапах разработки, внедрения и эксплуатации любого производственного продукта является создание условий, способствующих сохранению здоровья и высокой производительности труда.

Полностью избежать рисков в производственной среде невозможно. В контексте работы с программным обеспечением следует учитывать как вредные воздействия, связанные с использованием персональных электронно-вычислительных устройств, так и эргономические факторы, значительно влияющие на трудоспособность. Соблюдение установленных норм безопасности и санитарных требований, а также выполнение законодательных актов позволяет минимизировать негативные воздействия и снизить риски заболеваний, при этом обеспечивая комфорт и высокую производительность труда.

## 5.1 Анализ опасных и вредных производственных факторов и мероприятия по обеспечению безопасных и безвредных условий труда

Для проведения анализа и оценки безопасности помещения, в котором будет применяться разработанное программное обеспечение, необходимо изобразить план самого помещения. Данный план представлен на рисунке 5.1.

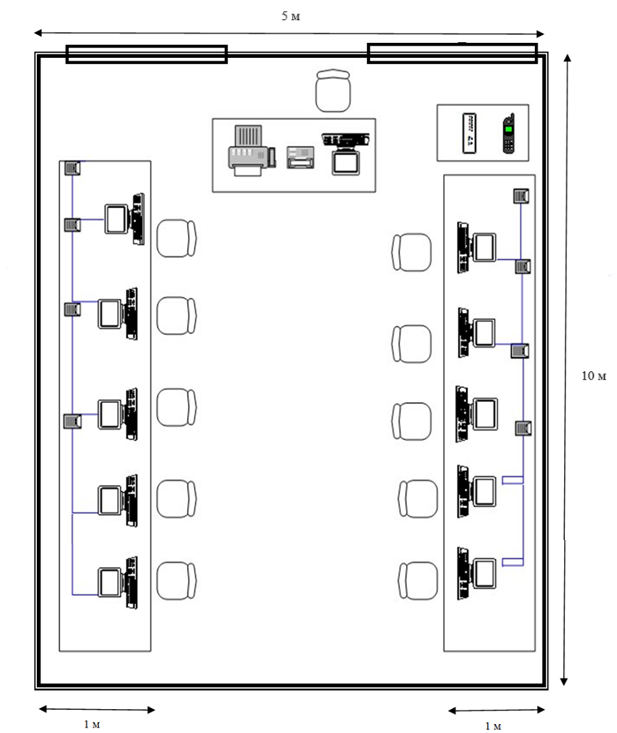


Рисунок 5.1 — План помещения

Параметры данного помещения составляют:

* длина 10 м;
* ширина 5 м;
* высота 3 м;
* площадь 50 м2;
* объем 150 м3.

Также помещение оборудовано 11 электронными вычислительными машинами, тремя столами, одиннадцатью стульями, десятью трубчатыми люминесцентными лампами, одним принтером, кондиционером. Освещение совмещенное, естественный свет от двух окон обеспечивает освещение днем, а также имеются четыре лампы.

Для обеспечения безопасных условий при работе с программным обеспечением в данном помещении был произведен анализ возможных производственных факторов, воздействующих на человека в таблице 5.1.

Таблица 5.1 — Анализ опасных и вредных производственных факторов

| № п/п | Опасные и вредные производственные факторы | Источник ОВПФ | Действие на организм человека | Мероприятия по устранению действия фактора | Нормативно -технический документ |
| --- | --- | --- | --- | --- | --- |
| 1 | Повышенный или пониженный уровень освещенности | Люминесцентные лампы и оконные проемы | Боль в глазах, повышенная утомляемость | Установка или удаление источников освещение, оборудование оконных проемов | СанПин 2.2.2/2.4.1340-03 |
| 2 | Повышенный или пониженный уровень статического электричества | ПЭВМ | Раздражение чувствительных нервных окончаний кожи | Установка заземления | СанПин 2.2.2/2.4.1340-03 |
| 3 | Повышенная или пониженная влажность воздуха | Кондиционер | Перегрев тела, обезвоживание, сухость слизистых оболочек | Установка систем увлажнения, осушения воздуха | СанПин 2.2.2/2.4.1340-03 |
| 4 | Повышенная или пониженная температура воздуха | Кондиционер, система отопления | Перегрев тела, простудные заболевания | Установка система кондиционирования воздуха | СанПин 2.2.2/2.4.1340-03 |
| 5 | Чрезмерная запыленность воздуха | Кондиционер, оконные проемы | Респираторные заболевания, аллергические реакции | Установка фильтров в системы кондиционирования | СанПин 2.2.2/2.4.1340-03 |
| 6 | Повышенный уровень электромагнитного излучения | ПЭВМ | Функциональные изменения деятельности нервной системы, эндокринной системы | Рациональное распределение проводки, расстановка ПЭВМ на более дальнем расстоянии друг от друга | СанПин 2.2.2/2.4.1340-03 |
| 7 | Нервно-эмоциональные перегрузки | ПЭВМ | Усталость, истощение, головные боли | Рациональное планирование рабочего графика | СанПин 2.2.2/2.4.1340-03 |

## 5.2 Экологичность проекта

Экологическая безопасность в рамках организации представляет собой комплекс мероприятий, направленных на поддержание экологического благополучия и улучшение экономической эффективности. Внедрение материалосберегающих технологий не только повышает производственную эффективность предприятия, но и способствует снижению воздействия вредных веществ на здоровье работников и окружающую среду.

Отходы, возникающие в процессе производства, могут включать как относительно безвредные остатки пищи и натуральных материалов, так и более проблемные материалы, такие как стекло, пластик и синтетические вещества. Большая часть этих отходов подлежит повторной переработке, в результате чего получаются материалы, пригодные для повторного использования.

Переработка компьютерной техники и других устройств проводится в строгом соответствии с экологическим законодательством. Процесс подготовки к утилизации включает списание оборудования с баланса и оформление необходимых документов. От списания до фактической утилизации, устройства не могут быть утилизированы как бытовые отходы из-за риска выделения токсичных веществ.

Драгоценные металлы, содержащиеся в комплектующих, отправляются на аффинажные заводы, а пластиковые компоненты перерабатываются во вторичное сырьё для использования в промышленности. Неисправные устройства могут быть демонтированы для восстановления работоспособных деталей или же полностью переработаны для извлечения ценных материалов.

Люминесцентные лампы, содержащие ртуть, утилизируются согласно законодательству, поскольку они классифицируются как чрезвычайно опасные отходы первого класса. Это предотвращает высокую степень вредного воздействия на окружающую среду, учитывая, что одна лампа может содержать до 5 мг ртути.

## 5.3 Устойчивость в ЧС

Нормы пожарной безопасности в офисах, изложенные в управленческом документе компании, подготовленном специалистом по пожарной безопасности и одобренном руководителем организации, играют ключевую роль в минимизации вероятности возгораний и повышении защиты работников. Эти требования важны для обеспечения безопасности.

Основные нормативные документы, регулирующие пожарную безопасность в офисах, включают Федеральный закон № 123-ФЗ «Технический регламент о требованиях пожарной безопасности» от 22 июля 2008 года и Правила противопожарного режима Российской Федерации, утвержденные Правительством РФ 16 сентября 2020 года   
постановлением № 1479.

Документ, регулирующий организацию офисного пространства и противопожарный режим, должен присутствовать в каждом офисе. В него входят следующие элементы:

* установка табличек с номером телефона пожарной охраны и данными лиц, ответственных за противопожарное состояние помещений, ответственность за размещение которых несет руководитель организации;
* разработка планов эвакуации для каждого этажа офисного здания и создание общего плана, хранящегося у ответственного лица. Для офисов в арендованных помещениях бизнес-центров создаются локальные планы эвакуации;
* размещение в офисе информационных стендов, плакатов и табличек по пожарной безопасности;
* наличие первичных средств пожаротушения, эффективных на начальной стадии возгорания;
* обеспечение свободного доступа к эвакуационным путям и выходам.

Придерживание противопожарных мероприятий в офисе не составляет сложности как для руководства, так и для сотрудников. Внимательное отношение к инструктажам, регулярная проверка наличия и исправности огнетушителей, а также корректно оформленные планы эвакуации в случае пожара способны предотвратить значительные материальные потери и сохранить жизни.

## 5.4 Расчет воздухообмена при общеобменной вентиляции

Имеются исходные данные: температура воздуха в рабочем помещении составляет 23 , температура приточного воздуха вне помещения составляет 21,3 , теплоемкость воздуха составляет 1,2 кДж/ кг⋅К, количество тепла, выделяемого одним человеком при работе, составляет 100 кДж/ч, количество вредных веществ в приточном воздухе составляет 0,03 мг/м3, а в удаляемом воздухе — 5 мг/м3.

Для начала необходимо рассчитать количество тепла, которое будет выделено при работе оборудования по формуле:

, (5.1)

,

где Nуст — установочная мощность оборудования;

K1 — коэффициент использования установочной мощности;

K2 — коэффициент одновременности работы оборудования.

Затем рассчитаем тепло, которое выделяется работниками, по формуле:

, (5.2)

,

где n — число работников;

Kр — тепло, выделяющееся одним человеком, кДж/ч (принимается равной при легкой работе 100 кДж/ч).

, (5.3)

.

Затем рассчитаем потребный воздухообмен для отвода излишнего тепла по формуле:

, (5.4)

,

где Qизб — избыточное количество теплоты, кДж⋅ч;

cp — теплоемкость воздуха, c = 1,2 кДж/ кг⋅К;

c — плотность воздуха, кг/м3;

tпр — температура приточного воздуха, (принятое значение 21,3 );

tуд — температура воздуха, удаляемого из помещения, принимается равной температуре воздуха в рабочей зоне.

Расход приточного воздуха для восполнения концентрации вредных веществ можно найти по формуле:

, (5.5)

,

где G — количество выделяемых вредных веществ, мг/ч;

Qуд — концентрация вредных веществ в удаляемом воздухе, которая не должна превышать предельно допустимую, мг/м3;

Qпр — концентрация вредных веществ в приточном воздухе, мг/м3.

Далее необходимо найти кратность воздуха:

, (5.6)

.

Кратность воздухообмена в помещениях, как правило, составляет   
от 1 до 10.

## 5.5 Анализ и расчет искусственного освещения

Ширина и длина помещения составляет соответственно 5 м и 10 м, площадь составляет 50 м2, а объем 150 м3. В помещении установлены люминесцентные лампы со световым потоком 2800 Лм, а высота их   
подвеса — 2 м.

Для определения количества световых приборов, определим величину светового потока, падающего на поверхность по формуле:

, (5.7)

,

где E — минимальное нормируемое значение освещенности;

S — площадь помещения;

K — коэффициент запаса. Этот показатель учитывает, что в процессе эксплуатации, осветительный прибор будет подвергаться загрязнению и его световой поток будет снижаться;

z — отношение средней освещенности к минимальной;

n — коэффициент использования, выбирается по таблице, исходя из индекса, который рассчитывается по формуле:

, (5.8)

,

где S — площадь помещения;

h — расчетная высота подвеса, h = 2 м;

A — ширина помещения;

B — длина помещения.

Теперь, зная индекс помещения I, найдем коэффициент по таблице 5,2   
n = 0,79.

Таблица 5.2 — Индекс помещения

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Индекс помещения | потолок | 0,8 | 0,7 | 0,7 | 0,5 | 0,5 | 0,5 |
| стены | 0,5 | 0,5 | 0,3 | 0,5 | 0,3 | 0,3 |
| пол | 0,3 | 0,3 | 0,3 | 0,3 | 0,3 | 0,1 |
| 0,60  0,80  1,00  1,25  1,50  1,75  2,00  2,50  3,00  4,00  5,00 | | 0,33 | 0,32 | 0,25 | 0,3 | 0,24 | 0,24 |
| 0,41 | 0,39 | 0,32 | 0,36 | 0,3 | 0,29 |
| 0,47 | 0,45 | 0,38 | 0,42 | 0,35 | 0,34 |
| 0,53 | 0,51 | 0,44 | 0,47 | 0,41 | 0,39 |
| 0,58 | 0,55 | 0,48 | 0,51 | 0,45 | 0,43 |
| 0,79 | 0,72 | 0,71 | 0,66 | 0,70 | 0,65 |
| 0,65 | 0,62 | 0,56 | 0,57 | 0,52 | 0,49 |
| 0,7 | 0,67 | 0,61 | 0,61 | 0,56 | 0,53 |
| 0,64 | 0,71 | 0,65 | 0,64 | 0,6 | 0,56 |
| 0,79 | 0,75 | 0,7 | 0,68 | 0,64 | 0,6 |
| 0,83 | 0,78 | 0,74 | 0,71 | 0,68 | 0,62 |

Для освещения используются люминесцентные лампы, световой поток которых равен F = 2800 Лм.

Рассчитаем необходимое количество лам по формуле:

, (5.9)

,

где N — определяемое число ламп;

F — световой поток;

Fл — световой поток лампы.

Теперь определим минимальное расстояние между рядами светильников и от стен до ближайшего ряда. Расстояние L между рядами светильников находится по формуле:

, (5.10)

,

,

где — коэффициент оптимального расстояния между центрами светильников;

Hn — высота подвеса осветительных приборов.

Фактическое количество ламп соответствует минимально допустимому количеству ламп для освещения помещения.

## 5.6 Выводы по главе

В ходе анализа помещения, где будет использоваться разработанное программное обеспечение, были проведены расчет, подтверждающие соответствие общеобменной вентиляции и искусственного освещения комфортным условиям труда.

Также в данной главе были описаны возможные противопожарные мероприятия и предписания для предотвращения пожара.

Проанализировано влияние на окружающую среду и указания, в соответствии с которыми уменьшается пагубное влияние на окружающую среду.

Заключение

В ходе выполнения данной работы были разработаны серверная и клиентская части бота, которые обеспечивают пользователей инструментом для автоматического отклика. Приложение отвечает всем установленным требованиям.

Данное программное средство полностью реализует задачи, поставленные в пункте 1.3, а именно:

* был реализован понятный и интуитивно простой интерфейс, который позволит не перегружать интерфейс информацией, и предоставит пользователю хороший пользовательский опыт;
* был выбран язык программирования;
* были разработаны необходимые алгоритмы;
* была сформирована структура базы данных;
* была протестирована клиентская часть.

Получены и закреплены навыки разработки и создания программного средства, связанные с полновесной разработкой telegram-бота.

Перечень использованных информационных ресурсов

1. Статья о HMAC на сайте MDN Web Docs [Электронный ресурс], URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/HMAC (дата обращения: 21.06.2024).
2. Статья о SHA-256 на сайте Cryptography Stack Exchange [Электронный ресурс],

URL: https://crypto.stackexchange.com/questions/2113/what-is-sha-256 (дата обращения: 21.06.2024).

1. Статья о библиотеке Aiogram Блог Хабр [Электронный ресурс],

URL: https://habr.com/ru/companies/microsoft/articles/262523/ (дата обращения: 21.05.2024).

1. Статья о языке программирования Python Блог Хабр [Электронный ресурс],

URL: https://habr.com/ru/companies/kryptonite/articles/807345/ (дата обращения: 21.05.2024).

1. Статья о библиотеке Aiogram Сервис для совместной разработки и хостинга проектов GitHub [Электронный ресурс],

URL: https://mastergroosha.github.io/aiogram-3-guide/ (дата обращения: 21.05.2024).

1. Официальная документация Playwright [Электронный ресурс],

URL: https://playwright.dev/docs/intro (дата обращения: 21.05.2024).

1. Статья о библиотеке SQLAlchemy [Электронный ресурс],

URL: https://habr.com/ru/articles/470285/ (дата обращения: 21.05.2024).

1. Статья о СУБД SQLite [Электронный ресурс],

URL: https://www.sqlite.org/ (дата обращения: 21.05.2024).

1. Официальная документация Docker Documentation [Электронный ресурс],

URL: https://docs.docker.com/ (дата обращения: 21.05.2024).

1. Статья о Dolphin на официальном сайте Dolphin Browser [Электронный ресурс],

URL: https://dolphin.com/ (дата обращения: 21.05.2024).

1. Статья об OpenAI ChatGPT на официальном сайте OpenAI [Электронный ресурс],

URL: https://www.openai.com/chatgpt (дата обращения: 21.05.2024).

Приложение А Техническое задание

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент кафедры «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_ С.А. Швидченко  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. | УТВЕРЖДЕНО  Зав. кафедрой «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_ В.В. Долгов  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

#### А.1 Общие сведения

#### А.1.1 Наименование программного средства

Программная реализация сервиса для автоматических откликов фриланс-заказов.

#### А.1.2 Область применения

Программа «FreelanceBot», представленная в данном проекте, представляет собой чат–бота для мессенджера Telegram, который предназначен для автоматизации процессов отклика на вакансии и заказы на фриланс-платформах. Основные области применения:

* фриланс и удаленная работа — программа позволяет фрилансерам автоматически откликаться на новые вакансии и заказы, соответствующие их навыкам и требованиям. Это особенно полезно для фрилансеров, работающих на нескольких платформах и стремящихся максимально увеличить количество заказов;
* рекрутинговые агентства — «FreelanceBot» может использоваться рекрутинговыми агентствами для автоматизации поиска подходящих кандидатов на открытые позиции. Бот может автоматически откликаться на резюме кандидатов, отправлять приглашения на интервью и уведомлять о новых подходящих кандидатах.

Эта программа находит применение в различных сферах, связанных с автоматизацией откликов на вакансии и заказы, управлением пользовательской информацией и настройками, а также взаимодействием с фриланс-платформами. Она может быть полезна как для индивидуальных фрилансеров и самозанятых, так и для рекрутинговых компаний, позволяя им эффективно управлять своими заявками и получать новые заказы.

#### А.2 Основание для разработки

Разработка ведется на основании документа «Учебный план для студентов ВУЗа» направление 02.03.03 «Математическое обеспечение и администрирование информационных систем» кафедры «Программное обеспечение вычислительной техники и автоматизированных систем» факультета «Информатика и вычислительная техника» Донского государственного технического университета.

#### А.3 Назначение разработки

#### А.3.1 Функциональное назначение

Основные функции программы включают:

1. Регистрация и аутентификация пользователей:

* пользователи могут легко зарегистрироваться в системе, вводя свои данные через интерактивный интерфейс бота;
* после регистрации пользователи могут проходить аутентификацию для доступа к функциям бота.

1. Управление профилем пользователя:

* пользователи могут создавать и обновлять свои профили, включая информацию о себе, описание навыков, шаблоны откликов и пожелания к заказам;
* бот предоставляет интерфейс для редактирования профиля и просмотра текущих данных.

1. Автоматические отклики на вакансии и заказы:

* бот автоматически отслеживает новые вакансии и заказы на фриланс-платформах, соответствующие параметрам, заданным пользователем;
* при появлении новых подходящих предложений бот автоматически отправляет отклик от имени пользователя.

1. Уведомления о новых заказах:

* пользователи получают мгновенные уведомления о новых вакансиях и заказах, соответствующих их профилю и требованиям;
* бот отправляет уведомления через Telegram, обеспечивая оперативное информирование пользователя.

1. Управление заказами и откликами:

* пользователи могут просматривать список своих откликов и управлять ими, получать обновления о статусе заказов и вакансий;
* бот предоставляет информацию о текущих заказах и возможностях для откликов.

1. Интеграция с фриланс-платформами:

* бот интегрируется с различными фриланс-платформами (например, Profi.ru, HH.ru), обеспечивая централизованный доступ к вакансиям и заказам;
* использование антидетект браузера (например, Dolphin Anty) для безопасного и анонимного взаимодействия с фриланс-платформами.

1. Безопасность и конфиденциальность:

* бот обеспечивает высокий уровень безопасности данных пользователей, используя шифрование и безопасные методы аутентификации;
* поддержка конфиденциальности информации, передаваемой между пользователем и ботом, а также между ботом и фриланс-платформами.

Программа предназначена для фрилансеров, рекрутеров и самозанятых, обеспечивая автоматизацию процессов, связанных с поиском и откликом на вакансии и заказы, управление профилем и заказами, а также получение актуальной информации и уведомлений.

#### А.3.2 Эксплуатационное назначение

Эксплуатационное назначение состоит в облегчении фрилансерам и самозанятым управление своими откликами и заказами.

#### A.4 Требования к программе

#### А.4.1 Требования к функциональным характеристикам

Программное средство должно осуществлять следующий функционал:

* установление и поддержание соединения с другими участниками сети с помощью взаимодействия с сервером;
* регистрация новых пользователей в системе;
* возможность создания и отправки текстовых сообщений другим пользователям сети;
* просмотр истории сообщений с контактами, включая отображение времени отправки и получения сообщений.

#### А.4.2 Требования к надежности

Надежное функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно–технических мероприятий, перечень которых приведен ниже:

* организация бесперебойного питания технических средств;
* организация правильного использования модулей;
* защиты программного средства от несанкционированного проникновения;
* копирование (архивирование) данных, необходимых для восстановления работы программного средства;
* регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие вирусов.

#### А.4.2.1 Входные данные

Входными данными являются ответы пользователя, получаемые за счет взаимодействия с графическим интерфейсом.

#### А.4.2.2 Выходные данные

Выходные данные представляются в виде информации, которую получает пользователь, в виде заказов или вакансий, собранных с сайта.

#### А.4.3 Условия эксплуатации

Для функционирования программной системы необходимо соблюдение всех требований и правил эксплуатации компьютерной техники.

Для стабильного функционирования и оптимальной работы программного продукта необходимо соблюдение всех требований и правил эксплуатации вычислительной техники.

Высокая квалификация пользователя программного средства не требуется. Дополнительных требований и ограничений не вводится.

#### А.4.4 Требование к составу и параметрам технических средств

В состав технических средств должно входить любое устройство с доступом к сети интернет и наличие на нем программы Telegram.

Дополнительные требования и ограничения к составу и параметрам технических средств не вводится.

#### А.4.5 Требования к исходным кодам и языкам программирования

Программное средство должно быть реализовано на языке программирования Python, а также языке разметки документа xml.

#### А.4.6 Требования к упаковке и маркировке

Требования к упаковке и маркировке программного средства не предъявляется.

#### А.4.7 Требования к транспортировке и хранению

Условия транспортирования, места хранения, условия складирования и сроки хранения в различных условиях должны соответствовать требованиям, предъявляемым к носителям информации, на которых будет содержаться данное программное изделие.

Допустимы все способы транспортирования и хранения, не нарушающие целостность используемого носителя данных. Программное средство может храниться на любом носителе информации, имеющее возможность подключения к персональному компьютеру.

#### А.5 Требования к программной документации

Программная документация должна состоять из следующих листов:

* титульный лист;
* пояснительная записка к дипломной работе;
* техническое задание по ГОСТ 19.201–78 ЕСПД;
* исходный код программного средства по ГОСТ 19.401–79 ЕСПД.

#### А.6 Стадии и этапы разработки

* постановка задачи (с 17.04.23 по 22.04.23);
* изучение предметной области (с 23.04.23 по 26.04.23);
* разработка алгоритмов решения задачи (с 28.04.23 по 02.05.23);
* разработка программы (с 04.05.23 по 09.05.23);
* тестирование программы (с 10.05.23 по 13.05.23).

#### А.7 Порядок контроля и приемки

Порядок и контроль приемки определяются заведующим кафедрой «ПОВТиАС» и основаны на демонстрации знаний технологии и умении создавать программные средства для различных предметных областей.

Главным требованием к приемке является наличие правильно работающего программного средства с тестовым примером и отчета, представленного в печатном виде.

|  |  |
| --- | --- |
| Разработчик технического задания:  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. | /Костюченко Артём Иванович /  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Приложение Б Листинг программы

Листинг Б.1 — Модуль main.py

from settings import \*

from bot.handlers import \*

from modules.database import \*

from modules import \*

from models import \*

async def run(database):

bot = Bot(token = config['Telegram']['api'])

bot.db = database

bot.users\_load = database.query(User).all()

bot.users\_id\_load = list(map(lambda x: x.id, bot.users\_load))

print(bot.users\_id\_load)

scheduler = AsyncIOScheduler(timezone='Europe/Moscow')

scheduler.add\_job(schedule\_task.database\_update, trigger = 'interval', seconds = 60,

kwargs={"bot" : bot})# обновление изменений в таблице каждую минуту

scheduler.start()

dp = Dispatcher()

dp.include\_routers(

commands.router,

queries.router,

messages.router,

registration.router,

profi\_usage.router,

profile.router

)

await bot.delete\_webhook(drop\_pending\_updates=True)

await dp.start\_polling(bot)

if \_\_name\_\_ == "\_\_main\_\_":

db = database\_init()

asyncio.run(run(db))

Листинг Б.2 — Модуль commands.py

from settings import \*

from bot.keyboards.menu import \*

router = Router()

@router.message(StateFilter(None), F.text, Command("start"))

async def start\_msg(message: types.Message):

await message.answer(text='Menu', reply\_markup=main\_menu(message.chat.id, message.bot))

@router.message(StateFilter(None), F.text, Command("test"))

async def start\_msg(message: types.Message):

data = {

"order\_id": "66998481",

"url": "/backoffice/n.php?o=66998481&analytics\_data=%7B%22source%22%3A%22new\_board%22%2C%22board\_order\_index%22%3A1%2C%22board\_item\_index%22%3A1%2C%22board\_search\_used%22%3Afalse%2C%22board\_search\_query%22%3A%22%22%2C%22caseId%22%3Anull%2C%22score%22%3A89.29302%7D",

"title": "Машинное обучение",

"name": "Cемен\n15 мая в 07:20",

"text": "18 лет · Для себя · Нужна помощь с домашней работой, накопились долги. Первый семестр сам тянул, второй по времени не успеваю.1 курс, с++",

"cost": "500–1000 ₽",

"solve\_time": "Дистанционно · Саратов\n +1 час\nСр 16:00-21:00, пт 16:00-21:00, сб 09:00-21:00, вс 07:00-20:00, вт 09:00-10:00, вт 16:00-21:00",

"order\_time": "15 мая в 07:20",

"concurents": "Неизвестно"

}

with suppress(TelegramBadRequest):

await message.answer(text=f'{data["title"]}\n\n{data["text"]}\n\n{data["solve\_time"]}\nЦена: {data["cost"]}',

reply\_markup=order\_menu(data['url'])

)

Листинг Б.3 — Модуль messages.py

from settings import \*

from bot.states.based import \*

router = Router()

# @router.message(F.text)

# async def all\_msg(message: types.Message):

# await message.reply('Test text')

Листинг Б.4 — Модуль profile.py

from bot.handlers.commands import start\_msg

from bot.keyboards.menu import \*

from bot.states.based import \*

from models.user import \*

from settings import \*

router = Router()

###

### Queries

###

@router.callback\_query(F.data == 'profile')

async def show\_profile(callback: types.CallbackQuery, state: FSMContext):

ind = callback.bot.users\_id\_load.index(callback.message.chat.id)

data: User = callback.bot.users\_load[ind]

await state.update\_data({callback.message.chat.id: data})

await callback.message.edit\_text(text = f'{data}',

reply\_markup=profile\_menu(),

parse\_mode='HTML')

@router.callback\_query(F.data == 'profile\_name')

async def change\_name(callback: types.CallbackQuery, state: FSMContext):

ind = callback.bot.users\_id\_load.index(callback.message.chat.id)

callback.bot.users\_load[ind]

await state.set\_state(Profile.NameInput)

data = await state.get\_data()

await callback.message.edit\_text(text=f'Ваше текущее имя: <blockquote>{data[callback.message.chat.id].name}</blockquote>Введите новое имя:',

parse\_mode='HTML')

@router.callback\_query(F.data == 'profile\_about\_me')

async def change\_about(callback: types.CallbackQuery, state: FSMContext):

await state.set\_state(Profile.AboutMeInput)

data = await state.get\_data()

await callback.message.edit\_text(text=f'Что вы написали о себе: <blockquote>{data[callback.message.chat.id].about\_me}</blockquote>',

parse\_mode='HTML')

@router.callback\_query(F.data == 'profile\_template')

async def change\_temp(callback: types.CallbackQuery, state: FSMContext):

await state.set\_state(Profile.TemplateInput)

data = await state.get\_data()

await callback.message.edit\_text(text=f'Ваш шаблон: <blockquote>{data[callback.message.chat.id].template}</blockquote>',

parse\_mode='HTML')

@router.callback\_query(F.data == 'profile\_description')

async def change\_desc(callback: types.CallbackQuery, state: FSMContext):

await state.set\_state(Profile.DescriptionInput)

data = await state.get\_data()

await callback.message.edit\_text(text=f'Ваше описание: <blockquote>{data[callback.message.chat.id].description}</blockquote>',

parse\_mode='HTML')

@router.callback\_query(F.data == 'profile\_wishes')

async def change\_wishes(callback: types.CallbackQuery, state: FSMContext):

await state.set\_state(Profile.WishesInput)

data = await state.get\_data()

await callback.message.edit\_text(text=f'Ваше пожелания: <blockquote>{data[callback.message.chat.id].wishes}</blockquote>',

parse\_mode='HTML')

### Messages

@router.message(Profile.NameInput, F.text)

async def change\_name\_inp(message: types.Message, state: FSMContext):

await state.set\_state()

await message.reply(text = f'Имя будет измененo на: "{message.text}" в течении минтуты')

ind = message.bot.users\_id\_load.index(message.chat.id)

message.bot.users\_load[ind].name = message.text

await start\_msg(message)

@router.message(Profile.DescriptionInput, F.text)

async def change\_name\_inp(message: types.Message, state: FSMContext):

await state.set\_state()

await message.reply(text = f'Описание будет изменено на: "{message.text}" в течении минтуты')

ind = message.bot.users\_id\_load.index(message.chat.id)

message.bot.users\_load[ind].description = message.text

await start\_msg(message)

@router.message(Profile.WishesInput, F.text)

async def change\_name\_inp(message: types.Message, state: FSMContext):

await state.set\_state()

await message.reply(text = f'Пожелание будет изменено на: "{message.text}" в течении минтуты')

ind = message.bot.users\_id\_load.index(message.chat.id)

message.bot.users\_load[ind].wishes = message.text

await start\_msg(message)

@router.message(Profile.AboutMeInput, F.text)

async def change\_name\_inp(message: types.Message, state: FSMContext):

await state.set\_state()

await message.reply(text = f'Информанция о себе будет изменена на: "{message.text}" в течении минтуты')

ind = message.bot.users\_id\_load.index(message.chat.id)

message.bot.users\_load[ind].about\_me = message.text

await start\_msg(message)

@router.message(Profile.TemplateInput, F.text)

async def change\_name\_inp(message: types.Message, state: FSMContext):

await state.set\_state()

await message.reply(text = f'Шаблон будет изменен на: "{message.text}" в течении минтуты')

ind = message.bot.users\_id\_load.index(message.chat.id)

message.bot.users\_load[ind].template = message.text

await start\_msg(message)

Листинг Б.5 — Модуль profi\_usage.py

from settings import \*

from bot.keyboards.menu import \*

from bot.states.based import \*

from models.user import \*

router = Router()

router

Листинг Б.6 — Модуль queries.py

from settings import \*

from bot.keyboards.menu import \*

from bot.states import \*

router = Router()

@router.callback\_query(F.data == 'profi\_menu')

async def profi\_msg(callback: types.CallbackQuery):

await callback.message.edit\_text(text='Profi MENU', reply\_markup=profi\_menu())

@router.callback\_query(F.data == 'hh\_menu')

async def hh\_msg(callback: types.CallbackQuery):

await callback.message.edit\_text(text='HH MENU', reply\_markup=profi\_menu())

Листинг Б.7 — Модуль registration.py

from settings import \*

from bot.keyboards.menu import \*

from bot.states.based import \*

from models.user import \*

router = Router()

###

### Queries

###

@router.callback\_query(F.data == 'register')

async def start\_register(callback: types.CallbackQuery, state: FSMContext):

await state.set\_state(Register.NameInput)

await callback.message.reply(text = 'Введите ваше имя', reply\_markup=pass\_())

### Messages

@router.message(Register.NameInput, F.text)

async def name\_input(message: types.Message, state: FSMContext):

await state.update\_data({'name': message.text})

await state.set\_state(Register.DescriptionInput)

await message.reply(text='Введите описание того чем занимаетесь, как долго и т.п.', reply\_markup=pass\_())

@router.message(Register.DescriptionInput, F.text)

async def desc\_input(message: types.Message, state: FSMContext):

await state.update\_data({'description': message.text})

await state.set\_state(Register.AboutMeInput)

await message.reply(text='Введите информацию о себе', reply\_markup=pass\_())

@router.message(Register.AboutMeInput, F.text)

async def temp\_input(message: types.Message, state: FSMContext):

await state.update\_data({'about\_me': message.text})

await state.set\_state(Register.TemplateInput)

await message.reply(text='Введите шаблон(ы) ', reply\_markup=pass\_())

@router.message(Register.TemplateInput, F.text)

async def temp\_input(message: types.Message, state: FSMContext):

await state.update\_data({'template': message.text})

await state.set\_state(Register.WishesInput)

await message.reply(text='Введите пожелания к желаемым откликам\вакансии ', reply\_markup=pass\_())

@router.message(Register.WishesInput, F.text)

async def wishes\_input(message:types.Message, state: FSMContext):

await state.update\_data({'wishes': message.text})

await state.set\_state()

await message.reply(text='Данные заполнены успешно')

data = await state.get\_data()

message.bot.db.add(User(message.chat.id, data))

message.bot.db.commit()

message.bot.users\_load = message.bot.db.query(User).all()

message.bot.users\_id\_load = list(map(lambda x: x.id, message.bot.users\_load))

Листинг Б.8 — Модуль fsm.py

from settings import \*

class Platform()

Листинг Б.9 — Модуль menu.py

from settings import \*

def main\_menu(id, bot: Bot):

buttons = [

[InlineKeyboardButton(text="Зарегистрироваться", callback\_data='register')] if id not in bot.users\_id\_load else [InlineKeyboardButton(text="Пофиль", callback\_data='profile')],

[InlineKeyboardButton(text="Profi.ru", callback\_data='profi\_menu')] if id in bot.users\_id\_load else [],

[InlineKeyboardButton(text="HH.ru", callback\_data='hh\_menu')] if id in bot.users\_id\_load else [],

]

return InlineKeyboardMarkup(inline\_keyboard=buttons, resize=True)

def profile\_menu():

buttons = [

[InlineKeyboardButton(text="Имя", callback\_data='profile\_name')],

[InlineKeyboardButton(text="Описание", callback\_data='profile\_description')],

[InlineKeyboardButton(text="О себе", callback\_data='profile\_about\_me')],

[InlineKeyboardButton(text="Шаблон(ы)", callback\_data='profile\_template')],

[InlineKeyboardButton(text="Пожелания", callback\_data='profile\_wishes')],

]

return InlineKeyboardMarkup(inline\_keyboard=buttons, resize=True)

def profi\_menu():

buttons = [

[InlineKeyboardButton(text="Авторизирваться в кабинете", callback\_data='s1')],

[InlineKeyboardButton(text="Включить автоотклик", callback\_data='s2')],

[InlineKeyboardButton(text="История откликов", callback\_data='s3')],

[InlineKeyboardButton(text="Назад", callback\_data='s4')],

]

return InlineKeyboardMarkup(inline\_keyboard=buttons, resize=True)

def hh\_menu():

buttons = [

[InlineKeyboardButton(text="Авторизирваться в кабинете", callback\_data='s1')],

[InlineKeyboardButton(text="Включить автоотклик", callback\_data='s2')],

[InlineKeyboardButton(text="История откликов", callback\_data='s3')],

[InlineKeyboardButton(text="Назад", callback\_data='s4')],

]

return InlineKeyboardMarkup(inline\_keyboard=buttons, resize=True)

def order\_menu(url: str):

buttons = [

[

InlineKeyboardButton(text="Откликнуться", callback\_data='s1'),

InlineKeyboardButton(text="Открыть отклик",callback\_data='s3'),

InlineKeyboardButton(text="Игнорировать", callback\_data='s2')],

]

return InlineKeyboardMarkup(inline\_keyboard=buttons, resize=True)

def pass\_():

buttons = [

[InlineKeyboardButton(text="Пропустить", callback\_data='pass')],

]

return InlineKeyboardMarkup(inline\_keyboard=buttons, resize=True)

Листинг Б.10 — Модуль based.py

from settings import \*

class Register(StatesGroup):

NameInput = State()

DescriptionInput = State()

AboutMeInput = State()

TemplateInput = State()

WishesInput = State()

class Profile(StatesGroup):

NameInput = State()

DescriptionInput = State()

AboutMeInput = State()

TemplateInput = State()

WishesInput = State()

Листинг Б.11 — Модуль hh.py

from settings import \*

class HHAuth(StatesGroup):

login = State()

code = State()

class HHmenu(StatesGroup):

s1 = State()

s2 = State()

s3 = State()

s4 = State()

Листинг Б.12 — Модуль profi.py

from settings import \*

class ProfiAuth(StatesGroup):

login = State()

password = State()

class ProfiMenu(StatesGroup):

s1 = State()

s2 = State()

s3 = State()

s4 = State()

# class ProfiAnkete(StatesGroup):

Листинг Б.13 — Модуль browser.py

from playwright.async\_api import async\_playwright, Page, Browser

from typing import Optional

import asyncio

class Browser():

"""Standart Browser Model for user action

"""

def \_\_init\_\_(self) -> None:

self.playwright = None

self.browser: Optional[Browser] = None

self.page: Optional[Page] = None

async def init\_browser(self) -> None:

self.playwright = await async\_playwright().start()

self.browser = await self.playwright.chromium.launch()

self.page = await self.browser.new\_page()

async def close\_browser(self) -> None:

if self.browser:

await self.browser.close()

Листинг Б.14 — Модуль orders.py

from modules.database import \*

class Order(DB):

"""

class for Freelance Orders from different sites

"""

\_\_tablename\_\_ = 'order'

# Определение столбцов таблицы

id = Column(Integer, primary\_key=True, autoincrement=True)

user\_id = Column(Integer)

order\_id = Column(String)

url = Column(String, nullable=True)

titile = Column(String, nullable=True)

cost = Column(String, nullable=True)

text = Column(String, nullable=True)

solve\_time = Column(DateTime, nullable=True)

order\_time = Column(DateTime, nullable=True)

name = Column(String, nullable=True)

concurents = Column(String, nullable=True)

date\_added = Column(DateTime, nullable=True)

def \_\_init\_\_(self, user\_id, order\_id, data) -> None:

self.user\_id = user\_id

self.order\_id = order\_id

self.url = data.get('url', '')

self.name = data.get('name', '')

self.title = data.get('title', '')

self.text = data.get('text', '')

self.cost = data.get('cost', '')

self.solve\_time = data.get('solve\_time', '')

self.order\_time = data.get('order\_time', '')

self.concurents = data.get('concurents', '')

def to\_dict(self):

return {

'order\_id': self.order\_id,

'url': self.url,

'title': self.title,

'name': self.name,

'text': self.text,

'cost': self.cost,

'solve\_time': self.solve\_time,

'order\_time': self.order\_time,

'concurents': self.concurents,

}

Листинг Б.15 — Модуль user.py

from typing import Self, Optional, List, Dict

from modules.database import \*

class User(DB):

\_\_tablename\_\_ = 'users'

id = Column(Integer, primary\_key=True)

name = Column(String, nullable=True)

description = Column(String, nullable=True)

about\_me = Column(String, nullable=True)

template = Column(String, nullable=True)

wishes = Column(String, nullable=True)

"""Telegram User model for uploading his data"""

def \_\_init\_\_(self, id: Optional[int], kwargs) -> None:

self.id = id

self.login = kwargs.get('login', None)

self.password = kwargs.get('password', None)

self.name = kwargs.get('name', None)

self.description = kwargs.get('description', None)

self.about\_me = kwargs.get('about\_me', None)

self.template = kwargs.get('template', None)

self.wishes = kwargs.get('wishes', None)

def to\_dict(self):

return {

'id': self.id,

'name': self.name,

'login': self.login,

'password': self.password,

'description': self.description,

'about\_me': self.about\_me,

'template': self.template,

'wishes': self.wishes

}

@staticmethod

def upload\_from\_data(data: Optional[Dict])-> List['User']:

users = {}

for id in data:

users[id] = User(id, data[id])

return users

def \_\_str\_\_(self):

return f"ID: <blockquote>{self.id}</blockquote>\nИмя: <blockquote>{self.name}</blockquote>\nОписание: <blockquote>{self.description}</blockquote>\nОбо мне: <blockquote>{self.about\_me}</blockquote>\nШаблон: <blockquote>{self.template}</blockquote>"[:4096]

Листинг Б.16 — Модуль DolphinController.py

import psutil

import json

import random

import requests

from settings import \*

platforms\_array = ["macos", "windows", "linux"]

vendors\_array = ["Google Inc. (Intel)", "Google Inc. (NVIDIA)", "Google Inc. (AMD)", "Google Inc.",

"Google Inc. (Microsoft)", "Google Inc. (Unknown)", "Intel Inc.",

"Google Inc. (NVIDIA Corporation)"]

renderers\_array = {"Google Inc. (Intel)": [

"ANGLE (Intel, Intel(R) HD Graphics Family Direct3D11 vs\_5\_0 ps\_5\_0, D3D11)",

"ANGLE (Intel, Intel(R) UHD Graphics Direct3D11 vs\_5\_0 ps\_5\_0, D3D11)"

],

"Google Inc. (NVIDIA)": [

"ANGLE (NVIDIA, NVIDIA GeForce GTX 1060 6GB Direct3D11 vs\_5\_0 ps\_5\_0, D3D11-30.0.14.7247)",

"ANGLE (NVIDIA, NVIDIA GeForce GTX 1050 Ti Direct3D11 vs\_5\_0 ps\_5\_0, D3D11-27.21.14.6172)"

],

"Google Inc. (AMD)": [

"ANGLE (AMD, AMD Radeon(TM) Vega 10 Graphics Direct3D11 vs\_5\_0 ps\_5\_0, D3D11)",

"ANGLE (AMD, AMD Radeon(TM) Graphics Direct3D11 vs\_5\_0 ps\_5\_0, D3D11-30.0.13044.0)"

],

"Google Inc.": [

"ANGLE (Intel(R) UHD Graphics Direct3D11 vs\_5\_0 ps\_5\_0)",

"ANGLE (Intel(R) UHD Graphics 620 Direct3D11 vs\_5\_0 ps\_5\_0)"

],

"Google Inc. (Microsoft)": [

"ANGLE (Microsoft, Microsoft Basic Render Driver Direct3D11 vs\_5\_0 ps\_5\_0, D3D11-10.0.19041.546)",

"ANGLE (Microsoft, Microsoft Basic Render Driver Direct3D11 vs\_5\_0 ps\_5\_0, D3D11-6.3.9600.16505)"

],

"Google Inc. (Unknown)": [

"ANGLE (Unknown, Qualcomm(R) Adreno(TM) 630 GPU Direct3D11 vs\_5\_0 ps\_5\_0, D3D11)",

"ANGLE (Unknown, Qualcomm(R) Adreno(TM) 630 GPU Direct3D11 vs\_5\_0 ps\_5\_0, D3D11-25.18.10500.0)"

],

"Intel Inc.": [

"Intel Iris OpenGL Engine"

],

"Google Inc. (NVIDIA Corporation)": [

"ANGLE (NVIDIA Corporation, GeForce RTX 3070/PCIe/SSE2, OpenGL 4.5.0 NVIDIA 461.40)"

]}

cpu = [2, 4, 6, 8, 12, 16]

memory = [2, 4, 8]

def authentication(username: str, password: str):

options = {

"url": "https://dolphin-anty-api.com/auth/login",

"data": {

"username": username,

"password": password

}

}

response = requests.post(options["url"], data=options["data"])

logging.info(f"Dolphin connection code: {response.status\_code}")

response\_json = response.json()

if "token" in response\_json:

return response\_json["token"]

else:

raise Exception(

"Authentication error, check your Internet connection and config data")

def get\_new\_user\_agent(platform: str, session: requests.Session):

version\_number = random.randint(101, 107)

options = {

"url": f"https://anty-api.com/fingerprints/useragent?browser\_type=anty&browser\_version=107&" +

f"platform={platform}"

}

response = session.get(options["url"]).json()

# print(response)

if "data" in response:

return response["data"]

else:

raise Exception("Can't get new user agent, something went wrong")

def create\_profile\_and\_send\_id(session: requests.Session):

null = None

# set current parameters for browser profile

current\_vendor = random.choice(vendors\_array)

current\_renderer = random.choice(renderers\_array[current\_vendor])

current\_platform = random.choice(platforms\_array)

current\_user\_agent = get\_new\_user\_agent(

platform=current\_platform, session=session)

# set options for browser profile

options = {

"url": "https://anty-api.com/browser\_profiles",

"data": {

"platform": current\_platform,

"browserType": "anty",

"useragent": {

"mode": "manual",

"value": current\_user\_agent

},

"webrtc": {

"mode": "altered",

"ipAddress": null

},

"canvas": {

"mode": "real"

},

"webgl": {

"mode": "real"

},

"webglInfo": {

"mode": "manual",

"vendor": current\_vendor,

"renderer": current\_renderer

},

"geolocation": {

"mode": "real",

"latitude": null,

"longitude": null

},

"cpu": {

"mode": "manual",

"value": random.choice(cpu)

},

"memory": {

"mode": "manual",

"value": random.choice(memory)

},

"timezone": {

"mode": "auto",

"value": null

},

"locale": {

"mode": "auto",

"value": null

},

"name": "Current Bot",

"tags": [

""

],

"mainWebsite": "",

"notes": {

"content": null,

"color": "blue",

"style": "text",

"icon": null

},

"proxy": null,

"statusId": 0,

"doNotTrack": False

}

}

response = session.post(options["url"], data=json.dumps(options["data"]))

# print(response.json())

profile\_id = get\_last\_browser\_profile(session=session)

session.close()

return profile\_id['id']

def get\_last\_browser\_profile(session: requests.Session):

options = {

"url": "https://anty-api.com/browser\_profiles"

}

response = session.get(options["url"]).json()

if "data" in response:

return response["data"][0]

else:

raise Exception("Can't get list of browser profiles")

def delete\_browser\_profile(session: requests.Session, profile\_id: str):

options = {

'url': "https://anty-api.com/browser\_profiles/"

}

response = session.delete(options["url"] + str(profile\_id))

# print(response.json())

def start\_automation(session, profile\_id: str):

options = {

"url": f"http://localhost:3001/v1.0/browser\_profiles/{profile\_id}/start?automation=1"

}

response = session.get(options['url']).json()

return response

def close\_browsers\_process(port = None):

for process in psutil.process\_iter(['pid', 'name']):

if process.info['name'].startswith('anty'):

connections = process.connections()

process.kill()

Листинг Б.17 — Модуль gpt.py

from settings import \*

import openai

import httpx

from settings import \*

async def answer(text: str, chat\_history: list[str] = []):

system\_prompt = "" # Начальные настройки ответа

proxy = config["Proxy"]

async with openai.AsyncOpenAI(api\_key=config['gpt']['token'], http\_client=httpx.Client(proxy=f"http://{proxy['login']}:{proxy['pass']}@{proxy['ip']}:{proxy['ip']}")) as client:

messages = [

{"role": "system", "content": system\_prompt},

]

messages.extend(list(map(lambda message\_text: {"role": "user", "content": message\_text}, chat\_history)))

messages.append({"role": "user", "content": text})

completion = await client.chat.completions.create(

model="gpt-4o",

messages = messages

)

return completion.choices[0].message

Листинг Б.18 — Модуль profi.py

from settings import \*

from models.browser import \*

from models.orders import Order

import modules.DolphinController as DolphinController

from modules.recapcha\_solver import \*

class ProfiBrowser(Browser):

def \_\_init\_\_(self) -> None:

super().\_\_init\_\_()

async def get\_details\_per\_url(self, url):

page2:Page = await self.browser.new\_page()

await page2.goto('url')

text\_element = await page2.query\_selector('')

text = text\_element.inner\_text()

concurents\_element = await page2.query\_selector('')

concurents = concurents\_element.inner\_text()

return text, concurents

async def auth(self, telegram\_id:int, page:Page):

try:

login = await page.query\_selector('//input[contains(@class,"login-form\_\_input-login")]')

await login.type('+79508569688')

await page.click('//a[@role="button"]')

await page.wait\_for\_selector('//div[@class="sidebar\_\_inner"]', timeout=30000)

print('авторизация пройдена')

# await recap\_solve(page, image\_solve=True)

except:

logging.error(traceback.format\_exc())

finally:

logging.info(f'TID: {telegram\_id} | Authorized')

async def check\_orders(self, telegram\_id:int, page:Page):

await page.goto('https://profi.ru/backoffice/n.php', timeout=120000)

await self.auth(telegram\_id, page)

order\_cards = await page.query\_selector\_all("//div[contains(@class, 'CardContainer')]//a")

orders: list[Order] = []

for card in order\_cards:

url = await card.get\_attribute('href')

order\_id = await card.get\_attribute('data-testid')

order\_id = re.findall(r"\d+", order\_id)[0]

text\_element = await card.query\_selector("//p[contains(@class, 'SnippetBodyStyles\_\_MainInfo')]")

text = await text\_element.inner\_text()

title\_element = await card.query\_selector('//div//h3')

title = await title\_element.inner\_text()

cost\_element = await card.query\_selector("//div/span[contains(@class, 'SubjectAndPriceStyles\_\_PriceLine')]")

cost = await cost\_element.inner\_text()

schedule\_element = await card.query\_selector('//ul[contains(@class, "LocationAndScheduleStyles\_\_Container")]')

schedule = await schedule\_element.inner\_text()

client\_name\_element = await card.query\_selector('//div[contains(@class, "StatusAndClientInfoStyles\_\_Container")]/div/div/div/span[1]')

client\_name = await client\_name\_element.inner\_text()

order\_time\_element = await card.query\_selector('//span[contains(@class, "Date\_\_DateText")]')

order\_time = await order\_time\_element.inner\_text()

data = {

'url': "https://profi.ru/"+url,

'title': title,

'name': client\_name,

'text': text,

'cost': cost,

'solve\_time': schedule,

'order\_time': order\_time,

'concurents': 'Неизвестно',

'time\_added': datetime.datetime.now(),

}

orders.append(Order(telegram\_id, order\_id, data))

return orders

async def prepare\_page(self):

DolphinController.close\_browsers\_process() # Закрытие всех браузеров

# await asyncio.sleep()

await self.init\_browser()

# print(self.playwright)

auth\_token = DolphinController.authentication(config['Dolphin']['login'], config['Dolphin']['pass'])

session = requests.session()

headers = {

"Authorization": f"Bearer {auth\_token}",

"Content-Type": "application/json"

}

session.headers = headers

# current\_profile\_id = DolphinController.create\_profile\_and\_send\_id(session)

automation\_json = DolphinController.start\_automation(session, profile\_id= config['Dolphin']['brow\_id'] )#current\_profile\_id)

print(f"{automation\_json=}")

dolphin\_port = automation\_json["automation"]["port"]

dolphin\_ws\_endpoint = automation\_json["automation"]["wsEndpoint"]

print(f'{dolphin\_port=} | {dolphin\_ws\_endpoint=}')

chromium = self.playwright.chromium

# self.browser = await chromium.connect\_over\_cdp(f"ws://127.0.0.1:{dolphin\_port}:{dolphin\_ws\_endpoint}")

self.browser = await chromium.connect\_over\_cdp(f"http://localhost:{dolphin\_port}")

context = self.browser.contexts[0]

page = await context.new\_page()

return page

async def run(self, id):

page = await self.prepare\_page()

orders = await self.check\_orders(0, page)

data\_in\_dict = {"data": list(map(lambda x: x.to\_dict(),orders))}

with open(f'data/user\_orders\_{id}.json', 'w', encoding='utf-8') as fl:

json.dump(data\_in\_dict, fl, ensure\_ascii=False, indent=4)

await self.browser.close()