**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет *ИСТ* Кафедра *ИВК*

К ЗАЩИТЕ ДОПУСТИТЬ

Зав. кафедрой

/ /

подпись инициалы, фамилия

« » 20 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Тема    *Web-ориентированная система обучения программированию*

Обучающийся / *И.И. Тагашев*           /

подпись инициалы, фамилия

Обозначение работы *ВКР-УлГТУ-09.03.02-19/1093-2023* Группа *ИСТбд-42*

Направление подготовки *09.03.02 «Информационные системы и технологии»*

код, наименование

Руководитель работы / *В.В. Родионов* /

подпись инициалы, фамилия

Консультанты:

Экономический раздел / *М.В. Рыбкина* /

наименование раздела подпись, дата инициалы, фамилия

Ульяновск, 2023

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет *ИСТ*  Кафедра *ИВК*

Направление подготовки *информационные системы и технологии*

УТВЕРЖДАЮ

Зав. кафедрой

/ /

подпись инициалы, фамилия

« » 20 г.

ЗАДАНИЕ

**на выпускную квалификационную работу**

обучающемуся *Тагашеву Ивану Игоревичу*  курса *4* группы *ИСТбд-42*

фамилия, имя, отчество

1. Тема работы *Web-ориентированная система обучения программированию*

утверждена приказом по университету № *40* от «*28*» *февраля* 20 *23* г.

2. Срок сдачи обучающимся законченной работы «*15* » *июня*             20 *23*г.

3. Исходные данные к работе *Создать web-ориентированную систему обучения программиро-ванию, предусмотрев наличие текстового учебного материала для изучения, а также функции ввода и компиляции кратких примеров на языках программирования C#, C++, Python, Pascal ABC. Должна присутствовать возможность администрирования системы.*

4. Содержание пояснительной записки *техническое задание на создание системы; информационное, алгоритмическое, программное обеспечение системы; тестирование системы.*

5. Перечень графического материала (чертежей) *схема алгоритма компиляции программного кода примеров (чертёж формата А1 по ГОСТ 19.701-90), концептуальная схема базы данных (чертёж формата А1 в нотации IDEF1X).*

6. Календарный график работы на весь период (с указанием сроков выполнения и содержания отдельных этапов)

|  |  |  |
| --- | --- | --- |
| **№ этапа** | **Содержание этапа** | **Срок выполнения** |
| *1* | *Реализация предварительной версии системы и её показ на выставке программных продуктов недели студенческой науки* | *14.04.2023* |
| *2* | *Реализация окончательной версии системы* | *31.05.2023* |
| *3* | *Тестирование системы* | *02.06.2023* |
| *4* | *Написание пояснительной записки* | *04.06.2023* |
| *4* | *Создание чертежей* | *06.06.2023* |
| *5* | *Прохождение нормоконтроля* | *08.06.2023* |
| *6* | *Предварительная защита* | *09.06.2023* |
| *7* | *Защита* | *20.06.2023* |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

7. Консультанты

|  |  |  |  |
| --- | --- | --- | --- |
| **Раздел** | **Ф.И.О. консультанта** | **Подпись, дата** | |
| задание выдал | задание принял |
| *Экономический раздел* | *М.В. Рыбкина* |  |  |
|  |  |  |  |

8. Дата выдачи задания «*17*» *марта* 20 *23* г.

Руководитель *доцент каф. ИВК, к.т.н., доцент* / *В.В. Родионов* /

должность, учёная степень, учёное звание подпись инициалы, фамилия

Задание принял к исполнению / *И. И. Тагашев* /

подпись инициалы, фамилия

**АННОТАЦИЯ**

**Выпускная квалификационная работа:** Тагашева Ивана Игоревича по теме «*Web-ориентированная система обучения программированию*».   
Руководитель Родионов Виктор Викторович. Работа защищена на кафедре «Измерительно-вычислительные комплексы» УлГТУ в 2023 году.

**Пояснительная записка**: 103 страницы, 6 разделов, 1 приложение,   
22 рисунка, 45 таблиц, 20 источник.

**Ключевые слова**: обучение программированию, Python, SQLite, интерактивность обучения, Django, информатика, ФГОС.

В данной выпускной квалификационной работе реализована система обучения программированию с возможностью выполнения пользовательского кода напрямую в браузере. Данная система имеет web-интерфейс, реализована средствами языка Python и средой выполнения кода WandboxAPI. Система представлена четыремя основными страницами: главная страница, страница главы, персональная страница пользователя и административная панель для управления системой, страница выполнения задания, а также предоставление отчетов по итогам выполнения работы учеником для преподавателя. Система реализована с использованием актуальных библиотек.

Содержание

[СПИСОК ИСПОЛЬЗОВАННЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ 9](#_Toc137554405)

[ВВЕДЕНИЕ 10](#_Toc137554406)

[1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА СОЗДАНИЕ СИСТЕМЫ 14](#_Toc137554407)

[1.1 Общие сведения 14](#_Toc137554408)

[1.1.1 Полное наименование системы и ее условное обозначение 14](#_Toc137554409)

[1.1.2 Перечень документов, на основании которых создается система 14](#_Toc137554410)

[1.2.2 Плановые сроки начала и окончания работы по созданию системы 14](#_Toc137554411)

[1.2 Назначение и цели создания системы 14](#_Toc137554412)

[1.2.1 Назначение системы 14](#_Toc137554413)

[1.2.2 Цели создания системы 15](#_Toc137554414)

[1.3 Характеристика объекта автоматизации 15](#_Toc137554415)

[1.3.1 Краткие сведения об объекте автоматизации 15](#_Toc137554416)

[1.3.2 Сведения об условиях эксплуатации 15](#_Toc137554417)

[1.4 Требования к системе 16](#_Toc137554418)

[1.4.1 Требования к структуре и функционирования системы 16](#_Toc137554419)

[1.4.2 Требования к численности и квалификации персонала системы 17](#_Toc137554420)

[1.4.3 Показатели назначения 18](#_Toc137554421)

[1.4.4 Требования к надежности системы 18](#_Toc137554422)

[1.4.5 Требования безопасности 18](#_Toc137554423)

[1.4.6 Требования к защите информации от несанкционированного доступа 18](#_Toc137554424)

[1.4.7 Требования к стандартизации и унификации 18](#_Toc137554425)

[1.4.8 Требования к лицензионной и патентной частоте 19](#_Toc137554426)

[1.5 Требования к функциям, выполняемым системой 19](#_Toc137554427)

[1.5.1 Функции авторизации, регистрации и контроля доступа 19](#_Toc137554428)

[1.5.2 Функция просмотра материалов обучения 19](#_Toc137554429)

[1.5.3 Функция просмотра учебного материала из главы 20](#_Toc137554430)

[1.5.4 Функция выполнения компиляции кода 20](#_Toc137554431)

[1.5.5 Функция добавления учебного материала 20](#_Toc137554432)

[1.5.6 Функция редактирования и удаления учебного материала 20](#_Toc137554433)

[1.5.7 Функция отслеживания прогресса пользователем 21](#_Toc137554434)

[1.5.8 Функция парного программирования 21](#_Toc137554435)

[1.5.9 Функция отправки жалобы на задание 21](#_Toc137554436)

[1.6 Требования к видам обеспечения 22](#_Toc137554437)

[1.6.1 Требования к информационному обеспечению 22](#_Toc137554438)

[1.6.2 Требования к алгоритмическому обеспечению 22](#_Toc137554439)

[1.6.3 Требования к математическому обеспечению 22](#_Toc137554440)

[1.6.4 Требования к лингвистическому обеспечению системы 22](#_Toc137554441)

[1.6.5 Требования к метрологическому обеспечению 22](#_Toc137554442)

[1.6.6 Требования к программному обеспечению 22](#_Toc137554443)

[1.6.7 Требования к организационному обеспечению 23](#_Toc137554444)

[1.7 Состав и содержание работ по созданию (развитию) системы 23](#_Toc137554445)

[1.7.1 Перечень стадий и этапов работ по созданию системы 23](#_Toc137554448)

[1.7.2 Перечень документов, предъявляемых по окончании стадий и этапов работ 25](#_Toc137554449)

[1.8 Требования к документированию 25](#_Toc137554450)

[1.8.1 Согласованный перечень подлежащих разработке комплектов и видов документов 25](#_Toc137554451)

[1.9 Источники разработки 26](#_Toc137554452)

[1.9.1 Перечень документов информационных материалов 26](#_Toc137554453)

[1.10 Анализ аналогичных разработок 26](#_Toc137554454)

[2 ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ 30](#_Toc137554455)

[2.1 Выбор средств управления данными 30](#_Toc137554456)

[2.2 Проектирование базы данных 31](#_Toc137554457)

[2.2.1 Концептуальная схема базы данных 31](#_Toc137554458)

[2.2.2 Внутренняя схема базы данных 35](#_Toc137554459)

[2.3 Организация сбора, передачи, обработки и выдачи информации 39](#_Toc137554460)

[3 АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ 40](#_Toc137554461)

[3.1 Алгоритм регистрации нового пользователя 40](#_Toc137554462)

[3.2 Алгоритм редактирования учебного материала 40](#_Toc137554463)

[3.3 Алгоритм компиляции программного кода примеров 42](#_Toc137554464)

[4 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ 44](#_Toc137554465)

[4.1 Структура программного обеспечения и функции его компонентов 44](#_Toc137554466)

[4.2 Выбор компонентов программного обеспечения 44](#_Toc137554467)

[4.2.1 Операционная система 44](#_Toc137554468)

[4.2.2 Инструментальное средство разработки и язык программирования 44](#_Toc137554469)

[4.2.3 Вспомогательное программное обеспечение 46](#_Toc137554470)

[4.3 Разработка прикладного программного обеспечения 47](#_Toc137554471)

[4.3.1 Структура прикладного программного обеспечения 47](#_Toc137554472)

[4.3.2 Реализация представлений в веб-приложении 48](#_Toc137554473)

[4.3.3 Реализация моделей, используемых в веб-приложении 48](#_Toc137554474)

[4.3.4 Реализация основных модулей веб-приложения 54](#_Toc137554475)

[4.4 Особенности реализации, эксплуатации и сопровождения системы 57](#_Toc137554476)

[4.5 Руководство пользователя 58](#_Toc137554477)

[4.5.1 Требования к условиям эксплуатации 58](#_Toc137554478)

[4.5.2 Инсталляция и настройка 58](#_Toc137554479)

[4.5.3 Порядок и особенности работы 59](#_Toc137554480)

[4.5.4 Исключительные ситуации и их обработка 65](#_Toc137554481)

[5 ТЕСТИРОВАНИЕ СИСТЕМЫ 66](#_Toc137554482)

[5.1 Условия и порядок тестирования 66](#_Toc137554483)

[5.2 Исходные данные для контрольных примеров 66](#_Toc137554484)

[5.2.1 Страница авторизации 66](#_Toc137554485)

[5.2.2 Страница главы 66](#_Toc137554486)

[5.2.3 Страница изменения, удаления, просмотра заданий 67](#_Toc137554487)

[5.2.4 Страница выполнения задания 67](#_Toc137554488)

[5.2.5 Страница с отчетами об отправках 67](#_Toc137554489)

[5.2.6 Страница обработки жалобы 67](#_Toc137554490)

[5.3 Результаты тестирования 68](#_Toc137554491)

[6 ЭКОНОМИЧЕСКИЙ РАЗДЕЛ 69](#_Toc137554492)

[6.1 Оценка совокупной стоимости владения 69](#_Toc137554493)

[6.2 Анализ качественных и количественных факторов воздействия проекта на бизнес-архитектуру организации 73](#_Toc137554494)

[ЗАКЛЮЧЕНИЕ 75](#_Toc137554495)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 76](#_Toc137554496)

[ПРИЛОЖЕНИЕ А 78](#_Toc137554497)

# СПИСОК ИСПОЛЬЗОВАННЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

**БД** (база данных) – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе.

**ГОСТ** (государственный стандарт) – это государственный стандарт, который включает в себя требования государства к качеству продукции, его геометрические размеры, отклонения от эталона и т. д.

**РД** (руководящий документ) – нормативный документ, содержащий правила, общие принципы, характеристики объектов нормирования, касающиеся определенных видов деятельности или их результатов, доступный широкому кругу потребителей (пользователей) и утвержденный в установленном в отрасли порядке.

**СПО** (свободное програмное обеспечение) – программное обеспечение, пользователи которого имеют права на его неограниченную установку, запуск, свободное использование, изучение, распространение и изменение, а также распространение копий и результатов изменения.

**ВВЕДЕНИЕ**

Современная система образования направлена на получение образования и интеллектуального развития человека, владеющего целостной картиной мира и понимающего взаимосвязи, события и процессы, составляющие эту картину. Человек, заканчивающий школу, должен уметь максимально быстро взаимодействовать с внешней средой, приспособиться и эффективно функционировать, то есть быть компетентным в самых важных сферах жизни: работа; страна; семья; здоровье; закон; политика; культура. И быть компетентным в самых значимых сферах жизни – значит быть квалифицированным человеком.

Главная цель учебного процесса в школах – дать возможность обучающимся развивать свои навыки обучения, которые должны стать основой для того, чтобы они научились взаимодействовать с окружающим миром и жить в постоянно меняющемся мире. К сожалению, в настоящее время учителя-предметники, работающие в своих классах, чаще всего обеспокоены актуальностью и возможностью практического применения знаний учащихся в конкретной области, в которой учитель работает в данный момент.

В то же самое время, в общем основном образовании формирование базовых компетентностей, обучающихся является одной из главных задач общего основного образования.

Сейчас процесс обучения программированию не является интерактивным. Под интерактивностью подразумевается заинтересованность участников процесса. Из-за низкой заинтересованности обучающихся по таким программам появляется низкая компетентность «специалистов с дипломами».

В общем смысле интерактивность означает выбор наилучшего и наиболее благоприятного варианта среди различных возможных условий, средств, действий и т. д. Если перенести данный метод на процесс обучения программированию, то это означает выбор такого метода, который обеспечивает наилучшие результаты для преподавателей и учеников с наименьшими затратами времени и усилий в данных условиях. Выбранный метод должен удовлетворять самому главному требованию – актуальности материала.

Нередко встречаются случаи, когда обучающиеся учебных заведений при написании своих первых программ концентрируют внимание больше на строках контретного языка программирования, чем на алгоритме. В результате появляются проблемы:

* непонимание программируемых алгоритмов, запоминание лишь правильного порядка строк кода конкретного языка программирования;
* привязка к одному языку программирования, невозможность представить один и тот же алгоритм на другом языке.

Процесс обучения должен идти в ногу с развитием индустрии программирования и не основываться на устаревших языках. Объективно говоря, если посмотреть на текущее развитие ИТ-индустрии, то завтра может появиться новый язык программирования, библиотека или метод, который будет лучше, чем сегодняшний. Из-за такой скорости образовательные учреждения просто не успевают адаптироваться к инновациям.

Существуют критерии адаптации учебного процесса в интерактивный вид, которые необходимы для оценки методики формата обучения:

* максимально возможные результаты с точки зрения знаний, учебных навыков и формирования отношения;
* минимальное количество времени, необходимое учащимся и учителям для достижения определенного результата;
* минимальная затрата усилий, необходимая для достижения заданного результата в течение определенного времени;
* минимальные затраты ресурсов, необходимые для достижения заданного результата за определенное время.

Одним словом, процесс обучения является успешным, если его результат высок и может быть достигнут за короткий промежуток времени при минимальных усилиях со стороны учащегося и преподавателя.

Кроме того, обучающиеся могут выбирать последовательность своих курсов и скорость обучения. Например, если в обычном учебном заведении курс растягивается на один год, то курс интерактивного обучения можно пройти за короткое время, а можно растянуть на несколько лет.

Целью данной работы является спроектировать систему для автоматизации учебного процесса в школе, что позволит систематизировать большую часть важной информации, и сделать основной упор на программирование среди учеников. Облегчить учетелю анализ результатов работы обучающихся, повысить заинтересованность и рентабельность обучения программированию. Данная система позволит в большей части автоматизировать процесс работы с учениками на уроках: практические, самостоятельные и домашние работы. Также будет предусмотрен такой метод программирования, как «парное программирование». Все результаты учеников будут храниться в базе данных.

Благодаря этому учитель с помощью отчетов о выполнениях заданий может увидеть полную картину, которая будет отображать время, затраченное на выполнение задания, количество попыток и код пользователя. На основании этих объективных данных он сможет выводить объективную оценку знаний учеников по той или иной теме.

Более подробно особенности проекта рассматриваются в основной части пояснительной записки. Необходимые требования к структуре системы и функционалу содержатся в техническом задании. В информационном обеспечении рассматривается использование информационных ресурсов. Основные алгоритмы подсистем приведены в алгоритмическом обеспечении. Результаты тестирования системы на соответствие заявлены в техническом задании требованиям приводятся в соответствующем разделе.

Информация, которая в основном использовалась при написании выпускной квалификационной работы была взята из нескольких источников:

Источник [4] рассказывает об особенностях языка Python, его синтаксисе. Содержащаяся в нем информация является очень полезной при написании большей части кода для создания системы.

В источнике [1] приводятся примеры реализации на языке Python, используя веб-фраймворк Flask. Flask берёт на себя большую часть хлопот веб-разработки, поэтому вы можете сосредоточиться на написании своего веб-приложения без необходимости изобретать велосипед.

Работа с базой данных SQLite рассматривается в источнике [5]. Так как SQLite имеет высокую скорость работы с данными, позволяет хранить данные в одном файле, который имеет хорошую надежность. Поэтому было решено в данной работе использовать именно её.

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА СОЗДАНИЕ СИСТЕМЫ

## **1.1 Общие сведения**

### **1.1.1 Полное наименование системы и ее условное обозначение**

Полное наименование системы: «WEB-ориентированная система обучения программированию». Краткое наименование системы: «Система обучения программированию».

### **1.1.2 Перечень документов, на основании которых создается система**

1. ГОСТ Р 56846-2015 «Системы дистанционного обучения».
2. ГОСТ 34.601-90 «Автоматизированные системы. Стадии создания».
3. ГОСТ 34.603-92 «Виды испытаний автоматизированных систем».
4. РД 50-34.698-90 «Автоматизированные системы. Требования к содержанию документов».
5. Приказ «Об утверждении федерального государственного образовательного стандарта основного общего образования» Минобрнауки России от 17.12.2010 N 1897 (ред. от 11.12.2020).

### **1.2.2 Плановые сроки начала и окончания работы по созданию системы**

Начало работ по созданию системы: 01.02.2023 г.

Предварительное окончания работ: 10.06.2023 г.

## **1.2 Назначение и цели создания системы**

### **1.2.1 Назначение системы**

Предлагаемая система позволяет частично организовать автоматизацию процесса обучения программированию, в частности, за счет предоставления теоретического материала и самостоятельной работы обучающихся, а также частичной оптимизации практических работ. Система может быть внедрена в любом учебном заведении, занимающемся обучением программированию. Благодаря применению этой системы можно улучшить процесс обучения, сократить время, затрачиваемое на одного ученика, и тем самым повысить производительность. Такая система может снизить расходы учреждения на лицензии на программное обеспечение, аренду помещений и закупку оборудования.

### **1.2.2 Цели создания системы**

Цели создания системы:

1. Повысить заинтересованность программированием обучаемого за счет внедрения информационных технологий.
2. Уменьшить расходы образовательного учреждения на процесс обучения программированию.

## **1.3 Характеристика объекта автоматизации**

### **1.3.1 Краткие сведения об объекте автоматизации**

Объект автоматизации является элементом процесса обучения навыкам программирования в учебных заведениях. Изучение программирования в учебных заведениях является значимой частью любого образовательного учреждения. Сегодня знания в этой области очень широко востребованы, и потребность в таких специалистах растет повсеместно. Все больше и больше систем автоматизируется, а те, что были автоматизированы, теряют свою полезность, и требуются новые решения.

### **1.3.2 Сведения об условиях эксплуатации**

На данный момент в большинстве учебных заведений обучение программированию состоит из нескольких этапов:

* предоставление теоретического материала;
* проведение практических работ;
* самостоятельная работа обучающего в домашних условиях.

Зачастую стадии, предполагающие фактическое пребывание обучающихся и преподавателей в образовательном учреждении, не поддаются автоматизации.

На уроках преподаватели лично представляли учебные материалы. Несмотря на то, что на уроках используется оргтехника, им не хватает интерактивности и автоматизации, так как студентам приходится записывать материал в тетради для последующего использования.

На занятиях по лабораторным работам также не хватает автоматизации проверочных работ. Текущая ситуация такова, что преподаватель вынужден проверять работу каждого ученика, т.е. проверять код каждого студента. Подготовка к лабораторным работам часто проводится в теоретической форме.

Самостоятельная работа студентов может быть автоматизирована за счет использования вспомогательных программ и сторонних ресурсов. Однако эта работа также зависит от материала, представленного на занятиях, поэтому этот процесс также не может быть автоматизирован.

## **1.4 Требования к системе**

### **1.4.1 Требования к структуре и функционирования системы**

Ориентировочная структура системы:

В основе структуры системы лежат две подсистемы:

* + серверная;
  + клиентская.

Клиентская подсистема должна включать в себя основные функциональные модули:

* + содержание;
  + курс;
  + информация пользователя;
  + администрирование.

Данная система может работать при разных режимах.

Нормальный режим:

* клиентское программное обеспечение пользователей и администратора системы обеспечивают возможность круглосуточного функционирования с перерывами на обслуживание;
* серверное программное обеспечение и технические средства серверов обеспечивают возможность круглосуточного функционирования, с перерывами на обслуживание;
* исправно функционирует системное, базовое и прикладное программное обеспечение системы.

Для обеспечения нормального режима функционирования системы необходимо выполнять требования и выдерживать условия эксплуатации программного обеспечения.

Аварийный режим функционирования системы характеризуется отказом одного или нескольких компонентов программного обеспечения.

В случае перехода системы в аварийный режим необходимо:

* перезапустить серверное программное обеспечение;
* выполнить резервное копирование БД.

После этого необходимо выполнить комплекс мероприятий по устранению причины перехода системы в аварийный режим.

В перспективе данную систему можно модернизировать, улучшая ее характеристики.

### **1.4.2 Требования к численности и квалификации персонала системы**

* численность пользователей может быть неограниченная;
* квалификация администратора системы – среднее специальное техническое образование и выше. Квалификация других сотрудников не имеет значения;
* режим работы персонала не влияет на использование системы;
* порядок подготовки администратора системы – знакомство и подготовка к работе с основным функционалом, доступным для администратора.

### **1.4.3 Показатели назначения**

Система должна обеспечивать возможность одновременной работы нескольких пользователей.

### **1.4.4 Требования к надежности системы**

Система должна сохранять работоспособность и обеспечивать полное восстановление своих функций при возникновении следующих ситуаций:

* при сбоях в работе аппаратной части, приводящих к перезагрузке ОС сервера;
* при ошибках, связанных с программным обеспечением сторонних производителей (ОС, драйверы устройств и др.).

В системе должна быть обеспечена возможность восстановления данных с внешнего накопителя после восстановления активного накопителя.

### **1.4.5 Требования безопасности**

Обеспечение безопасности зависит не только от оснащенности объектов образования современными техническими устройствами и оборудованием, но от степени профессионализма управляющего этим оборудованием персонала, от грамотности и компетентности людей, отвечающих за безопасность образовательных организаций и учебного процесса, от слаженности их совместной работы с администрацией и педагогическими работниками.

### **1.4.6 Требования к защите информации от несанкционированного доступа**

Компоненты подсистемы должны обеспечивать: идентификацию пользователя; проверку полномочий пользователя при работе с системой; разграничение доступа пользователей к различным задачам системы.

### **1.4.7 Требования к стандартизации и унификации**

* система должна быть разработана в соответствии с требованиями существующих национальных стандартов и с учетом сферы ее применения, одной из которых является использование современных технологий, протоколов и стандартов для взаимодействия с внешними системами;
* все компоненты пользовательского интерфейса должны быть согласованы: форма, в которой информация отображается на экране, тип и форма информации, а также запросы к пользователю.

### **1.4.8 Требования к лицензионной и патентной частоте**

* система должна использовать только лицензионное программное обеспечение;
* установка системы в целом, как и установка отдельных частей системы не должна предъявлять дополнительных требований к покупке лицензий на программное обеспечение сторонних производителей.

## **1.5 Требования к функциям, выполняемым системой**

### **1.5.1 Функции авторизации, регистрации и контроля доступа**

Сроки реализации: первоочередная.

Входные данные: две строки, которые содержат логин и пароль соответственно. Данные строки вводятся пользователем в соответствующие формы.

Выходные данные: создание пользователя в базе данных и вывод доступных для него страниц таблиц на основе разрешений, установленных контролем доступа. Если этот пользователь является администратором, он должен иметь доступ к странице администрирования. Любой пользователь может получить доступ к своему профилю, который содержит информацию о нем.

### **1.5.2 Функция просмотра материалов обучения**

Сроки реализации: первоочередная.

Входные данные: содержимое главы, созданного на основе данных о темах и заданиях, содержащихся в БД.

Выходные данные: информация о содержании учебного материала выводится в виде структурированного списка.

### **1.5.3 Функция просмотра учебного материала из главы**

Сроки реализации: первоочередная.

Входные данные: учебный материал, который содержит в себе данные о содержимом курса и данные о заданиях главы, взятые из БД.

Выходные данные: информация о материале курса отображается в виде страницы, содержащей сам текст курса, а также список заданий, представленных в курсе. Каждое задание имеет уровень сложности, который отображается при создании задания и отражает важность задания.

### **1.5.4 Функция выполнения компиляции кода**

Сроки реализации: первоочередная.

Входные данные: программа, введенная обучаемым в текстовом редакторе, на языках программирования.

Выходные данные: в случае неудачного выполнения кода будет отображено сообщение об ошибке, полученное во время выполнения кода пользователя. Если выполнение кода прошло успешно, и пользователь был авторизован, информация о выполнении задания будет сохранена в профиле пользователя.

### **1.5.5 Функция добавления учебного материала**

Сроки реализации: не первоочередная.

Входные данные: обязательные поля: название блока в виде строки, id темы.

Выходные данные: на основе ввода дополнительных параметров определяется и сохраняется в базе данных нужная тема учебника. Если введен недопустимый параметр, отображается сообщение об ошибке.

### **1.5.6 Функция редактирования и удаления учебного материала**

Сроки реализации: первоочередная.

Входные данные: id нужной темы, содержимое темы в виде строки.

Выходные данные: редактирование или удаление содержимого указанного блока с учетом переданных параметров.

### **1.5.7 Функция отслеживания прогресса пользователем**

Сроки реализации: не первоочередная.

Входные данные: информация о содержании обучения, информация о заданиях, выполняемых обучаемым. Информация о содержании обучения содержит список заданий, доступных системе. Информация о заданиях содержит список заданий, выполненных пользователем.

Выходные данные: график с отображением проведенных обучаемым заданиях, список последних выполненных заданий.

### **1.5.8 Функция парного программирования**

Сроки реализации: не первоочередная.

Входные данные: программа, введенная обучаемыми в текстовом редакторе, на языках программирования. Возможность выбора ученика для добавления в парное программирование.

Выходные данные: в случае неудачного выполнения кода будет отображено сообщение об ошибке, полученное во время выполнения кода пользователя. Отображение напарника в отчете о выполнении задания.

### **1.5.9 Функция отправки жалобы на задание**

Сроки реализации: не первоочередная.

Входные данные: выбор задания и переход к нему, если по мнению пользователя задание не является корректным, то он может нажать кнопку «Жалоба».

Выходные данные: в случае отправки жалобы на задание, она приходит в личный кабинет администратора системы в раздел «Жалобы» для дальнейшего рассмотрения. Администратор в праве удовлетворить жалобу, либо же закрыть ее без обратной связи.

## **1.6 Требования к видам обеспечения**

### **1.6.1 Требования к информационному обеспечению**

Реляционная СУБД должна использоваться для хранения данных системы. Ввод данных для клиентского приложения должен быть автоматизирован. Все требуемые данные об учебном материале, используемом в приложении, могут быть введены с помощью форм приложения. Доступ к данным является многопользовательским. Время выполнения запросов не ограничено, но должно быть сведено к минимуму для повышения удобства пользования системой. Система не требует прямого доступа к данным с использованием языка запросов. Роли пользователей должны быть разграничены для предотвращения несанкционированного доступа к данным. Защита данных не требуется, так как в данном приложении не используется личная информация.

### **1.6.2 Требования к алгоритмическому обеспечению**

Требования к алгоритмическому обеспечению не предъявляются.

### **1.6.3 Требования к математическому обеспечению**

Требования к алгоритмическому обеспечению не предъявляются.

### **1.6.4 Требования к лингвистическому обеспечению системы**

Язык, используемый для взаимодействия с пользователем, должен быть русским.

### **1.6.5 Требования к метрологическому обеспечению**

Требования к метрологическому обеспечению не предъявляются.

### **1.6.6 Требования к программному обеспечению**

Данная система должна быть платформонезависимой. Структура программного обеспечения должна быть легко масштабируема и расширяема.

При разработке прикладного программного обеспечения необходимо использовать язык программирования Python 3.10. Для удобства разработки также необходимо использовать библиотеку Django для клиентской части, а также библиотеку socketserver для серверной части.

Взаимодействие между пользователем и клиентом должно осуществляться через графический наглядный интерфейс. Пользовательский интерфейс системы должен быть прозрачным и дружественным, не должен быть перенасыщен графическими элементами и должен предусматривать быстрое отображение в виде экранной формы. Навигационные элементы должны быть выполнены в удобной для пользователя форме. Ввод/вывод системных данных, прием команд управления и отображение результатов выполнения должны быть интерактивными. Графический интерфейс должен отвечать современным эргономическим требованиям и обеспечивать легкий доступ к основным функциям и операциям системы.

### **1.6.7 Требования к организационному обеспечению**

К работе с системой должны допускаться сотрудники, имеющие навыки работы на персональном компьютере, ознакомленные с правилами эксплуатации и безопасности.

## **1.7 Состав и содержание работ по созданию (развитию) системы**



### **1.7.1 Перечень стадий и этапов работ по созданию системы**

Разработка «Системы обучения языкам программирования» выполняется по стадиям ГОСТ 34.601-90 с объединением стадий теоретического и технического проектирования. Стадии работ и их содержание представлены в таблице 1.

Таблица 1 – Стадии работы по созданию системы

|  |  |  |
| --- | --- | --- |
| **Стадия** | **Этап** | **Работы** |
| Техническое задание | Разработка и утверждение технического задания | Разработка текста технического задания, согласование и утверждение. |
| Технический проект | Разработка проектных решений | Разработка решений по проектирование структур решений на python. |
| Разработка проектной документации | Разработка и оформление описаний проектных решений. |
| Рабочая документация | Разработка рабочей документации | Рабочая документация на «Система обучения программированию» в целом. |
|  | Разработка и адаптация программного обеспечения | Разработка классов.  Разработка структуры пакетов (папок) с разграничением доступа.  Разработка программы и методики испытаний. |
| Внедрение | Подготовка и проведение испытаний | Установка «Системы обучения программированию» на технических средствах учебного заведения.  Комплексная проверка функционирования.  Подготовка персонала учебного заведения, участвующего в испытаниях.  Проведение испытаний. |
|  | Доработка по результатам испытаний | Выполнение доработок программного обеспечения и документации. |
|  | Опытная эксплуатация | Установка обновленных программных средств.  Проведение методической подготовки сотрудников учебного заведения.  Сопровождение опытной эксплуатации. |

### **1.7.2 Перечень документов, предъявляемых по окончании стадий и этапов работ**

Перечень документов, предъявляемых по окончании соответствующих стадий и этапов работ, определяется в соответствии с ГОСТ 34.201-89 «Виды, комплектность и обозначение документов при создании автоматизированных систем», их перечисление в таблице 2.

Таблица 2 – Перечень документов

|  |  |
| --- | --- |
| **Стадия** | **Документы** |
| Техническое задание | Текст технического задания на создание «Системы обучения программированию» |
| Технический проект | Описание проектных решений:   * пояснительная записка. |
| Рабочая документация | Общее описание «Системы обучения программированию».  Руководство пользователя.  Алгоритмическое обеспечение системы. |

## **1.8 Требования к документированию**

### **1.8.1 Согласованный перечень подлежащих разработке комплектов и видов документов**

В процессе создания «Системы обучения программированию» разрабатывается следующая документация:

* общее описание «Системы обучения программированию»;
* пояснительная записка к техническому проекту;
* руководство пользователя.

Вся документация представляется на компакт дисках и один экземпляр каждого документа в виде твердой копии.

## **1.9 Источники разработки**

### **Перечень документов информационных материалов**

1. Поляков, К. Ю Информатика / К. Ю Поляков. – Лаборатория знаний. – М : БИНОМ, 2020. – 288 с.
2. Босова А. Ю. Информатика / А. Ю. Босова, Л. Л. Босова. – М. : БИНОМ. Лаборатория знаний, 2019. – 245 с.
3. [Федеральный закон от 29.12.2012 N 273-ФЗ (ред. от 29.12.2022) «Об образовании в Российской Федерации» (с изм. и доп., вступ. в силу с 11.01.2023)](https://www.consultant.ru/document/cons_doc_LAW_140174/).
4. Приказ Министерства просвещения РФ от 31 мая 2021 г. № 287 «Об утверждении федерального государственного образовательного стандарта основного общего образования».

## **1.10 Анализ аналогичных разработок**

***«Python.org»***

Система обеспечивает возможность доступа к высококачественным учебным пособиям по языку программирования Python. Такая система довольно популярна, поскольку содержит много полезной для пользователя информации. Кроме того, одним из преимуществ этой системы является то, что она бесплатна.

По сравнению с разрабатываемой системой здесь отсутствует возможность выполнения кода напрямую в этой системе. Данная система полагается на другие ресурсы, т.е. она зависит от сторонних ресурсов. На рисунке 1 главная страница «Python IDLE».

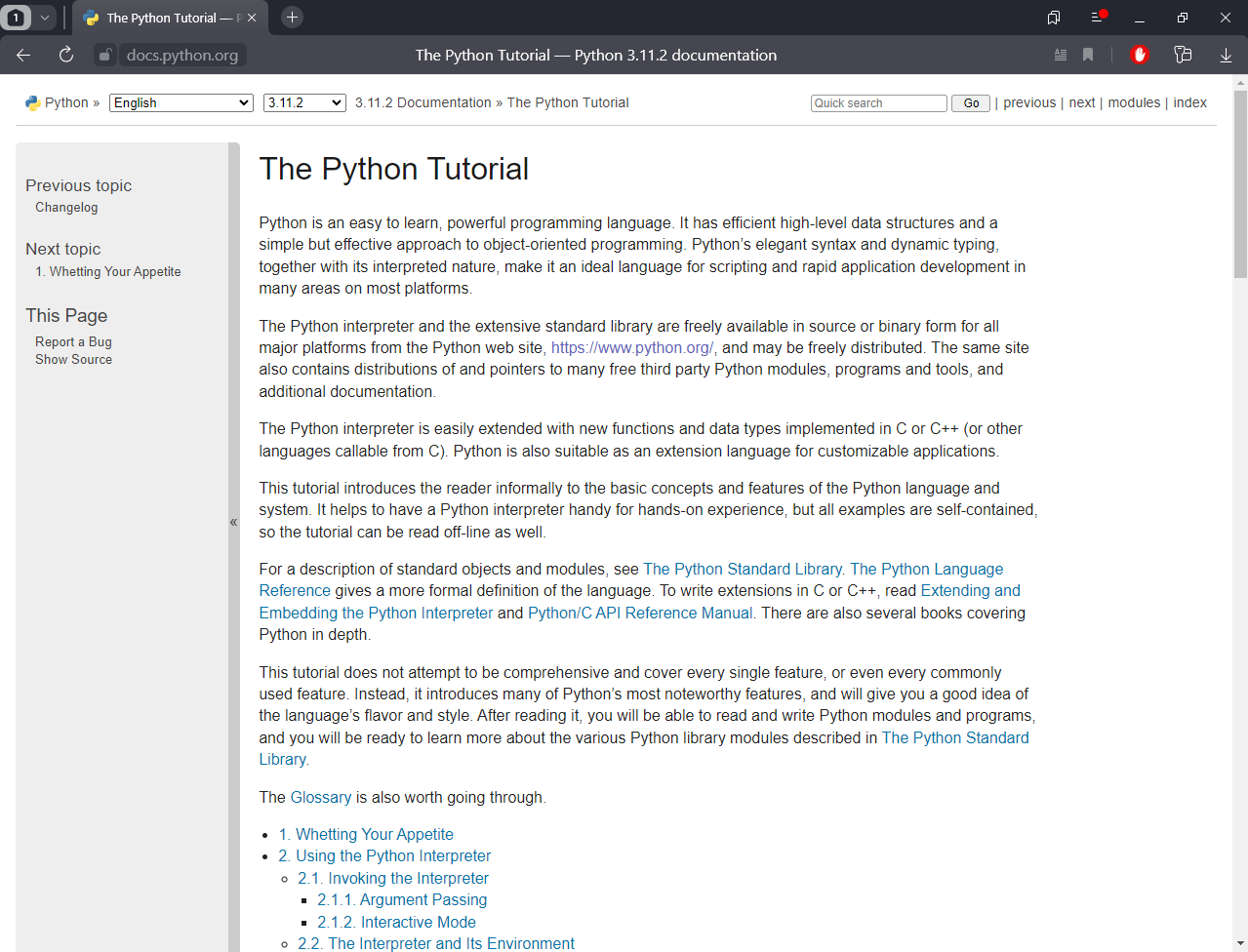


Рисунок 1 – Главная страница «Python.org»

***«w3schools.com»***

В отличие от прошлой системы, эта имеет собственный редактор кода. Помимо этого, данная система имеет огромное количество учебных курсов и тестовых заданий для тестирования ваших полученных навыков. Однако у этой системы есть один серьезный недостаток – она не распространяется свободно. На рисунке 2 показана главная страница этого ресурса.

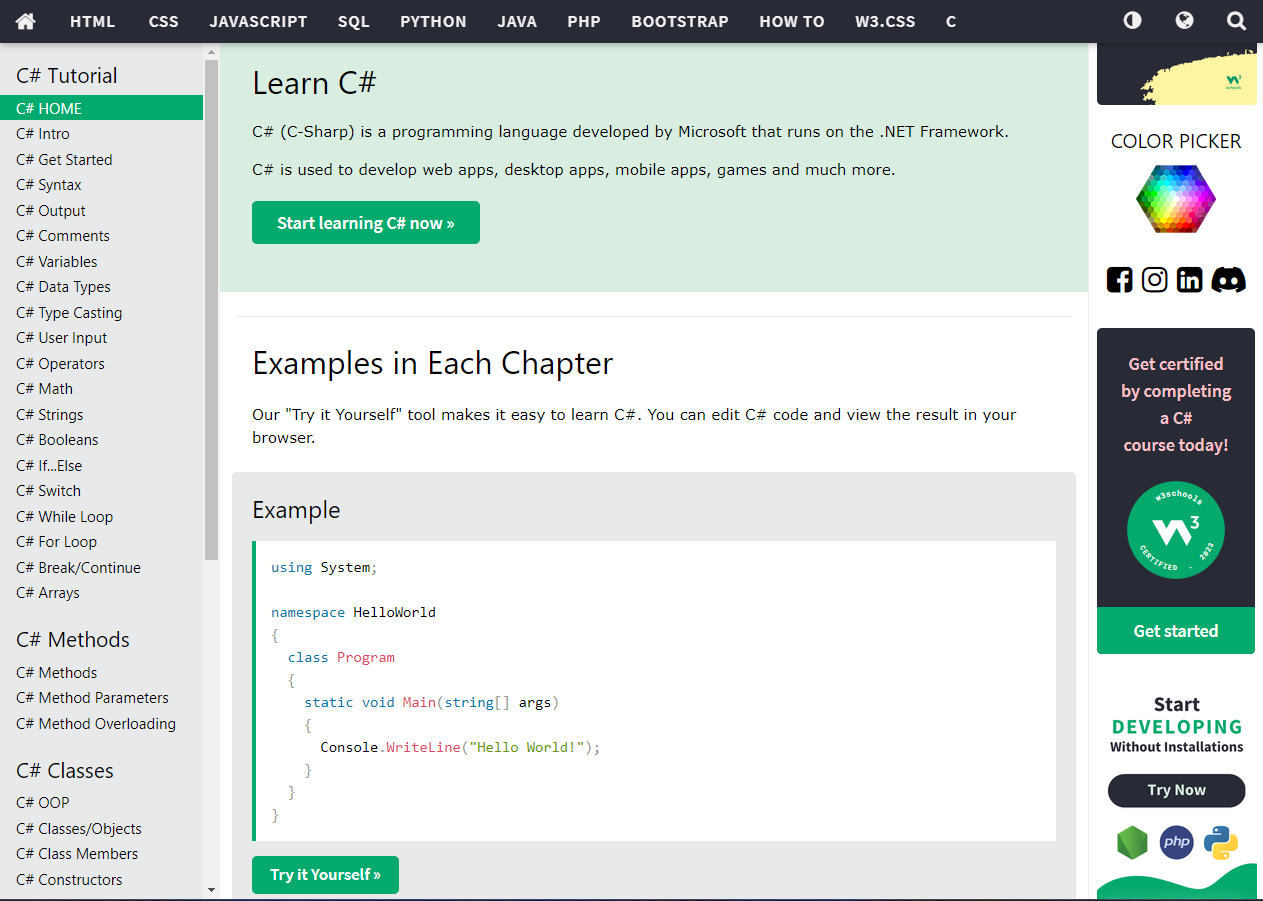


Рисунок 2 – Пример обучения на «w3schools.com»

***«ITVDN»***

В отличие от предшествующей системы, эта система имеет наставникиков, закрепленных за учениками. В ней также имеется большое количество обучающих уроков и тестов для проверки знаний, также встроенный редактор кода. Но у этой системы есть один очевидный недостаток - она не бесплатна. На рисунке 3 показана главная страница этого ресурса.

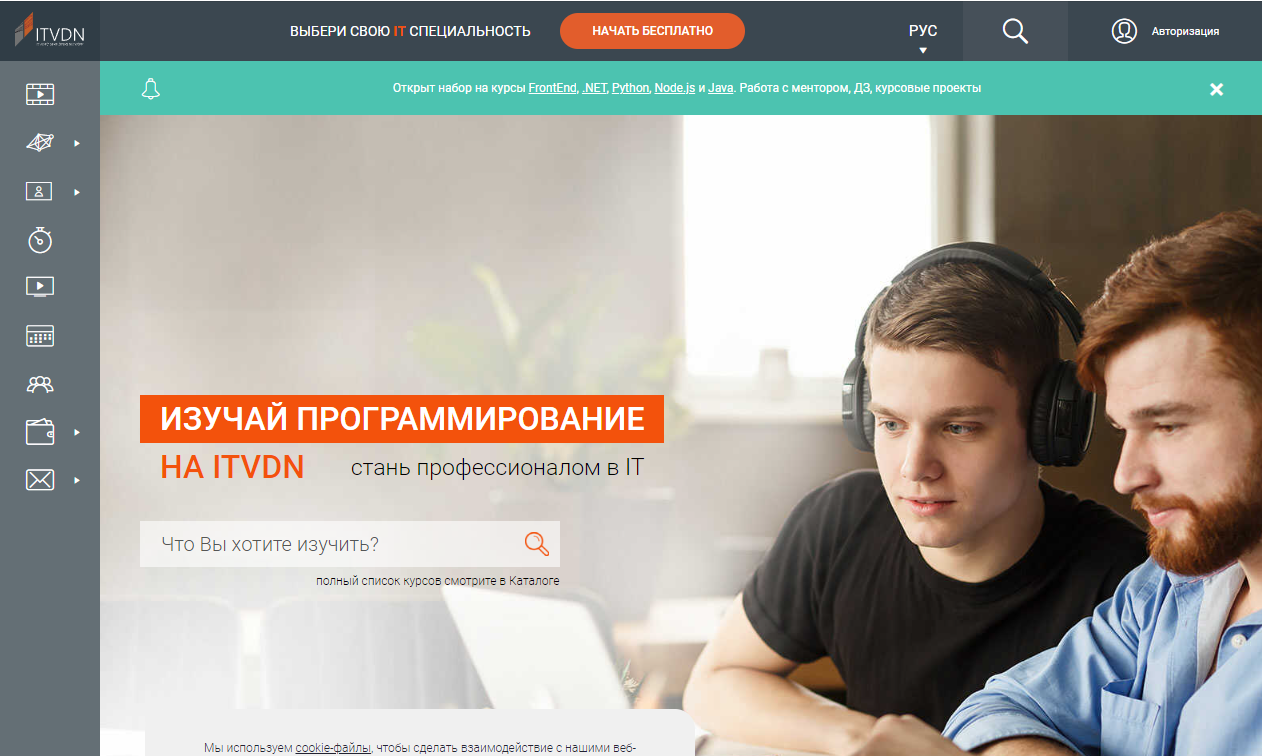


Рисунок 3 – Главная страница «ITVDN»

В таблице 3 проводится сравнение аналогов и разработанной системы.

Таблица 3 – Сравнение аналогов и разработанной системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Критерий** | **Python.org** | **w3schools.com** | **ITVDN** | **Разработанная ИС** |
| Свободное распространение | + | - | - | + |
| Закрепление куратора за пользователем | - | + | - | + |
| Возможность компилирования кода | - | + | + | + |
| Тестирование | - | + | + | - |
| Возможность прохождения обучения в свободной форме | - | + | + | + |
| Тёмная версия | - | - | - | + |
| Пользовательская безопасность | + | + | - | + |

Под пользовательской безопасностью подразумевается возможность привязыки своих данных, которые в дальнейшем могут быть использованы для восстановления доступа к системе. В разработанной ИС предусмотрено подтверждение по электронной почте.

Прохождение обучения в свободной форме означает выбор самостоятельного порядка выполнения заданий, в разработанной системе нет конкретных рамок и ограничений на выполнение.

В связи с тем, что система предполагается бесплатной и общедоступной, она должна быть достаточно востребована пользователями. Ожидается, что учебные заведения получат доступ к системе и затем развернут ее в своей среде, что позволит им частично автоматизировать процесс обучения.

# ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ

## **2.1 Выбор средств управления данными**

Для организации хранения и обработки данных было решено использовать реляционную СУБД. Эта система не требует хранения файлов, так как все данные, предоставляемые системой, выводятся в корректной форме, и нет необходимости в дополнительных файлах на стороне сервера. Кроме того, было решено использовать реляционную СУБД, а не нереляционную, так как между сущностями может быть установлена логическая связь, и таким образом использование реляционной базы данных избавит от необходимости отслеживать все зависимости между данными системы. В итоге мы остановили свой выбор на SQLite.

SQLite – это система управления базами данных, отличительной особенностью которой является то, что она встроена в приложение. Это означает, что большинство СУБД являются автономными приложениями с возможностью взаимодействия клиент-сервер. Клиентское приложение отправляет SQL-запрос, а СУБД (возможно, на удаленном компьютере) возвращает результаты запроса.

Например, при добавлении содержимого в приложение с помощью Redis (нереляционной базы данных) сразу возникает необходимость в четкой привязке каждого блока. В блоке должна быть предусмотрена часть, которая, в свою очередь, может содержать задачи. В SQLite все это управляется с помощью ограничений, тогда как в Redis придется добавлять эти поля в каждую сущность и проверять их на согласованность во время запроса. По сравнению с MongoDB, SQLite имеет внутреннюю систему пользователей, что делает использование SQLite более безопасным.

Поэтому использование SQLite не только безопаснее, но и лучше с точки зрения расширяемости приложения, поскольку позволяет обрабатывать большие объемы данных от разных пользователей, которые необходимо разграничивать.

## **2.2 Проектирование базы данных**

### **2.2.1 Концептуальная схема базы данных**

Модель «сущность-связь» представлена на рисунке 4.

В таблице 4 представлен список сущностей БД. В таблицах 5-13 представлены атрибуты сущности БД.

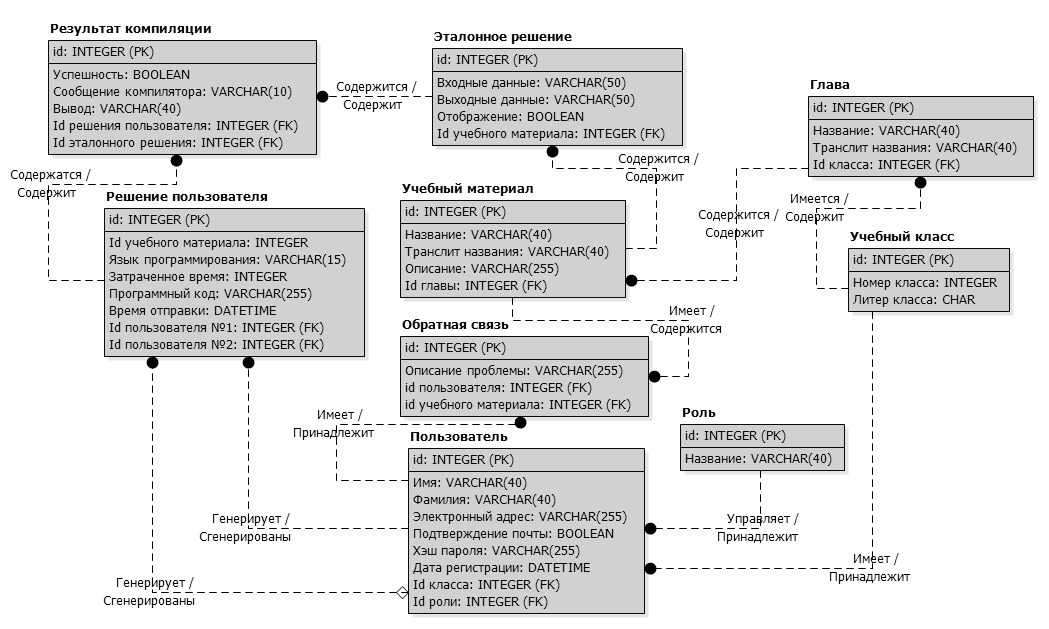


Рисунок 4 – Модель «сущность-связь»

Таблица 4 – Сущности концептуальной схемы

|  |  |  |
| --- | --- | --- |
| **№** | **Название сущности** | **Описание** |
| 1 | Результат компиляции | Результат компилирования кода |
| 2 | Решение пользователя | Хранятся данные о каждом решении, отправленном на компиляцию |
| 3 | Эталонное решение | Входные и выходные данные для проверки |
| 4 | Учебный материал | Задание главы |
| 5 | Глава | Глава учебного материала |
| 6 | Обратная связь | Сообщение о внештатной ситуации специалисту |

Продолжение таблицы 4

|  |  |  |
| --- | --- | --- |
| **№** | **Название сущности** | **Описание** |
| 7 | Пользователь | Пользователи системы |
| 8 | Учебный класс | Классы пользователей системы, в которых проходят обучение |
| 9 | Роль | Уровни доступа в системе (пользователь, администратор) |

Таблица 5 – Атрибуты сущности «Результат компиляции»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер результата |
| 2 | Id эталонного решения | Числовой | Номер теста, к которому относится решение |
| 3 | Id решения пользователя | Числовой | Номер данных для выполнения компиляции |
| 4 | Успешность выполнения | Логический | Успешность выполнения компиляции |
| 5 | Сообщение компилятора | Текстовый | Сообщение о выполнении (Success, ApiError, ExecError) |
| 6 | Вывод | Текстовый | Пользовательский вывод |

Таблица 6 – Атрибуты сущности «Решение пользователя»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер компиляции |
| 2 | Id учебного материала | Числовой | Номер задания, к которому относятся данные |
| 3 | Язык программирования | Текстовый | Язык компилирования |
| 4 | Затраченное время | Числовой | Время, которое пользователь затратил на компиляцию |
| 5 | Программный код | Текстовый | Код, который выполнил пользователь |
| 6 | Время отправки | Дата | Время отправки результата |
| 7 | Id пользователя №1 | Числовой | Номер пользователя №1 |
| 8 | Id пользователя №2 | Числовой | Номер пользователя №2 |

Таблица 7 – Атрибуты сущности «Эталонное решение»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер проверки |
| 2 | Id учебного материала | Числовой | Номер задания, к которому относится решение |
| 3 | Входные данные | Текстовый | Данные, поданные на вход программе |
| 4 | Выходные данные | Текстовый | Данные, которые пользователь должен вывести по итогу выполнения компиляции |
| 5 | Отображение | Логический | Отображение результата для просмотра пользователем |

Таблица 8 – Атрибуты сущности «Учебный материал»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер задания |
| 2 | Id главы | Числовой | Номер главы, в которой лежит задание |
| 3 | Название | Текстовый | Название задания |
| 4 | Транслит названия | Текстовый | Транслитное название |
| 5 | Описание | Текстовый | Описание задания |

Таблица 9 – Атрибуты сущности «Глава»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер главы |
| 2 | Id класса | Числовой | Номер класса, в котором числится пользователь |
| 3 | Название | Текстовый | Название главы |
| 4 | Транслит название | Текстовый | Транслитное название |

Таблица 10 – Атрибуты сущности «Обратная связь»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер отчета |
| 2 | Id пользователя | Числовой | Номер пользователя в системе |
| 3 | Id задания | Числовой | Номер задания в системе |

Продолжение таблицы 10

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 4 | Описание проблемы | Текстовый | Текст обращения к администратору системы |

Таблица 11 – Атрибуты сущности «Пользователь»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер пользователя |
| 2 | Имя | Текстовый | Имя пользователя |
| 3 | Фамилия | Текстовый | Фамилия пользователя |
| 4 | Электронный адрес | Текстовый | Электронный адрес, использованный при регистрации |
| 5 | Подтвержение | Логический | Подтвержение пользователя |
| 6 | Id роли | Числовой | Роль в системе |
| 7 | Id класса | Числовой | Номер класса, в котором числится пользователь |
| 8 | Хэш пароля | Текстовый | Хэш пароля пользователя |
| 9 | Дата регистрации | Дата | Дата регистрации |

Таблица 12 – Атрибуты сущности «Учебный класс»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер класса |
| 2 | Id класса | Числовой | Цифра от 5 до 11 |
| 3 | Литер класса | Текстовый | Символ (А, Б, В) |

Таблица 13 – Атрибуты сущности «Роль»

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | Id | Числовой | Номер роли |
| 2 | Название | Числовой | Название роли |

### **2.2.2 Внутренняя схема базы данных**

В таблицах 14-24 представлены физические модели БД.

Таблица 14 – Физическая модель test\_result [Результат компиляции]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_test\_res [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Task\_id [Номер задания] | INTEGER | 1 | 1…2147483647 |
| 3 | Submission\_id [Номер данных] | INTEGER | 1 | 1…2147483647 |
| 4 | Success [Выполненность] | BOOLEAN | - | True/False |
| 5 | Message [Сообщение] | VARCHAR(MAX) | - | Без ограничений |
| 6 | User\_output [Пользовательский результат] | VARCHAR(MAX) | - | Без ограничений |

В таблице test\_result в качестве индекса используется первичный ключ по полю Id\_test\_res, который является уникальным и сортируется по возрастанию.

Таблица 15 – Физическая модель submissions [Решение пользователя]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_sub [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Task\_id [Номер задания] | INTEGER | 1 | 1…2147483647 |
| 3 | Language [Язык] | VARCHAR(255) | - | - |
| 4 | Passed\_tests [Результат] | INTEGER | 1 | 1…2147483647 |
| 5 | Sourse\_code [Код компилирования] | TEXT | - | Без ограничений |
| 6 | Submission\_date [Время выполнения] | DATETIME | - | 12.01.2023 |

В таблице submissions в качестве индекса используется первичный ключ по полю Id\_sub, который является уникальным и сортируется по возрастанию.

Таблица 16 – Физическая модель user\_submissions [Привязка данных]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_user [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Submissions\_id [Номер данных] | INTEGER | 1 | 1…2147483647 |

В таблице user\_submissions в качестве индекса используется первичный ключ по полю Id\_user, который является уникальным и сортируется по возрастанию.

Таблица 17 – Физическая модель tests [Эталонное решение]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_test [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Task\_id [Номер задания] | INTEGER | 1 | 1…2147483647 |
| 3 | Stdin [Входные данные test] | TEXT | - | Без ограничений |
| 4 | Stdout [Выходные данные test] | TEXT | - | Без ограничений |
| 5 | Is\_hidden [Отображение] | BOOLEAN | false | True/False |

В таблице tests в качестве индекса используется первичный ключ по полю Id\_test, который является уникальным и сортируется по возрастанию.

Таблица 18 – Физическая модель tasks [Учебный материал]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_task [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Topic\_id [Номер главы] | INTEGER | 1 | 1…2147483647 |
| 3 | Name [Название] | VARCHAR(50) | - | Без ограничений |
| 4 | Translit\_name [Транслит] | VARCHAR(50) | - | Без ограничений |
| 5 | Text [Текст задания] | TEXT | - | Без ограничений |

В таблице task в качестве индекса используется первичный ключ по полю Id\_task, который является уникальным и сортируется по возрастанию.

Таблица 19 – Физическая модель topics [Глава]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_topic [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Grade\_id [Номер класса] | INTEGER | 1 | 1…2147483647 |
| 3 | Name [Название] | VARCHAR(50) | - | Без ограничений |
| 4 | Translit\_name [Транслит] | VARCHAR(50) | - | Без ограничений |

В таблице topic в качестве индекса используется первичный ключ по полю Id\_topic, который является уникальным и сортируется по возрастанию.

Таблица 20 – Физическая модель examples [Данные для выполнения]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_example [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Task\_id [Номер задания] | INTEGER | 1 | 1…2147483647 |
| 3 | Example\_input [Входные данные] | TEXT | - | Без ограничений |
| 4 | Example\_output [Выходные данные] | TEXT | - | Без ограничений |

В таблице examples в качестве индекса используется первичный ключ по полю Id\_ example который является уникальным и сортируется по возрастанию.

Таблица 21 – Физическая модель reports [Обратная связь]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_report [id] | INTEGER | 1 | 1…2147483647 |
| 2 | User\_id [Номер пользователя] | INTEGER | 1 | 1…2147483647 |
| 3 | Task\_id [Номер задания] | INTEGER | 1 | 1…2147483647 |
| 4 | Text [Условие задания] | TEXT | - | Без ограничений |

В таблице reports в качестве индекса используется первичный ключ по полю Id\_ report который является уникальным и сортируется по возрастанию.

Таблица 22 – Физическая модель users [Пользователь]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_user [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Name [Имя] | VARCHAR(50) | 1 | 1…2147483647 |
| 3 | Surname [Фамилия] | VARCHAR(50) | 1 | 1…2147483647 |
| 4 | Email [Электронный адрес] | VARCHAR(50) | - | Без ограничений |
| 5 | Verified [Подтверждение] | BOOLEAN | false | True/False |
| 6 | Role\_id [Номер роли] | INTEGER | 1 | 1…2147483647 |
| 7 | Grade\_id [Номер класса] | INTEGER | 1 | 1…2147483647 |
| 8 | Grade\_letter [Литер класса] | VARCHAR(50) | - | Без ограничений |
| 9 | Hasged\_password [Хэш пароля] | VARCHAR(50) | - | Без ограничений |
| 10 | Registration\_date [Дата регистрации] | DATETIME | - | 00:00:00 12.01.2023 |

В таблице users в качестве индекса используется первичный ключ по полю Id\_ user который является уникальным и сортируется по возрастанию.

Таблица 23 – Физическая модель grades [Учебный класс]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_grade [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Number [Номер класса] | INTEGER | 1 | 1…2147483647 |

В таблице grades в качестве индекса используется первичный ключ по полю Id\_ grade который является уникальным и сортируется по возрастанию.

Таблица 24 – Физическая модель roles [Роль]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | Id\_role [id] | INTEGER | 1 | 1…2147483647 |
| 2 | Name [Название роли] | VARCHAR(10) | - | Без ограничений |

В таблице roles в качестве индекса используется первичный ключ по полю Id\_ role который является уникальным и сортируется по возрастанию.

## **Организация сбора, передачи, обработки и выдачи информации**

Исходная информация вносится администратором вручную через систему. Информация вносится вручную, так как данные учебной системы не должны часто обновляться и поэтому нет необходимости автоматизировать обновление учебного материала.

Данные передаются на сервер средствами AJAX-запроса. Затем сервер участника отправляет асинхронные запросы в [WandboxAPI](https://github.com/melpon/wandbox). Здесь выполняется код, а затем сервер участника обрабатывает результаты выполнения и возвращает JSON, который преобразуется в HTML**-**код. Данная технология была выбрана из-за того, что она наиболее удобна в использовании и не требует дополнительных библиотек, как на сервере, так и на клиенте и полностью удовлетворяет потребности системы.

Данные проверяются средствами СУБД при помощи ограничений, которые были установлены на таблицах.

Обработка и приведение данных к нужному виду происходит на клиентской части приложения. Серверная часть приложения отвечает за проверку валидности данных и их хранение в БД.

# АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ

## **Алгоритм регистрации нового пользователя**

**Общая характеристика:** алгоритм предназначен для регистрации пользователя в системе.

**Используемые данные:** входными данными являются электронный адрес и пароль пользователя в виде строк, длиной не более 255 символов. При вводе входных данных система определяет существует ли пользователь с таким электронным адресом в базе данных.

**Результаты выполнения:** результатом выполнения алгоритма является регистрация пользователя в системе и создание персонального аккаунта. Если пользователь с такой электронной почтой существует, система выдаёт ошибку «Пользователь с этой почтой уже зарегистрирован!», если нет, происходит процесс регистрации в системе и создание нового аккаунта пользователя.

**Логическое описание:** схема алгоритма (рисунок 5).

## **Алгоритм редактирования учебного материала**

**Общая характеристика:** алгоритм предназначен для редактирование учебного материала в системе.

**Используемые данные:** на вход передается название задания для редактирования, новое содержимое.

**Результаты выполнения:** результатом выполнения алгоритма является изменение определенного задания в учебном материале. Если задание отсутствует в бд, то система выдаст ошибку «Такое задание не существует!», в противном случае, система позволит отредактировать содержимое учебного материала.

**Логическое описание:** схема алгоритма (рисунок 6).

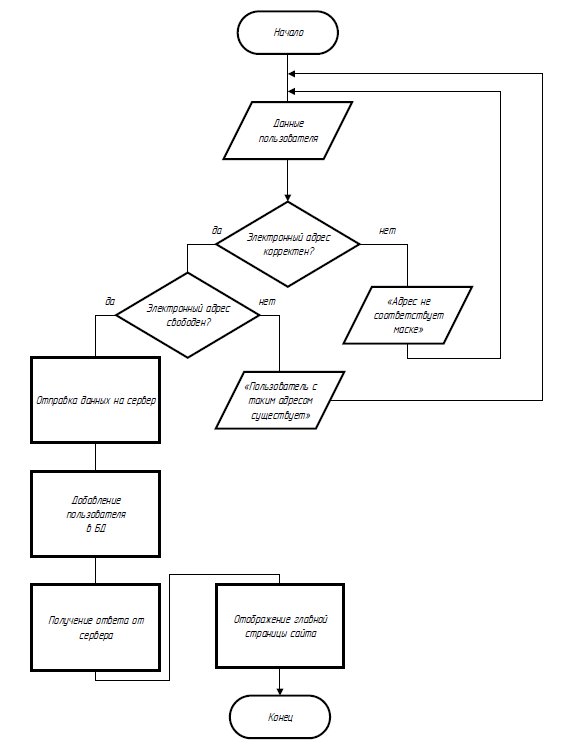


Рисунок 5 – Схема алгоритма регистрации пользователя

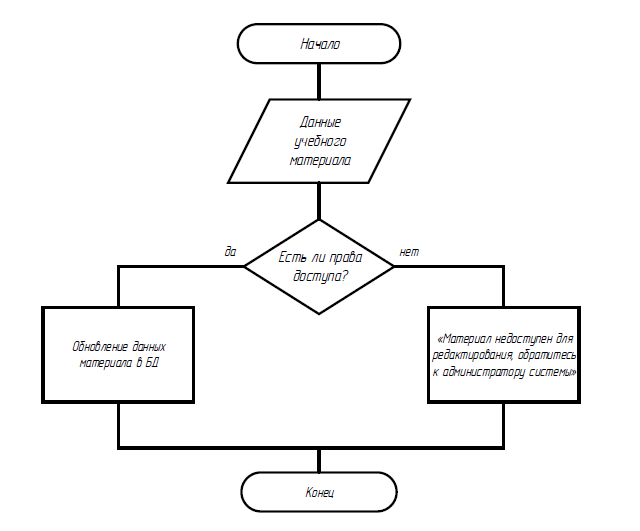


Рисунок 6 – Схема алгоритма редактирования учебного материала

## **Алгоритм компиляции программного кода примеров**

**Общая характеристика:** алгоритм предназначен для выполнения компиляции простых практических примеров из учебного материала в системе.

**Используемые данные:** на вход подается программма, написанная в редакторе пользователем, согласно условию задания для выполнения.

**Результаты выполнения:** результатом выполнения алгоритма является выполнение практического задания в учебном материале, либо же его провал. Если компиляция прошла успешна, то система выведет пользователю сообщение «Success», в противном случае, система система выведет исключение о том, что компиляция завершилась некорректно «ExecutionError».

**Логическое описание:** схема алгоритма (рисунок 7).

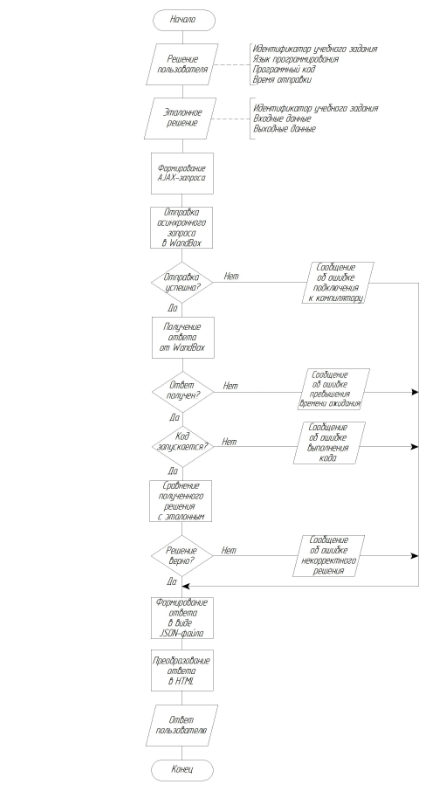


Рисунок 7 – Схема алгоритма компиляции программного кода примеров

# ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ

## **Структура программного обеспечения и функции его компонентов**

ОС – ALT Linux.

Инструментальное средство разработки – VSCode.

Язык программирования – Python 3.10.

Вспомогательное программное обеспечение – Yandex Browser

## **Выбор компонентов программного обеспечения**

### **Операционная система**

Для данного программного продукта подходит любая современная ОС. Поэтому ОС для разворачивания сервера можно определять по основным отличительным признакам. Например, можно выбрать любую, свободно распространяемую UNIX-подобную систему, так как системы Windows платные. Также выбирая UNIX-подобную систему можно выбирать ту, которая будет требовать наименьшее количество ресурсов от машины, на которой будет развернута. Более подробное сравнение приведено в таблице 27.

Таблица 25 – Сравнение операционной системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Критерий сравнения** | **Windows** | **Ubuntu** | **ALT Linux** |
| 1 | Бесплатная | - | + | + |
| 2 | Нетребовательная к системе | - | - | + |
| 3 | Удобная работа с консолью | + | + | + |

Поэтому для разворачивания серверной системы предлагается использовать ALT Linux.

### **Инструментальное средство разработки и язык программирования**

Для клиентской части при выборе языка программирования выбор стоит довольно легкий, так как у языка Python абсолютная монополия в веб-приложениях и выбирать особо не приходится. Для облегчения разработки было решено использовать библиотеку Flask. Она ускоряет разработку в разы и имеет большое количество разных вспомогательных инструментов. Выбирая Flask, выбор стоял между самим Flask, а также двумя фреймворками Turbogears и Web2py. Выбор пал на Flask по нескольким причинам:

1. Приложения Flask по умолчанию защищены от распространенных уязвимостей и атак безопасности.
2. В нем реализованы строгие правила и соглашения, которым должны следовать разработчики. Хотя это может быть ограничивающим, оно также обеспечивает стандартизированные методы кодирования, несложное тестирование и отладку, и рабочую архитектуру приложения.
3. Для удобства доступа доступны маршрутизация URL, механизм шаблонов, мощный ORM и множество миграций схемы базы данных.

Для серверной части использовалась среда выполнения Python – Apache2. Apache2 – является кроссплатформенным программным обеспечением, поддерживает популярные операционные системы такие как Windows, Linux, Mac OS. В серверной части у Python монополии нет и выбор языков программирования очень большой. Основной выбор стоял между JavaScript, Java и C#. Java и C# больше подходят для высоконагруженных приложений, где есть необходимость в обработке большого количества данных и запросов. Но несмотря на то, что Python и не имеет всех тех преимуществ, которые имеют Java и C#, он вполне подходит для выполнения основной задачи, которая заложена в этом приложении. Также используя Python, мы уменьшаем количество технологий, которые должен знать разработчик данной системы.

При выборе инструментального средства разработки выбор пал на VSCode, так как он имеет большое количество надстроек и удобен в использовании. Это средство разработки довольно гибкое в плане настроек и его можно сделать «под себя». Сравнение VSCode с некоторыми конкурентами приведено в таблице 28.

Таблица 26 – Сравнение инструментального средства разработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Критерий сравнения** | **VSCode** | **Python IDLE** | **Notepad++** |
| 1 | Большое количество плагинов | + | - | - |
| 2 | Настройка под себя | + | + | + |
| 3 | Подсветка кода | + | + | + |
| 4 | Интегрированная консоль | + | - | - |

### **Вспомогательное программное обеспечение**

Для просмотра результатов работы необходим браузер, в котором удобно работать, который поддерживает последние возможности языка HTML и JS. Выбирая браузер, выбор стоит зачастую не между самими браузерами, а их движками. Например, Google Chrome и Opera используют один движок, поэтому они одинаковы в плане удобств для разработчика. Yandex Browser написан на своем собственном движке и имеет некоторые отличия. Internet Explorer также написан на своем собственном движке и имеет наибольшее количество отличий от других браузеров. Сравнение приводится в таблице 29.

Таблица 27 – Сравнение браузеров

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Критерий сравнения** | **Google Chrome** | **Yandex Browser** | **Internet Explorer** |
| 1 | Движок | V8 | Blink | Trident |
| 2 | Удобная панель рабработчика | + | + | - |
| 3 | Поддержка последних стандартов | + | + | + |
| 4 | Урезанный интерфейс | + | - | + |

По итогу выбор пал на Yandex Browser, так как движок, который используется в нем, также используется и в Opera и в других браузерах, что означает, что, используя Yandex Browser мы можем быть уверены, что и во многих других браузерах отображение будет идентичным.

## **Разработка прикладного программного обеспечения**

### **Структура прикладного программного обеспечения**

Разработанная система имеет следующую структуру, представленную в таблице 28.

Таблица 28 – Структура разработанной системы

|  |  |  |
| --- | --- | --- |
| **Элемент** | | **Описание элемента** |
| **app.py** | | Головной модуль приложений, определяющий основные настройки и реализующий веб-сервер |
| **config.py** | | Файл, содержащий конфигурации для переменных |
| **README.md** | | Презентация основных функций системы |
| **requirements.txt** | | Список используемых версий для стабильной работы системы |
| **app** | blueprint | Папка содержит реализацию основных модулей веб-приложения |
| contester | Папка, содержащая реализацию программного интерфейса  приложения (API) |
| db | Папка, в которой содержится дамп базы данных |
| static | Папка содержит файлы css, js для оформления интерфейса веб-страниц для роли «пользователь» |
| templates | Папка, содержащая шаблоны html для оформления веб-страниц для роли «пользователь» |
| utils | Папка содержит вспомогательные функции. Например, создание цепочек навигации, которые отображают путь пользователя от корня сайта до текущей страницы |
| \_init\_.py | Файл инициализации |
| forms.py | Файл, содержащий формы для взаимодействия с пользователем |
| models.py | Файл, содержащий основные классы (модели) |
| routes.py | Файл маршрутизации |
| **docs** | | Папка, содержащая руководство пользователя для ролей «администратор» и «пользователь» |
| **migrations** | | Папка, содержащая конфигурацию единой базы данных для Flask |
| **tests** | | Папка тестирования разработанной системы |

### **Реализация представлений в веб-приложении**

Используемый фреймворк Flask позволяет разделять сложный проект на набор независимых модулей, при чем каждый модуль имеет свои шаблоны, стили оформления и наборы изображения.

Основные наборы модулей представлений, разработанные для данного веб-приложения:

* модуль «admin» (содержит представления для административной панели);
* модуль «api» (содержит представления для API (программного интерфейса приложения));
* модуль «auth» (содержит представления для регистрации и авторизации пользователей);
* модуль «errors» (содержит представления для вывода ошибок).

Структура каждого модуля соответствует одному шаблону, который представлен в таблице 29.

Таблица 29 – Структура модулей

|  |  |
| --- | --- |
| **Модуль** | **Описание модуля** |
| **static** | Папка, содержащая файлы css, js для оформления интерефейса |
| **templates/admin** | Папка, содержащая шаблоны html |
| **\_\_init\_\_.py** | Файл инициализации |
| **forms.py** | Файл, содержащий поля классов для реализации форм веб-интерфейса |
| **routes.py** | Файл маршрутизации |

### **4.3.3 Реализация моделей, используемых в веб-приложении**

Модели данных в разработанном веб-приложении представлены в файле models.py и имеют реализацию, представленную в таблицах 30-39.

Таблица 30 – Модель данных class User [Пользователь]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор пользователя |
| 2 | name | Имя пользователя |
| 3 | surname | Фамилия пользователя |
| 4 | email | Электронный адрес |
| 5 | verified | Подтверждение электронной почты |
| 6 | role\_id | Идентификатор роли пользователя |
| 7 | grade\_id | Идентификатор класса |
| 8 | hashed\_password | Хэш пароля |
| 9 | registration\_date | Дата регистрации |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def set\_password (self, password) | Метод, генерирующий хэш пароля для параметра password |
| 2 | def check\_password (self, password) | Метод, сравнивающий параметр password с параметром hashed\_password |
| 3 | def is\_admin (self) | Метод, проверяющий роль пользователя на соответствие роли «администратор» |
| 4 | def classmates (self) | Метод, возвращающий список одногруппников пользователя |
| 5 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 31 – Модель данных class Grade [Класс]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор класса |
| 2 | name | Номер класса |

Продолжение таблицы 31

|  |  |  |
| --- | --- | --- |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 32 – Модель данных class Role [Роль]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор роли |
| 2 | name | Название роли |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 33 – Модель данных class Topic [Глава]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор главы |
| 2 | grade\_id | Идентификатор класса |
| 3 | name | Название главы |
| 4 | translit\_name | Транслит названия главы |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def set\_translit\_name (self) | Метод, возвращающий транслитерацию параметра name |
| 2 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 34 – Модель данных class Task [Учебный материал]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор учебного материала |
| 2 | topic\_id | Идентификатор главы |
| 3 | name | Название учебного материала |
| 4 | translit\_name | Транслит названия учебного материала |
| 5 | text | Описание учебного материала |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def set\_translit\_name (self) | Метод, возвращающий транслитерацию параметра name |
| 2 | def url (self, tab: t.Optional[str] = None) | Метод, формирующий url-адрес для шаблона учебного материала (Класс – Название главы – Название учебного материала) |
| 3 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 35 – Модель данных class Example [Данные для проверки]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор данных для проверки |
| 2 | task\_id | Идентификатор учебного материала |
| 3 | example\_input | Пример входных данных |
| 4 | example\_output | Пример выходных данных |

Таблица 36 – Модель данных class Test [Данные для выполнения]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор данных для выполнения |
| 2 | task\_id | Идентификатор учебного материала |
| 3 | stdin | Данные для выполнения на вход |
| 4 | stdout | Данные для выполнения на выход |
| 5 | is\_hidden | Скрыть данные |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 37 – Модель данных class Submission [Данные об отправках]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор данных об отправках |
| 2 | task\_id | Идентификатор учебного материала |
| 3 | language | Язык программирования |
| 4 | passed\_tests | Затраченное время |
| 5 | source\_code | Код |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def processed\_code(self) | Метод, обрабатывающий параметр soure\_code, а именно заменяющий некоторые символы исходного кода на корректные |
| 2 | def get\_result(self) | Метод, возвращающий сообщение пользователю о результате проверки кода |

Продолжение таблицы 37

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Описание** |
| 3 | def \_\_repr\_\_(self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 38 – Модель данных class TestResult [Результат компиляции]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор результата компиляции |
| 2 | test\_id | Идентификатор данных для выполнения |
| 3 | submission\_id | Идентификатор данных об отправках |
| 4 | succes | Успешность |
| 5 | message | Сообщение |
| 6 | user\_output | Вывод |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

Таблица 39 – Модель данных class Report [Отчет]

|  |  |  |
| --- | --- | --- |
| **Поля** | | |
| **№** | **Название поля** | **Описание** |
| 1 | id | Идентификатор отчета |
| 2 | user\_id | Идентификатор пользователя |
| 3 | task\_id | Идентификатор учебного материала |
| 4 | text | Описание |

Продолжение таблицы 39

|  |  |  |
| --- | --- | --- |
| **Методы** | | |
| **№** | **Название метода** | **Описание** |
| 1 | def \_\_repr\_\_ (self) | Метод, возвращающий текстовое или строковое представление сущности |

### **Реализация основных модулей веб-приложения**

По основным функциям, описанным в п. 4.2 Технического задания, в разработанном веб-приложении были реализованы следующие модули:

* модуль авторизации, регистрации и разграничения доступа;
* модуль работы с учебным материалом;
* модуль компиляции кода пользователя;
* модуль взаимодействия между пользователем и подсистемой   
  решения задач.

Некоторые из данных модулей реализованы в папке blueprints, где содержатся веб-представления, данная структура упрощает логику взаимодействия между модулями. Однако основной модуль компиляции кода, написанного пользователем, реализован в папке contester.

#### **Модуль авторизации, регистрации и разграничения доступа**

Данный модуль реализован в файле app/blueprints/api/auth.py. Количество строк кода: 85. Основные методы данного модуля описаны в таблице 40.

Таблица 40 – Методы модуля авторизации, регистрации и разграничения доступа

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Описание** |
| 1 | def signup() | Метод регистрации пользователя в системе, который осуществляет проверку на уникальность адреса электронной почты и совпадения пароля и повторного пароля, при успешной проверке создается новая запись в БД в таблице «Пользователь» с ролью «user» |

Продолжение таблицы 40

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Описание** |
| 2 | def login() | Метод авторизации пользователя в системе, который осуществляет проверку на совпадение для пары «адрес электронной почты – пароль» введенных пользователем и находящихся в БД, при успешном совпадении запускается сессия для данного пользователя с определенными правами доступа для роли данного пользователя |
| 3 | def logout() | Метод, осуществляющий выход пользователя из системы с закрытием сессии |
| 4 | def confirm\_email() | Метод подтверждения электронной почты при помощи отправки письма с подтверждением на введенный пользователем адрес электронной почты |

#### **Модуль работы с учебным материалом**

Данный модуль реализован в файле app/blueprints/api/admin.py. Количество строк кода: 113. Основные методы данного модуля описаны в таблице 41.

Таблица 41 – Методы модуля работы с учебным материалом

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Описание** |
| 1 | def create\_topic() | Метод, создающий новую главу учебного материала на основе данных, полученных от пользователя через форму, также осуществляется проверка на уникальность названия главы, при успешном создании главы создается новая запись в БД в таблице «Глава» |
| 2 | def update\_topic(topic\_id) | Метод редактирования главы учебного материала на основе данных, полученных от пользователя через форму, при успешном обновлении главы запись в БД в таблице «Глава» обновляется |
| 3 | def create\_task() | Метод создания задания учебного материала на основе данных (название, описание, данные для проверки, данные для выполнения), полученных от пользователя через форму, также осуществляется проверка на уникальность названия учебного материала, при успешном добавлении задания создаются новые записи в БД в таблицах «Учебный материал», «Данные для проверки», «Данные для выполнения» |

Продолжение таблицы 41

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Описание** |
| 4 | def update\_task(task\_id) | Метод редактирования задания учебного материала на основе данных (название, описание, данные для проверки, данные для выполнения), полученных от пользователя через форму, также осуществляется проверка на уникальность названия учебного материала, при успешном редактирования задания записи в БД в таблицах «Учебный материал», «Данные для проверки», «Данные для выполнения» обновляются |
| 5 | def delete\_task() | Метод удаления задания учебного материала |

#### **Модуль взаимодействия между пользователем и подсистемой решения задач**

Данный модуль реализован в файле app/blueprints/api/task.py. Количество строк кода: 78. Основные методы данного модуля описаны в таблице 42.

Таблица 42 – Методы модуля взаимодействия между пользователем и подсистемой решения задач

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Описание** |
| 1 | def render\_solution\_failure(message: str) | Метод, возвращающий шаблон веб-страницы, используемый для сообщения пользователю об ошибке при выполнении задачи |
| 2 | def render\_solution\_success  (response: ContesterResponse) | Метод, возвращающий шаблон веб-страницы, используемый для сообщения пользователю об успешном выполнении задачи |
| 3 | def send\_solution() | Метод отправки решения на сервер для дальнейшей компиляции кода пользователя, который также проверяет регистрацию пользователя и верификацию электронной почты пользователя и его партнера |
| 4 | def send\_report() | Метод отправки сообщения пользователя о проблеме при выполнении задачи |
| 5 | def delete\_report() | Метод удаления сообщения пользователя о проблеме при выполнении задачи с предоставлением обратной связи |

#### **Модуль компиляции кода пользователя**

Данный модуль реализован в папке contester, файл с основными функциями расположен в файле api\_service.py. Количество строк кода: 80. Основные методы данного модуля описаны в таблице 43.

Таблица 43 – Методы модуля компиляции задания

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Описание** |
| 1 | def \_\_compare\_answers (user\_output: str, expected\_output: str) -> bool | Метод сравнения ответа, отправленного пользователем, и верного ответа, хранящегося в базе данных |
| 2 | def \_\_get\_api\_response( self) -> t.Optional[t.Any] | Метод, формирующий запрос компиляции кода к онлайн-компилятору Wandbox и возвращающий файл JSON с результатами компиляции |
| 3 | def \_\_process\_response (self, response: t.Any) -> None | Метод, разделяющий ошибки, найденные при компиляции кода, на два вида исключения, реализованных в веб-приложении |
| 4 | def parse\_api\_call (call: ApiCall) -> ParsedApiResponse | Метод обрабатывающий полученный результат компиляции для дальнейшего предоставления результата проверки пользователю в веб-интерфейсе системы |

## **Особенности реализации, эксплуатации и сопровождения системы**

Система реализована на языке программирования Python и это значит, что её можно использовать на любом веб-сервере Apache.

Основной сложностью являлась реализации по функциям, выполняемым с компиляцией кода, написанным пользователем. Правильное добавление и редактирование глав и заданий отобразится на удобстве пользования функционалом потенциальным как обучающимся, так и преподавателем.

Если будет необходимо, то можно расширять функционал системы до нужного уровня и корректировать и исправлять недочёты в текущей работе системы. База внедрения продукта – МБОУ СШ №82 г. Ульяновска.

## **Руководство пользователя**

### **Требования к условиям эксплуатации**

Основные системные требования системы определяются требованиями клиентской части, которые в свою очередь определяются требованиями браузера. Минимальные системные требования приведены в таблице 46.

Таблица 44 – Минимальные системные требования

|  |  |  |
| --- | --- | --- |
| **Подключение к интернету** | | Wi-Fi, проводное подключение |
| **Характеристики компьютера** | Процессор | Intel Core i3 |
| Жесткий диск | 1 Gb |
| Объём оперативной памяти | 2 Gb |
| **ОС** | | Windows 7 и выше |
| **ПО** | | Браузер Yandex Browser |

Для работы серверной части потребуется:

* бесперебойная подача питания;
* защита от несанкционированного воздействия;
* доступ к сети Интернет.

### **Инсталляция и настройка**

Для работы клиентской части необходим браузер. Для запуска сервера сначала необходимо установить Python. После чего в папке сервера выполнить *pip install -r requirements.txt* для установки всех необходимых пакетов. Затем необходимо выполнить команду *python app.py* для запуска сервера. После успешного запуска сервера в консоль выведется соответствующее сообщение, которое представлено на рисунке 8.

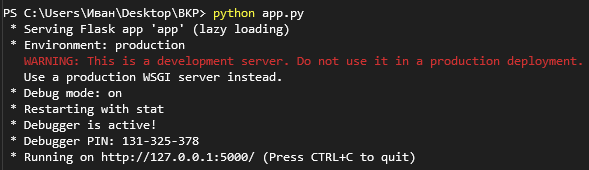


Рисунок 8 – Сообщение после запуска сервера

### **Порядок и особенности работы**

После того, как пользователь ввел адрес веб-сервиса в браузере, он попадает на главную страницу, которая является стартовой страницей сайта. Она представлена на рисунке 9.

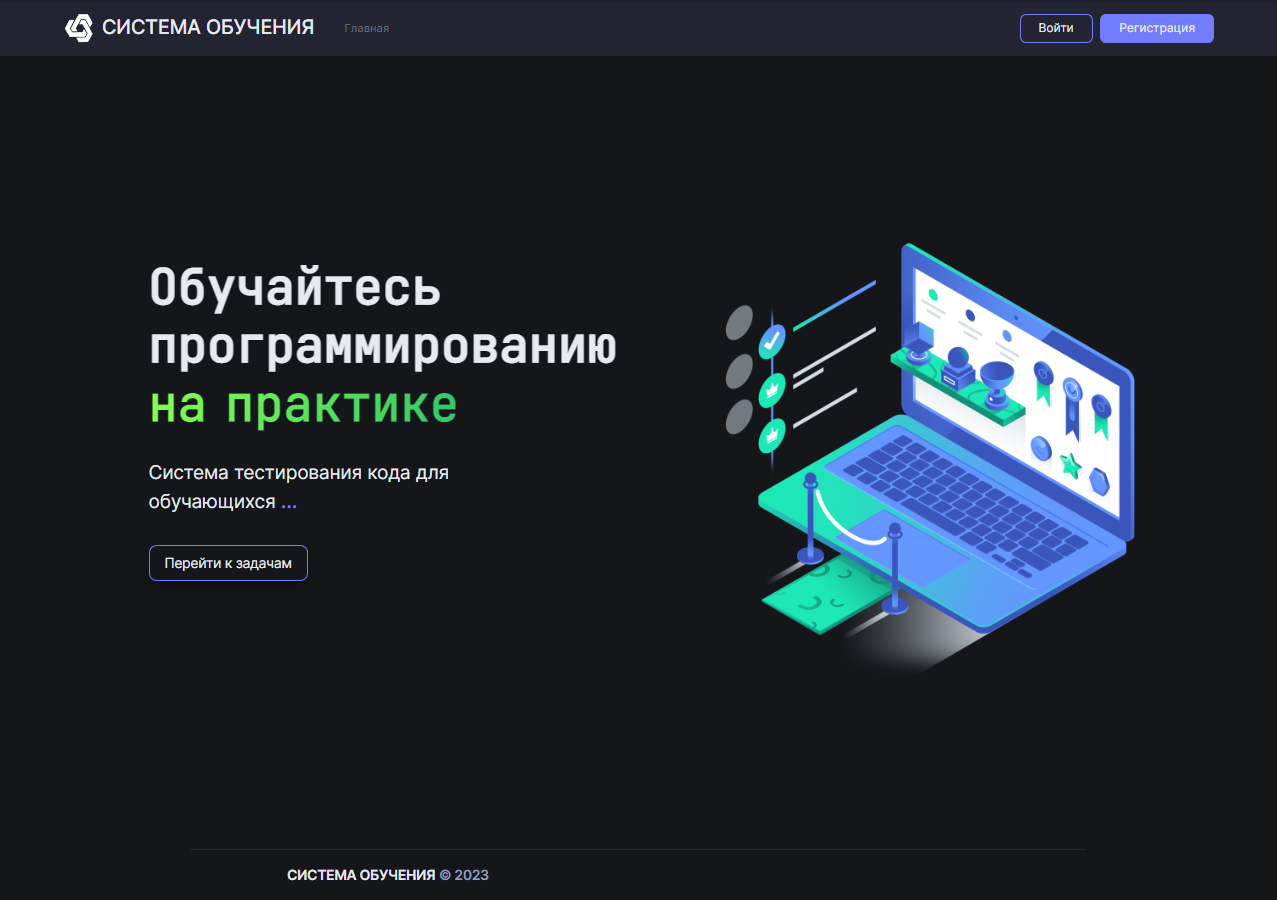


Рисунок 9 – Главная страница сайта

Главная страница состоит из блока, который содержит в себе ссылку на добавленные темы для обучения. В правом верхнем углу есть две кнопки – «Вход» и «Регистрация». При нажатии на «Вход» пользователь попадает на страницу авторизации, которая представлена на рисунке 10.

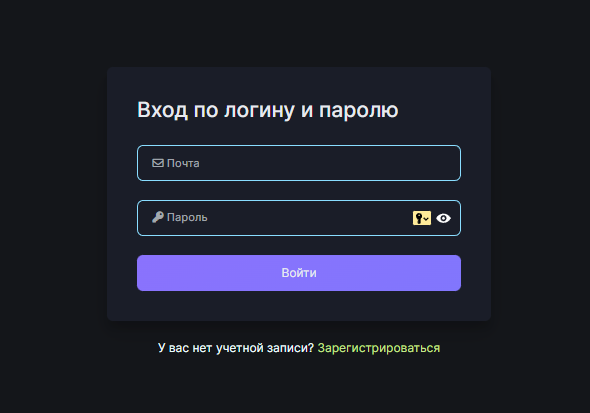


Рисунок 10 – Форма авторизации пользователя

Если пользователь уже имеет аккаунт в системе, то он может войти в систему под ранее указанными логином и паролем. Если же пользователь впервые в системе, то он может зарегистрироваться, нажав на «Регистрация» на главной странице. После нажатия, он попадает на форму регистрации, которая представлена на рисунке 11.

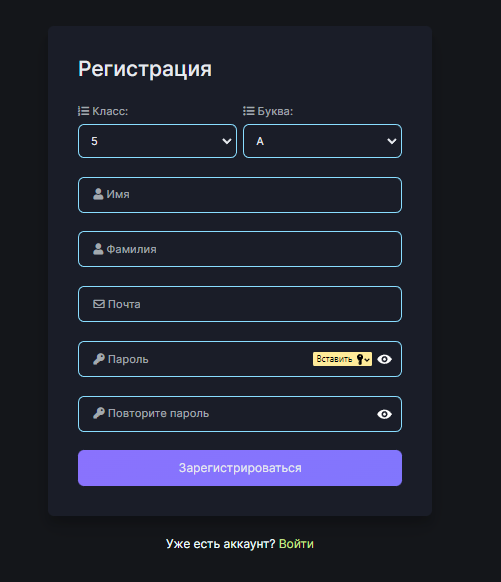


Рисунок 11 – Форма регистрации пользователя

После успешного входа/регистрации пользователь попадает на свою личную страницу, а кнопка «Вход» меняется на указанный пользователем логин. Личная страница представлена на рисунке 12.

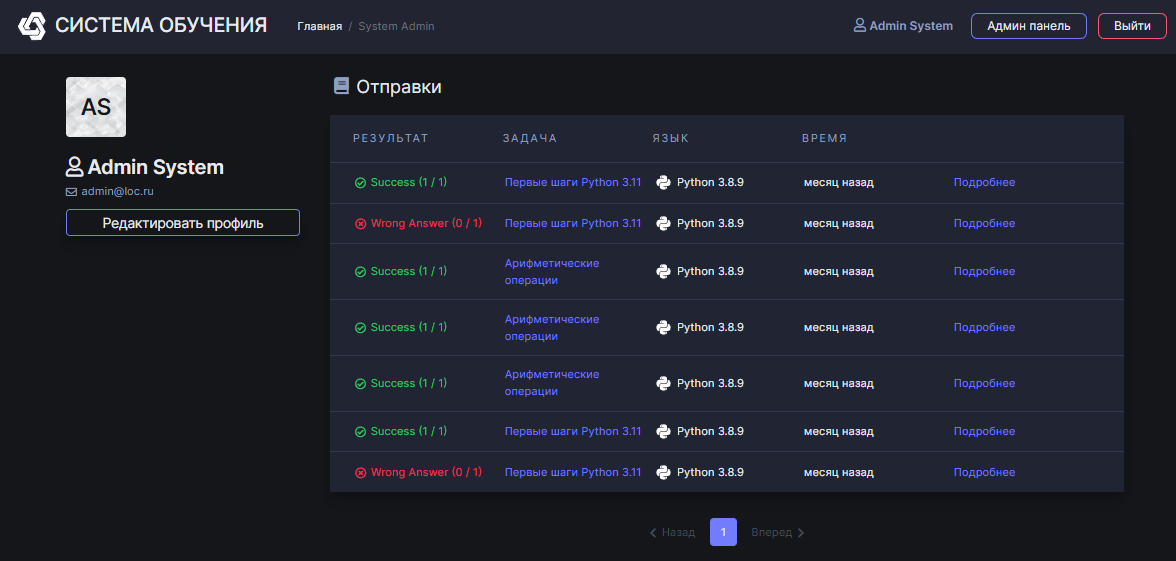


Рисунок 12 – Личная страница пользователя

Нажав на заголовок приложения «Система обучения», пользователь может вернуться на главную. Перейдя по ссылке «Перейти к задачам», перед пользователем появляется содержимое класса, в котором он находится, и доступные темы для выполнения. Пример представлен на рисунке 13.

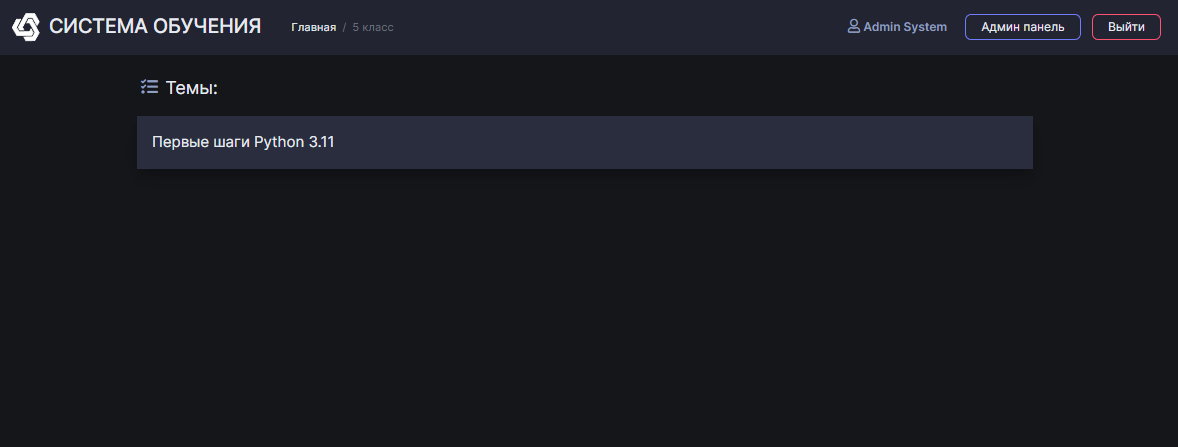


Рисунок 13 – Глава учебного материала

В конце каждой главы подводится краткий итог по главе. Выполняя задание, код выполняется прямо в браузере. На рисунке 14 приведен пример выполнения написанной пользователем функции. В текстовом редакторе написан валидный Python код, который выводит «Hello world», предоставлен на рисунке 15. Результатом выполнения является вывод состояния компилирования в браузере, которое содержит состояние компилирования.

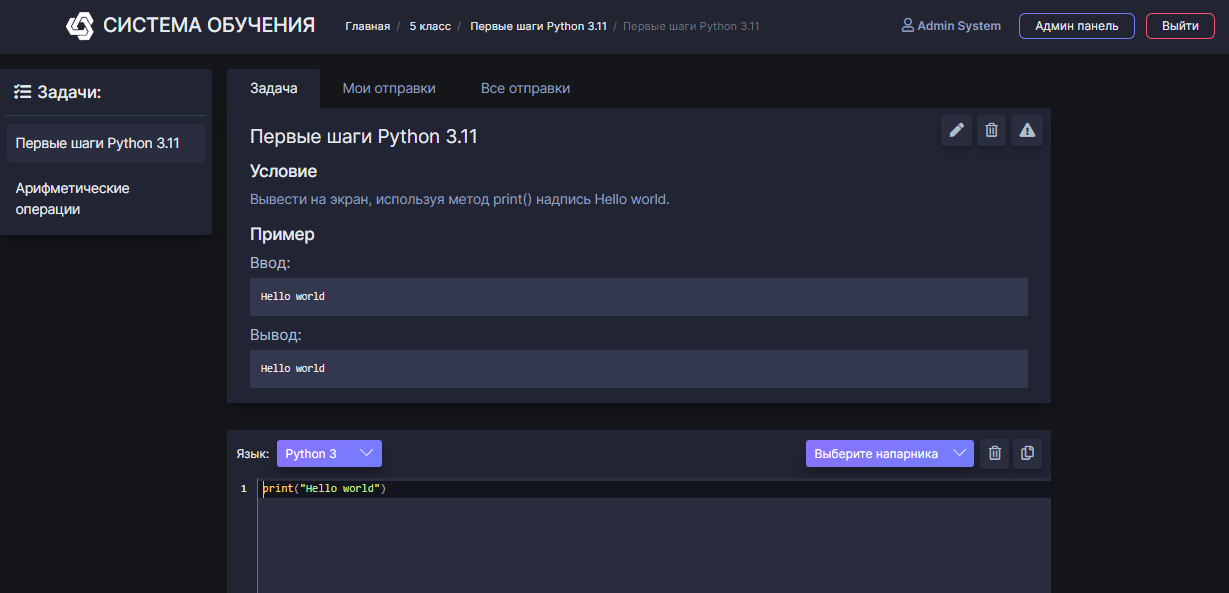


Рисунок 14 – Задание из главы «Первые шаги Python 3.11»



Рисунок 15 – Выполнение задания в браузере

Прогресс выполнения заданий можно отслеживать в своем личном кабинете. Выполнив все задания главы, она подкрашивается в зеленым. Также личный кабинет содержит все решения пользователя. Пример представлен на рисунке 16.

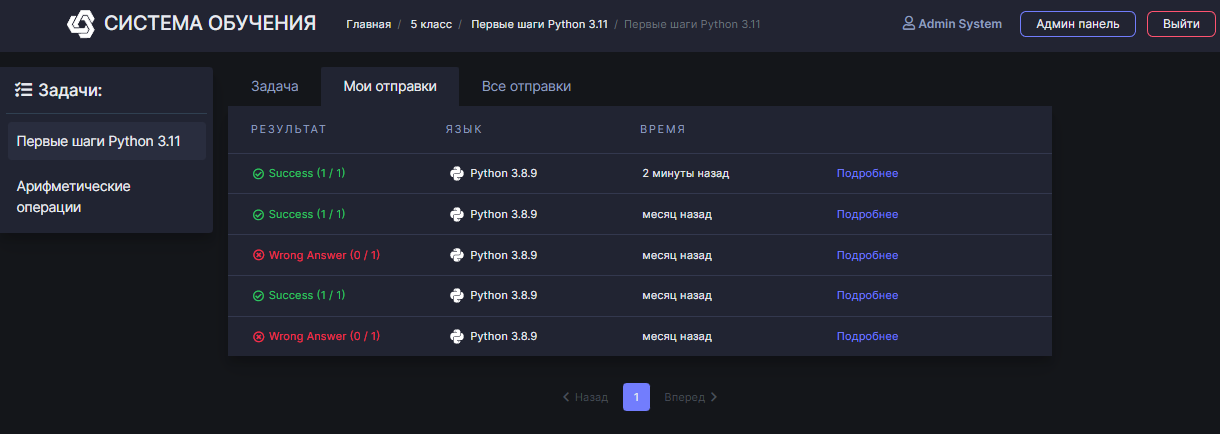


Рисунок 16 – Прогресс пользователя и его выполненные задания

Если пользователь является администратором, то у него в правом верхнем углу доступна также кнопка «Админ панель», по которой он может зайти на страницу администрирования. Которая представлена на рисунке 17.

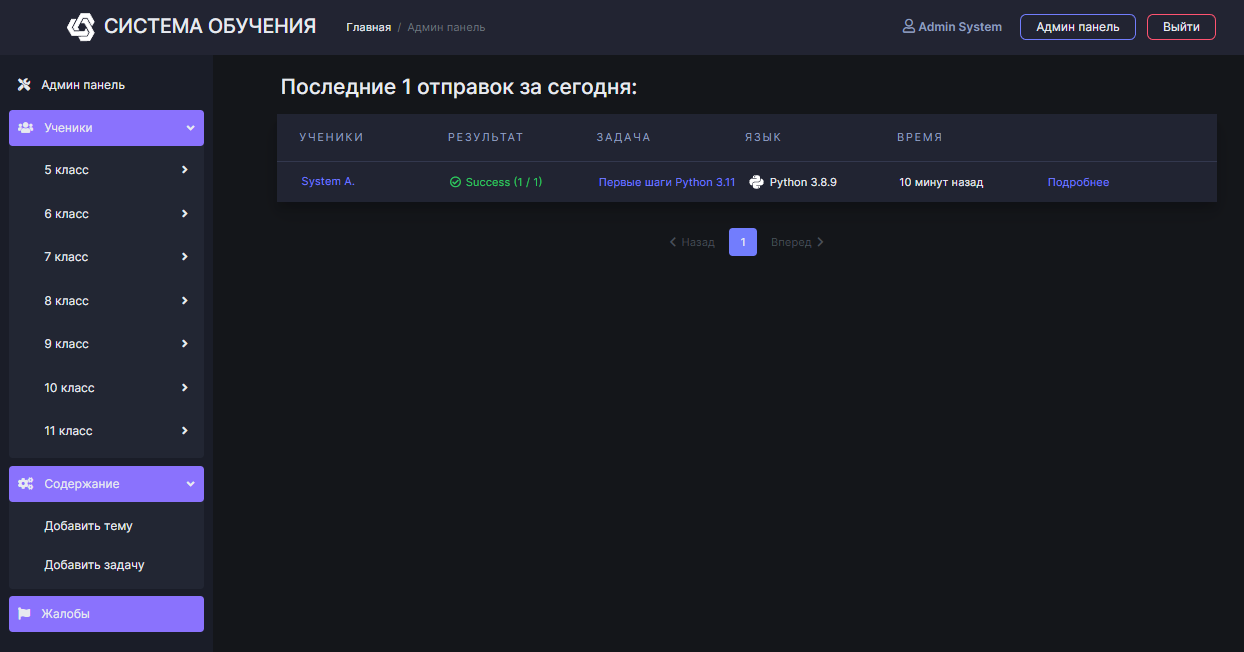


Рисунок 17 – Страница администратора

На ней он может как редактировать текущие главы и задания, так и добавлять новые. В блоке «Содержание» администратор может добавить блок содержания, главу к теме или же задание к главе. Для того, чтобы добавить тему, необходимо ввести только название, после чего нажать кнопку добавить. Чтобы добавить главу, необходимо заполнить два поля – название, id класса. Глава с введённым названием добавится к тому классу, который был указан. Для добавления задачи к главе необходимо выбрать ее из выпадающего списка.

Чтобы отредактировать текущее содержание, нужно в блоке «Содержание» выбрать одну из вкладок и выбрать необходимую тему, главу или задачу для редактирования. Выбрав, например, главу ее можно либо удалить, либо изменить ее содержимое. Пример изменения главы представлен на рисунке 18 и 19.

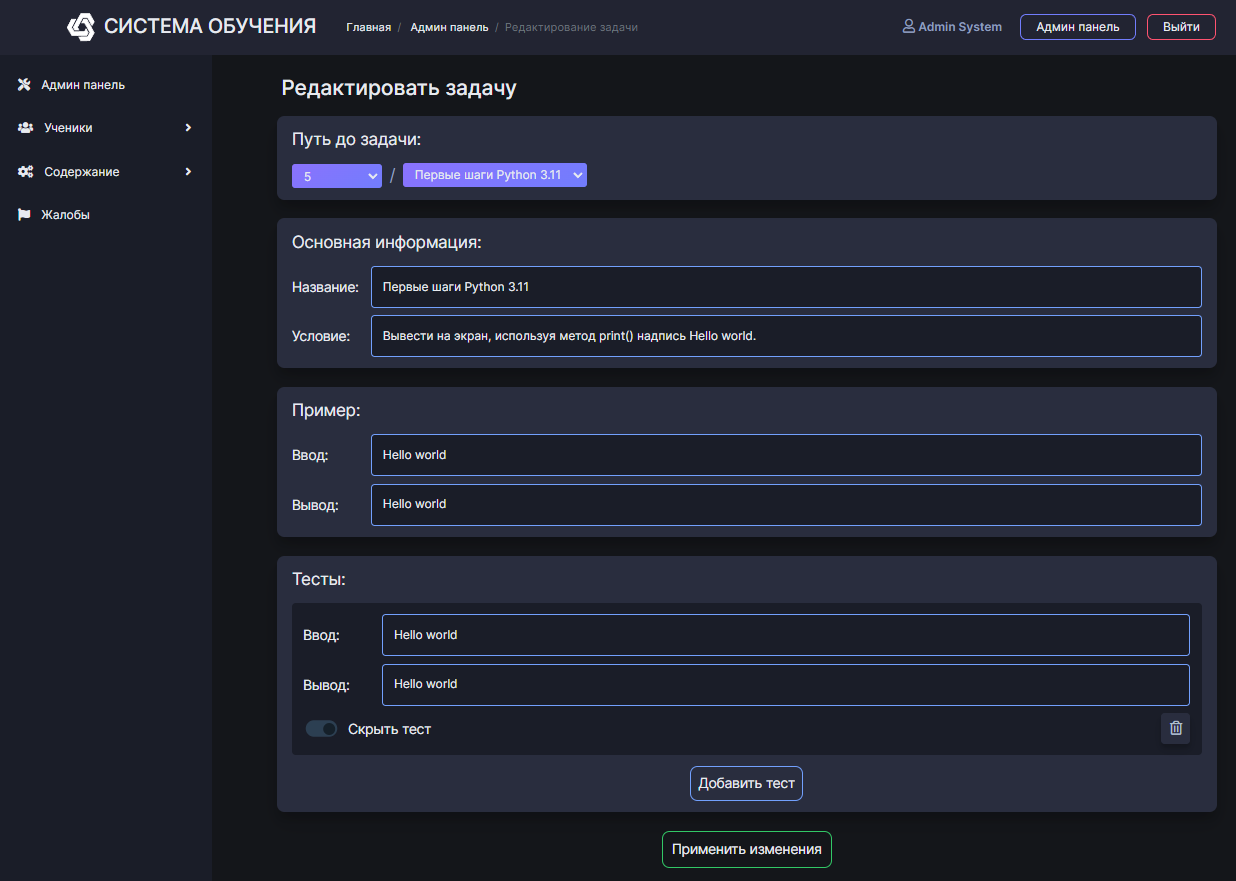


Рисунок 18 – Редактирование задания

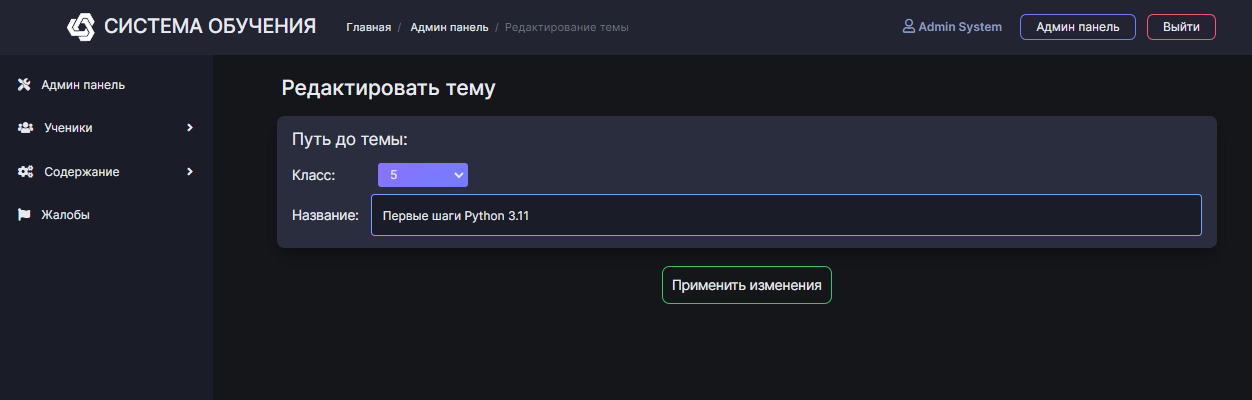


Рисунок 19 – Редактирование темы

### **Исключительные ситуации и их обработка**

При возникновении исключительных ситуации, которые связаны с работой сервера или БД в системе всплывает сообщение об ошибке в правом нижнем углу.

Например, при авторизации пользователя, который не авторизован, выводится следующая ошибка, которая представлена на рисунке 20 и 21.

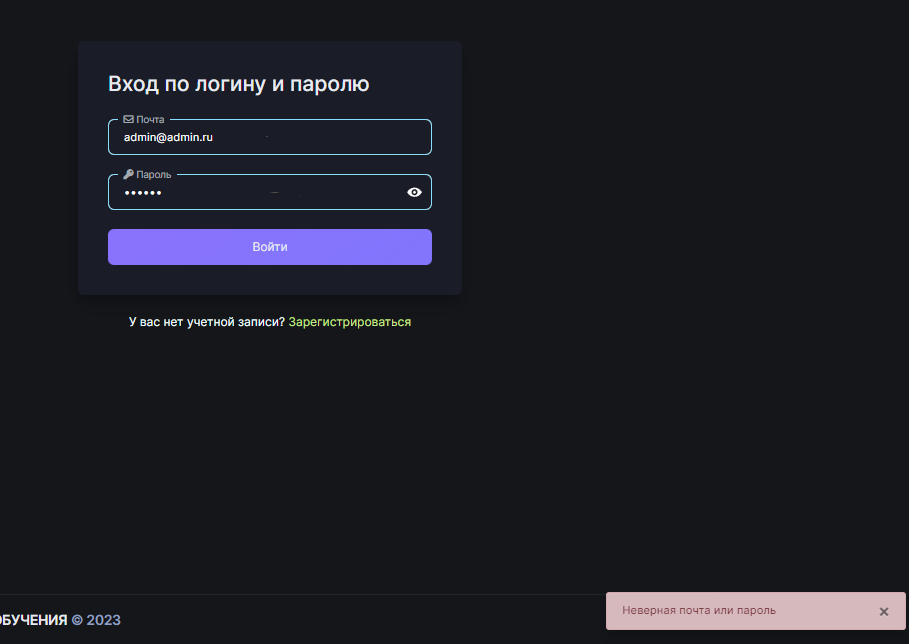


Рисунок 20 – Ошибка авторизации

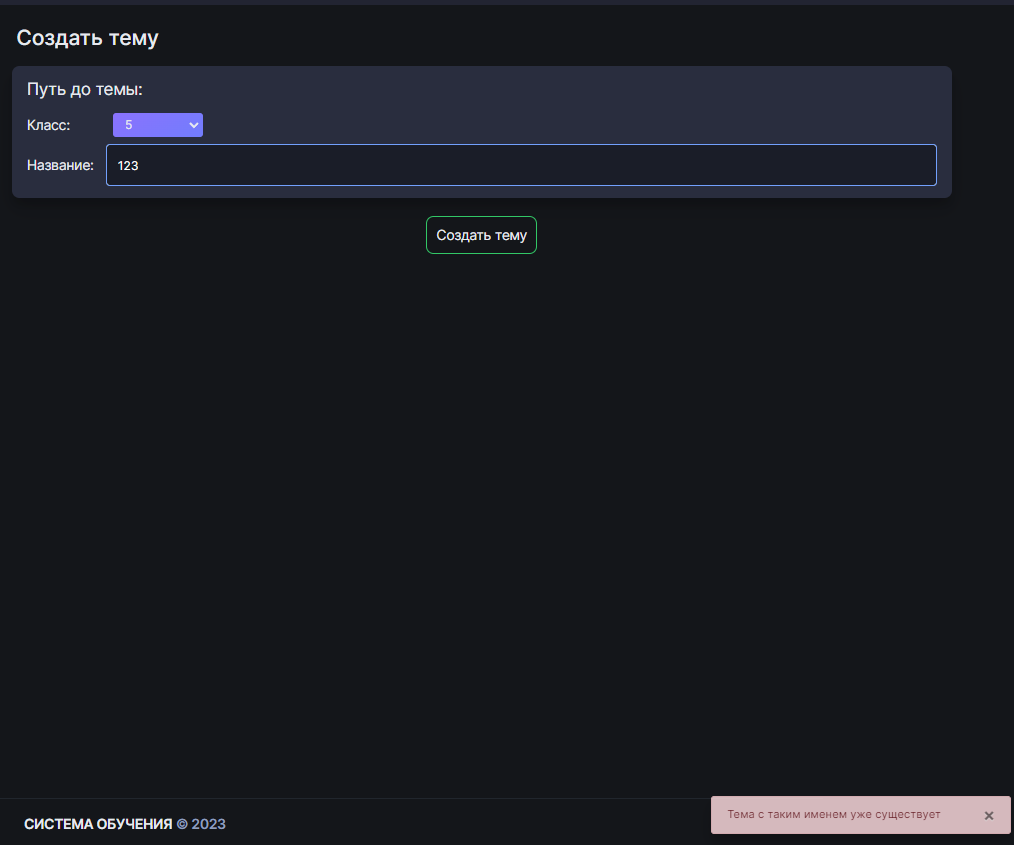


Рисунок 21 – Ошибка создания темы (или же главы)

При ошибках на стороне клиента, когда он выполняет код в редакторе, ошибки выводятся рядом с текстовым редактором. Данные ошибки являются ошибками, которые представлены в языке Python, Pascal ABC, C#. Поэтому все возможные ошибки можно посмотреть в документации Python. На рисунке 22 представлена одна из них.

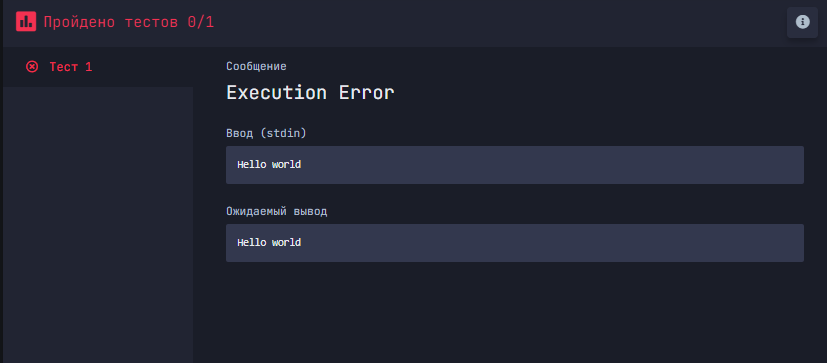


Рисунок 22 – Ошибка выполнения кода

# ТЕСТИРОВАНИЕ СИСТЕМЫ

## **Условия и порядок тестирования**

Объектом тестирования является прикладное программное обеспечение – облачная информационная система, предназначенная для автоматизации обучением программированию в средней школе и повышением эффективности получением новых навыков у учеников.

Тип тестироваиия: ручной.

Уровень тестирования: приёмочное тестирование. В этом случае тестирование выполняется на основе набора типовых случаев и сценариев, разработанных на основании требований к данному приложению.

При тестировании данной системы было принято использовать метод «модель чёрного ящика».

Порядок тестирования:

* снятие входных данных;
* снятие выходных данных.

## **Исходные данные для контрольных примеров**

### **Страница авторизации**

Пользователь вводит логин и пароль, нажимает «Войти».

Возможные варианты развития событий:

* + - ошибка «Неверная почта или пароль» (рисунок 20);
    - данные корректны, и пользователь проходит аутентификацию   
      (рисунок 12).

### **Страница главы**

Пользователь переходит на страницу с доступными ему главами.

Возможные варианты развития событий:

* + - страница отобразилась с данными корректно (рисунок 13);
    - страница отобразилась без данных (в случае потери связи с базой данных).

### **Страница изменения, удаления, просмотра заданий**

Пользователь переходит по ссылке на страницу.

Возможные варианты развития событий:

* + - страница отображается в штатном режиме с нужными данными (рисунок 18);
    - страница ошибки в случае возможного неверного запроса на сервер (рисунок 21).

### **Страница выполнения задания**

Пользователь переходит на страницу выполнения задания.

Возможные варианты развития событий:

* + - код вернул правильный вывод, который совпадает с эталонными значениями (рисунок 15);
    - код вернул ошибочный вывод, и на странице отображается сообщение об ошибке (рисунок 22).

### **Страница с отчетами об отправках**

Пользователь переходит на страницу выполнения задания.

Возможные варианты развития событий:

* вывод решений пользователя на странице пользователя, когда есть выполненные задания (рисунок 16);
* отсуствует вывод решений, когда есть выполненные задания у пользователя.

### **Страница обработки жалобы**

Администратор переходит на страницу обработки жалобы.

Возможные варианты развития событий:

* + - * жалобы корректно отображаются в виде списка для работы с ними;
      * жалоба была отправлена пользователем, но не отображается в личном кабинете администратора системы.

## **Результаты тестирования**

Результаты тестирования показали, что основной функционал системы реализован и работает исправно. Проверялось правильное отображение всех окон. Все ошибки обрабатываются исправно, кроме отказов системного программного обеспечения.

Данная система соотвествует предъявляемым ей функциональным требованиям.

# ЭКОНОМИЧЕСКИЙ РАЗДЕЛ

Для оценки эффективности разработанной системы был проведен экономический анализ стоимости владения системой, а также воздействие системы на бизнес архитектуру. Источник формул для расчета затрат [10].

## **Оценка совокупной стоимости владения**

Затраты на разработку программного продукта рассчитываются по следующей формуле:

где – общий фонд оплаты труда разработчиков ПП;

– начисления на заработную плату разработчиков ПП во внебюджетные фонды;

– затраты, связанные с эксплуатацией техники;

– затраты на специальные программные продукты, необходимые для разработки ПП;

– затраты на хозяйственно-операционные нужды (бумага, литература, носители информации и т. п.);

– накладные расходы ( = 70 % от ).

При разработке программного продукта общее время разработки составило мес.

Соответственно, машинное время (непосредственная работа с вычислительной и оргтехникой) составляет .

Фонд оплаты труда за время работы над программным продуктом:

Сумма начислений на заработную плату во внебюджетные фонды составляет:

Затраты, связанные с использованием вычислительной и оргтехники:

где – коэффициент готовности ЭВМ, = 0.95;

– количество единиц техники, равно 1;

– себестоимость машиночаса.

Стоимость машинного часа эксплуатации ЭВМ показывает среднюю величину затрат организации на один час работы ЭВМ. Она определяется по формуле:

где – стоимость машиночаса эксплуатации ЭВМ, ;

– суммарные годовые затраты, связанные с содержанием и эксплуатацией ЭВМ, включая затраты на содержание сетевого оборудования, оргтехники и т. д., ;

– годовой эффективный фонд времени работы одной ЭВМ, ;

– коэффициент готовности;

– количество ЭВМ, .

Суммарные затраты за год, связанные с содержанием и эксплуатацией ЭВМ:

где – годовые затраты на материалы (магнитные носители, расходные материалы, бумага и т. д.), .;

– годовая заработная плата персонала, непосредственно обслуживающего электронно-вычислительную технику, .;

– начисления на заработную плату персонала, непосредственно обслуживающего технику, в социальные внебюджетные фонды, .;

– амортизационные отчисления с оборудования, входящего в электронно-вычислительный комплекс, .;

– износ программных продуктов общего назначения, .;

– годовые расходы на содержание и эксплуатацию оборудования, .;

– прочие накладные расходы, .

где – коэффициент материальных затрат;

– балансовая стоимость оборудования, .

Годовая заработная плата персонала, непосредственно обслуживающего вычислительную технику:

где – оклад персонала, непосредственно обслуживающего вычислительную технику (техника, системного администратора и т. д.);

– коэффициент дополнительной заработной платы.

Начисления на заработную плату во внебюджетные фонды:

где – коэффициент отчислений во внебюджетные фонды.

Амортизационные отчисления составляют:

где – норма амортизационных отчислений для компьютерной техники . Затраты на хозяйственно-организационные нужды представлены   
в таблице 45.

Таблица 45 – Затраты на хозяйственно-организационные нужды

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование** | **Цена за единицу (руб.)** | **Кол-во (шт.)** | **Всего (руб.)** |
| Бумага | 250 | 1 | 250 |
| Итого |  |  | 250 |

Износ программных продуктов общего назначения:

где – стоимость инструментальных программных продуктов, .;

– норма амортизационных отчислений для программных продуктов .

Годовые расходы на содержание и эксплуатацию оборудования рассчитываются по формуле:

где – годовые затраты на ремонт и обслуживание техники, .;

– годовые затраты на электроэнергию, .

где – коэффициент затрат на текущий ремонт оборудования.

где – мощность ρ-го оборудования, входящего в электронно-вычислительный комплекс, кВт;

– цена электроэнергии, руб./час.

Годовая величина прочих накладных расходов рассчитывается по формуле:

где – коэффициент прочих накладных расходов.

Общие годовые затраты составят:

В результате стоимость машинного часа эксплуатации ЭВМ:

– машинное время работы над программным продуктом, равно

Накладные расходы:

Общие затраты на разработку программного продукта составят:

Таким образом, затраты на разработку программного продукта составили:

## **Анализ качественных и количественных факторов воздействия проекта на бизнес-архитектуру организации**

На примере МБОУ СШ №82 внедрение разработанной системы в учебный процесс позволит увеличить эффективность работы за счет сокращения временных затрат на проведении учебных занятий.

Разработанная система не приведет к реорганизации штата или увольнению сотрудников.

Для того, чтобы определить на сколько в процентном соотношении произойдет сокращение временных затрат на проведение учебных занятий, было произведен замер показателей T2 и T1, которые обозначают время предоставления материала на занятии.

Время предоставления материала по лекции сократится на 37.5% (расчет производился по формуле 1).

где – процентное соотношение времени ;

– время, которое предполагается тратить на проведение занятия после внедрения системы;

– время, которое тратится в настоящий момент на проведение занятия.

При обработке данных: ., ,

Таким образом, главным эффектом от внедрения программного обеспечения станет снижение временных затрат на проведение занятия на .

# ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе была реализована система, которая позволяет облегчить процесс обучения для учителей информатика в   
МБОУ СШ №82 г. Ульяновска, а также повысить интерес к программированию у учеников, используя современные подходы.

Добавление интерактивности в процесс обучения программированию – непростой процесс, так как IT сфера постоянно улучшается, с каждым днем появляются новые ньансы в каждом из современных языков. Система позволяет подстраивать содержание курсов, что позволяет постоянно изменять среду. Она существенно сокращает затраченное время учителя в проведении практических, самостоятельных и проверочных работ, помимо этого в системе предусмотрена возможность автоматической проверки решений учеников.

Данная система была полностью реализована согласно техническому заданию, но для того, чтобы сделать ее еще лучше более удобной как для учителей, так и для учеников, в неё могут быть добавлены новые возможности. Система прошла процесс внедрения в средней школе №82 в г. Ульяновске. В тестировании системы приняли участие ученики 7-11 классов, нагрузку в 20-25 человек среда программирования выдерживает, что является оптимальным для проведения уроков информатики. Планируется дальнейшее развитие и поддержка системы.

Предполагается, что по требованию учителей-предметников в учебных заведених доступ к данной системе будет выдаваться безвозмездно. Площадки смогут без ограничений смогут развернуть её в своей среде, что позволит им автоматизировать процесс обучения программированию.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Flask documentation : сайт. – URL: <https://docs.flask.com/en/> (дата обращения: 01.06.2023)
2. ITVDN – Видеокурсы по программированию : сайт. – URL: <https://itvdn.com/ru> дата обращения: 01.06.2023)
3. jQuery API Documentation : сайт. – URL: <https://api.jquery.com/> (дата обращения: 01.06.2023)
4. Python.org – Современный учебник Python 3.10 : сайт. – URL: <https://learn.python.org/> (дата обращения: 01.06.2023)
5. SQLite Documentation : сайт. – URL: <https://www.sqlite.ru/docs/> (дата обращения: 01.06.2023)
6. Босова, А. Ю. Информатика / А. Ю. Босова, Л. Л. Босова. – М. : БИНОМ. Лаборатория знаний, 2019. – 245 с.
7. ГОСТ 34.601-90 Информационная технология (ИТ). Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания. : дата введения 01.01.92. – Москва : Стандартинформ, 2009. – 5 с.
8. ГОСТ 34.603-92 Информационная технология. Виды испытаний автоматизированных систем : дата введения 1993-01-01. – М. : Стандартимформ, 2009. – 5 с.
9. Мамедли, Р.Э. Системы управления базами данных / Р.Э. Мамедли. – Нижневартовск : Нижневартовский государственный университет, 2021. – 214 с. – ISBN 978-5-00047-585-0.
10. Оценка трудоемкости разработки программного продукта: метод указания / сост. Шанченко Н.И. – Ульяновск : УлГТУ, 2015. – 21с.
11. Поляков, К. Ю. Информатика / К. Ю Поляков. – Лаборатория знаний. – М : БИНОМ, 2020. – 288 с.
12. Преимущества языка Python : сайт. – URL: <https://metanit.com/python/> (дата обращения: 01.06.2023)
13. Приказ «Об утверждении федерального государственного образовательного стандарта основного общего образования» Минобрнауки России от 17.12.2010 N 1897 (ред. от 11.12.2020).
14. Приказ Министерства просвещения РФ от 31 мая 2021 г. № 287 «Об утверждении федерального государственного образовательного стандарта основного общего образования».
15. РД 50-34.698-90 «Автоматизированные системы. Требования к содержанию документов».
16. Родионов, В.В. Выполнение выпускной квалификационной работы бакалавра : учебно-методическое пособие для студентов направления 09.03.02 «Информационные системы и технологии» / Родионов В.В. – Ульяновск: УлГТУ, 2017. – 74 с. – Доступен также в Интернете. – ISBN 978-5-9795-1630-1. – URL: <http://lib.ulstu.ru/venec/disk/2017/14.pdf>
17. Рудаков, А. В. Технология разработки программных продуктов. Практикум: учебное пособие для студ. учреждений сред. проф. образования / А. В. Рудаков. – М. : Издательский центр «Академия», 2014. – 149 с.
18. Создание таблиц в базах данных SQLite // ZametkiNaPolyah.ru : сайт. – URL: <https://zametkinapolyah.ru/zametki-o-mysql/chast-10-1-sozdanie-tablic-v-bazax-dannyx-sqlite.html> (дата обращения: 17.05.2023)
19. [Федеральный закон от 29.12.2012 N 273-ФЗ (ред. от 29.12.2022) «Об образовании в Российской Федерации» (с изм. и доп., вступ. в силу с 11.01.2023)](https://www.consultant.ru/document/cons_doc_LAW_140174/).
20. Форсье, Дж. Django. Разработка веб-приложений на Python. / Дж. Форсье, П. Биссекс, У. Чан. – СПб.: Символ-Плюс, 2010. – 456 с.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Исходные тексты программы**

***app.py***

**from** app **import** create\_app

app **=** create\_app**()**

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

app**.**run**(**port**=**5000**,** host**=**'127.0.0.1'**,** debug**=**app**.**config**[**'DEBUG'**])**

***config.py***

**from** os **import** environ**,** path

**from** dotenv **import** load\_dotenv

basedir **=** path**.**abspath**(**path**.**dirname**(**\_\_file\_\_**))** # Getting base directory

load\_dotenv**(**path**.**join**(**basedir**,** '.env'**))** # Loading env

# Base Config

**class** **Config(object):**

SECRET\_KEY **=** environ**.**get**(**'SECRET\_KEY'**)**

STATIC\_FOLDER **=** 'static'

TEMPLATES\_FOLDER **=** 'templates'

SQLALCHEMY\_DATABASE\_URI **=** 'sqlite:///' **+** path**.**join**(**basedir**,** 'app'**,** 'db'**,** 'app.db'**)**

SQLALCHEMY\_TRACK\_MODIFICATIONS **=** **False**

MAIL\_SERVER **=** 'smtp.mail.ru'

MAIL\_USERNAME **=** environ**.**get**(**'MAIL\_USERNAME'**)**

MAIL\_DEFAULT\_SENDER **=** environ**.**get**(**'MAIL\_USERNAME'**)**

MAIL\_PASSWORD **=** environ**.**get**(**'MAIL\_PASSWORD'**)**

MAIL\_PORT **=** 465

MAIL\_USE\_SSL **=** **True**

MAIL\_USE\_TLS **=** **False**

RECORDS\_PER\_PAGE **=** 10

# Production Config

**class** **ProdConfig(**Config**):**

FLASK\_ENV **=** 'production'

DEBUG **=** **False**

TESTING **=** **False**

# Developing Config

class DevConfig(Config):

FLASK\_ENV = 'development'

DEBUG = True

TESTING = True

config = {

'dev': DevConfig,

'prod': ProdConfig,

'default': DevConfig,

}

***app/forms.py***

**from** flask\_wtf **import** FlaskForm

**from** wtforms **import** StringField**,** SelectField**,** BooleanField**,** SubmitField

**from** wtforms**.**validators **import** DataRequired

**class** **EditProfileForm(**FlaskForm**):**

surname **=** StringField**(**'Имя'**,** validators**=[**DataRequired**()])**

name **=** StringField**(**'Фамилия'**,** validators**=[**DataRequired**()])**

grade\_id **=** SelectField**(**'Класс'**,** choices**=[],** validators**=[**DataRequired**()])**

grade\_letter **=** SelectField**(**'Буква'**,** choices**=[**'А'**,** 'Б'**,** 'В'**,** 'Г'**],** validators**=[**DataRequired**()])**

is\_admin **=** BooleanField**(**'Права администратора'**,** default**=False)**

submit **=** SubmitField**(**'Отправить'**)**

***app/models.py***

**import** typing **as** t

**import** datetime

**from** os **import** environ

**import** sqlalchemy

**from** sqlalchemy**.**orm **import** relationship

**from** sqlalchemy**.**ext**.**hybrid **import** hybrid\_property**,** hybrid\_method

**from** werkzeug**.**security **import** generate\_password\_hash**,** check\_password\_hash

**from** flask **import** url\_for

**from** flask\_login **import** UserMixin

**from** transliterate **import** slugify

**from** app **import** db**,** login\_manager

user\_submission **=** db**.**Table**(**

'user\_submission'**,**

db**.**Column**(**'user\_id'**,** db**.**Integer**,** db**.**ForeignKey**(**'users.id'**)),**

db**.**Column**(**'submission\_id'**,** db**.**Integer**,** db**.**ForeignKey**(**'submissions.id'**))**

**)**

**class** **BaseModel(**db**.**Model**):**

\_\_abstract\_\_ **=** **True**

**def** \_\_repr\_\_**(**self**)** **->** **str:**

**return** self**.**\_repr**(id=**self**.id)**

**def** \_repr**(**self**,** **\*\***fields**:** t**.**Dict**[str,** t**.**Any**])** **->** **str:**

field\_strings **=** **[]**

at\_least\_one\_attached\_attribute **=** **False**

**for** key**,** field **in** fields**.**items**():**

**try:**

field\_strings**.**append**(**f'{key}={field!r}'**)**

**except** sqlalchemy**.**orm**.**exc**.**DetachedInstanceError**:**

field\_strings**.**append**(**f'{key}=DetachedInstanceError'**)**

**except** **AttributeError:**

field\_strings**.**append**(**f'{key}=Undefined'**)**

**else:**

at\_least\_one\_attached\_attribute **=** **True**

**if** at\_least\_one\_attached\_attribute**:**

**return** f"<{self**.**\_\_class\_\_**.**\_\_name\_\_}({','**.**join**(**field\_strings**)**})>"

**return** f"<{self**.**\_\_class\_\_**.**\_\_name\_\_} {**id(**self**)**}>"

**class** **User(**BaseModel**,** UserMixin**):** # lgtm [py/missing-equals]

\_\_tablename\_\_ **=** "users"

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

name **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**,** nullable**=False)**

surname **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**,** nullable**=False)**

email **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**,** index**=True,** unique**=True)**

verified **=** sqlalchemy**.**Column**(**sqlalchemy**.**Boolean**,** default**=False)**

role\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"roles.id"**))**

role **=** relationship**(**'Role'**,** lazy**=**'joined'**,** uselist**=False)**

grade\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"grades.id"**))**

grade **=** relationship**(**'Grade'**)**

grade\_letter **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

hashed\_password **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

registration\_date **=** sqlalchemy**.**Column**(**sqlalchemy**.**DateTime**,** default**=**datetime**.**datetime**.**utcnow**)**

submissions **=** db**.**relationship**(**

'Submission'**,**

secondary**=**user\_submission**,**

backref**=**db**.**backref**(**'users'**,** lazy**=**'joined'**),**

lazy**=**'dynamic'**,**

order\_by**=**"desc(Submission.submission\_date)"**,**

cascade**=**'all,delete'

**)**

reports **=** db**.**relationship**(**

'Report'**,**

backref**=**db**.**backref**(**'user'**,** lazy**=**'joined'**),**

lazy**=**'dynamic'**,**

cascade**=**'all,delete'

**)**

**def** set\_password**(**self**,** password**):**

self**.**hashed\_password **=** generate\_password\_hash**(**password**)**

**def** check\_password**(**self**,** password**):**

**return** check\_password\_hash**(**self**.**hashed\_password**,** password**)**

*@hybrid\_property*

**def** is\_admin**(**self**)** **->** **bool:**

**if** self**.**role**.**name **==** 'admin'**:**

**return** **True**

**return** **False**

*@hybrid\_property*

**def** classmates**(**self**)** **->** **list:**

**return** db**.**session**.**query**(**User**).filter(**

User**.**grade\_id **==** self**.**grade\_id**,**

User**.**grade\_letter **==** self**.**grade\_letter**,**

User**.id** **!=** self**.id**

**).all()**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**

**id=**self**.id,**

name**=**f'{self**.**name} {self**.**surname}'**,**

grade**=**f'{self**.**grade}{self**.**grade\_letter}'**,**

email**=**self**.**email**,**

role**=**self**.**role**,**

**)**

**class** **Grade(**BaseModel**):**

\_\_tablename\_\_ **=** "grades"

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

number **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** nullable**=True)**

topics **=** relationship**(**'Topic'**,** backref**=**db**.**backref**(**'grade'**,** lazy**=**'joined'**),** lazy**=**'dynamic'**)**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**number**=**self**.**number**)**

**class** **Role(**BaseModel**):**

\_\_tablename\_\_ **=** "roles"

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

name **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**,** nullable**=True)**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**name**=**self**.**name**)**

**class** **Topic(**BaseModel**):**

\_\_tablename\_\_ **=** "topics"

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

grade\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"grades.id"**))**

name **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

translit\_name **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

tasks **=** relationship**(**'Task'**,** backref**=**db**.**backref**(**'topic'**,** lazy**=**'joined'**),** lazy**=**'dynamic'**)**

**def** set\_translit\_name**(**self**):**

self**.**translit\_name **=** slugify**(**self**.**name**)**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**

**id=**self**.id,**

name**=**f'{self**.**name} ({self**.**translit\_name})'**,**

grade**=**self**.**grade**,**

**)**

**class** **Task(**BaseModel**):**

\_\_tablename\_\_ **=** "tasks"

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

topic\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"topics.id"**))**

name **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

translit\_name **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

text **=** sqlalchemy**.**Column**(**sqlalchemy**.**Text**)**

example **=** relationship**(**'Example'**,** uselist**=False,** backref**=**'task'**,** cascade**=**'all,delete'**)**

tests **=** relationship**(**'Test'**,** backref**=**'task'**,** lazy**=**'subquery'**,** cascade**=**'all,delete'**)**

submissions **=** db**.**relationship**(**

'Submission'**,**

backref**=**db**.**backref**(**'task'**,** lazy**=**'joined'**),**

lazy**=**'dynamic'**,**

cascade**=**'all,delete'

**)**

reports **=** relationship**(**'Report'**,** backref**=**'task'**,** cascade**=**'all,delete'**)**

**def** set\_translit\_name**(**self**):**

self**.**translit\_name **=** slugify**(**self**.**name**)**

*@hybrid\_method*

**def** url**(**self**,** tab**:** t**.**Optional**[str]** **=** **None):**

**return** url\_for**(**

'problems.task\_page'**,**

grade\_number**=**self**.**topic**.**grade**.**number**,**

topic\_translit\_name**=**self**.**topic**.**translit\_name**,**

task\_translit\_name**=**self**.**translit\_name**,**

tab**=**tab

**)**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**

**id=**self**.id,**

name**=**f'{self**.**name} ({self**.**translit\_name})'**,**

topic**=**self**.**topic**.**name**,**

grade**=**self**.**topic**.**grade**.**number

**)**

**class** **Example(**BaseModel**):**

\_\_tablename\_\_ **=** "examples"

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

task\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"tasks.id"**))**

example\_input **=** sqlalchemy**.**Column**(**sqlalchemy**.**Text**)**

example\_output **=** sqlalchemy**.**Column**(**sqlalchemy**.**Text**)**

**class** **Test(**BaseModel**):**

\_\_tablename\_\_ **=** "tests"

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

task\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"tasks.id"**))**

stdin **=** sqlalchemy**.**Column**(**sqlalchemy**.**Text**)**

stdout **=** sqlalchemy**.**Column**(**sqlalchemy**.**Text**)**

is\_hidden **=** sqlalchemy**.**Column**(**sqlalchemy**.**Boolean**,** default**=True)**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**

**id=**self**.id,**

task**=**self**.**task**,**

data**={**

'stdin'**:** self**.**stdin**,**

'stdout'**:** self**.**stdout**,**

'hidden'**:** self**.**is\_hidden

**}**

**)**

**class** **Submission(**BaseModel**):**

\_\_tablename\_\_ **=** "submissions"

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

task\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"tasks.id"**))**

language **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

passed\_tests **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**)**

source\_code **=** sqlalchemy**.**Column**(**sqlalchemy**.**Text**)**

submission\_date **=** sqlalchemy**.**Column**(**sqlalchemy**.**DateTime**,** default**=**datetime**.**datetime**.**utcnow**)**

test\_results **=** relationship**(**'TestResult'**,** backref**=**"submission"**,** lazy**=**'subquery'**,** cascade**=**'all,delete'**)**

*@hybrid\_property*

**def** processed\_code**(**self**)** **->** **str:**

# Dictionary with symbols that can break CodeMirror

replacements **=** **{**

'"'**:** r'\"'**,** # double quote

"'"**:** r"\'"**,** # singe quote

"`"**:** r"\`" # backtick

**}**

code **=** **repr(**self**.**source\_code**)[**1**:-**1**]**

**for** character **in** replacements**.**keys**():**

code **=** code**.**replace**(**character**,** replacements**[**character**])**

**return** code

**def** get\_result**(**self**)** **->** **dict:**

failed\_tests **=** **[**result **for** result **in** self**.**test\_results **if** **not** result**.**success**]**

**if** **any(**failed\_tests**):**

failed\_test **=** failed\_tests**[**0**]**

**return** **{**'success'**:** **False,** 'message'**:** failed\_test**.**message**}**

**return** **{**'success'**:** **True,** 'message'**:** 'Success'**}**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**

**id=**self**.id,**

users**=**self**.**users**,**

task**=**self**.**task**,**

result**=**self**.**get\_result**()[**'message'**]**

**)**

**class** **TestResult(**BaseModel**):**

\_\_tablename\_\_ **=** "test\_result"

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

test\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"tests.id"**))**

test **=** relationship**(**'Test'**)**

submission\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"submissions.id"**))**

success **=** sqlalchemy**.**Column**(**sqlalchemy**.**Boolean**)**

message **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

user\_output **=** sqlalchemy**.**Column**(**sqlalchemy**.**String**)**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**

**id=**self**.id,**

test**=**self**.**test**,**

submission**=**self**.**submission**,**

message**=**self**.**message

**)**

**class** **Report(**BaseModel**):**

\_\_tablename\_\_ **=** "reports"

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

user\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"users.id"**))**

task\_id **=** sqlalchemy**.**Column**(**sqlalchemy**.**Integer**,** sqlalchemy**.**ForeignKey**(**"tasks.id"**))**

text **=** sqlalchemy**.**Column**(**sqlalchemy**.**Text**)**

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_repr**(**

user**=**self**.**user**,**

task**=**self**.**task**,**

text**=**self**.**text

**)**

**def** init\_db\_data**():**

# Creating grades

**if** db**.**session**.**query**(**User**).**count**()** **==** 0**:**

**for** grade **in** **range(**5**,** 12**):**

db**.**session**.**add**(**Grade**(**number**=**grade**))**

# Creating roles

user\_role **=** Role**(**name**=**'user'**)**

db**.**session**.**add**(**user\_role**)**

admin\_role **=** Role**(**name**=**'admin'**)**

db**.**session**.**add**(**admin\_role**)**

# Creating admin

admin **=** User**(**

name**=**'Админ'**,**

surname**=**'Админович'**,**

email**=**'contester@mail.ru'**,**

verified**=True,**

role\_id**=**db**.**session**.**query**(**Role**).filter(**Role**.**name **==** 'admin'**).**first**().id,**

grade\_id**=None,**

grade\_letter**=None**

**)**

admin**.**set\_password**(**environ**.**get**(**'ADMIN\_PASSWORD'**))**

db**.**session**.**add**(**admin**)**

***app/routes.py***

from flask import current\_app as app

from flask import render\_template, redirect, url\_for, request, session, abort

from flask\_login import login\_required, current\_user

from flask\_breadcrumbs import register\_breadcrumb

from app import db, login\_manager

from app.models import User, Role, Submission

from app.contester.db\_manager import load\_from\_database

from app.contester.languages import languages

from app.utils.routes import next\_url

from app.utils.forms import init\_grades\_select

from app.forms import EditProfileForm

import app.utils.breadcrumbs as bc

# Unauthorized handler

@login\_manager.unauthorized\_handler

def unauthorized\_callback():

session['next\_url'] = request.path

return redirect(url\_for('auth.login\_page'))

@app.route('/', methods=['GET'])

@register\_breadcrumb(app, '.', 'Главная')

@next\_url

def home\_page():

return render\_template('new\_home.html', title='Главная')

@app.route('/contacts', methods=['GET'])

@register\_breadcrumb(app, '.contacts', 'Контакты')

@next\_url

def contacts\_page():

return render\_template('contacts.html', title='Контакты')

@app.route('/submissions/<int:submission\_id>', methods=['GET'])

@login\_required

def submission\_page(submission\_id):

submission = Submission.query.get\_or\_404(submission\_id)

if submission in current\_user.submissions or current\_user.is\_admin:

context = {

'submission': submission,

'language': languages.get\_language(submission.language, object\_only=True),

'code': submission.processed\_code,

'response': load\_from\_database(submission)

}

return render\_template('submission.html', title=f'Отправленное решение ({submission.task.name})', \*\*context)

abort(404)

@app.route('/profile', defaults={'user\_id': None}, methods=['GET', 'POST'])

@app.route('/user/<int:user\_id>', methods=['GET', 'POST'])

@register\_breadcrumb(app, '.user', '', dynamic\_list\_constructor=bc.view\_user\_dlc)

@login\_required

def profile\_page(user\_id):

# User identification

if user\_id is not None:

user: User = db.session.query(User).get\_or\_404(user\_id)

# Redirecting to /profile if user\_id from /user/<user\_id> is current user's id

if user.id == current\_user.id:

return redirect(url\_for('profile\_page', user\_id=None))

# Raising error 403 if not an admin tries to access someone else's profile

elif not current\_user.is\_admin:

abort(403)

else:

user: User = current\_user

# Form initialization

form = EditProfileForm(obj=user)

init\_grades\_select(form)

# Handling form submission

if form.validate\_on\_submit():

# General info

user.surname = form.surname.data.capitalize()

user.name = form.name.data.capitalize()

user.grade\_id = form.grade\_id.data

user.grade\_letter = form.grade\_letter.data

# Role

if form.is\_admin.data:

role = db.session.query(Role).filter\_by(name='admin').first()

else:

role = db.session.query(Role).filter\_by(name='user').first()

user.role\_id = role.id

db.session.commit()

return redirect(url\_for('profile\_page', user\_id=user.id))

# Forming table data

page = request.args.get('table\_page', type=int, default=1)

table\_data = {

'submissions': user.submissions.paginate(per\_page=app.config['RECORDS\_PER\_PAGE'], page=page, error\_out=False),

'show\_task': True,

}

# Forming page context

context = {

'user': user,

'form': form,

}

return render\_template('profile.html', title=f'{user.surname} {user.name}', \*\*table\_data, \*\*context)

***tester/api\_service.py***

import typing as t

from dataclasses import dataclass, field

import asyncio

from aiohttp import ClientSession

from .exceptions import ContesterError, ApiServiceError, ExecutionError, WrongAnswerError, TimeOutError

API\_URL = 'https://wandbox.org/api/compile.json' # API URL

HEADERS = {'Content-Type': "application/json;charset=UTF-8"} # Request headers

class ApiCallParameters(t.TypedDict):

code: str

compiler: str

stdin: str

class ParsedApiResponse(t.NamedTuple):

success: bool

message: str

user\_output: t.Optional[str]

@dataclass

class ApiCall:

\_\_session: ClientSession

\_\_data: ApiCallParameters

\_\_expected\_output: str

\_\_user\_output: t.Optional[str] = field(init=False, default=None)

@staticmethod

def \_\_compare\_answers(user\_output: str, expected\_output: str) -> bool:

"""Compares user's answer and expected answer"""

if user\_output.strip() == expected\_output.strip():

return True

return False

async def \_\_get\_api\_response(self) -> t.Optional[t.Any]:

"""Requests code compilation in Wandbox and returns JSON with results

Raises

------

ApiServiceError

If can't get valid response from API service

TimeOutError

If the request has exceeded the maximum allowed time

"""

try:

async with self.\_\_session.post(url=API\_URL, headers=HEADERS, json=self.\_\_data, timeout=10) as api\_response:

if api\_response.status == 200:

response\_json = await api\_response.json()

return response\_json

raise ApiServiceError

except asyncio.TimeoutError:

raise TimeOutError

def \_\_process\_response(self, response: t.Any) -> None:

"""Processes API response by checking the response status and comparing the expected output with user's

Raises

------

ExecutionError

If the API service failed to run user's code

WrongAnswerError

If user's output doesn't match expected output

"""

if response['status'] == '0':

self.\_\_user\_output = response['program\_output'].strip()

if self.\_\_compare\_answers(self.\_\_user\_output, self.\_\_expected\_output):

return

raise WrongAnswerError

raise ExecutionError

async def run(self):

"""Sends API call and processes response"""

response = await self.\_\_get\_api\_response()

self.\_\_process\_response(response)

@property

def user\_output(self):

return self.\_\_user\_output

async def parse\_api\_call(call: ApiCall) -> ParsedApiResponse:

try:

await call.run()

except ContesterError as error:

success = False

message = error.message

else:

success = True

message = 'Success'

return ParsedApiResponse(

success=success,

message=message,

user\_output=call.user\_output

)

***tester/db\_manager.py***

import typing as t

from flask\_login import current\_user

from app import db

from app.models import User, Task, Submission, TestResult

from .languages import languages

from .types import ContesterResponse, SingleTestResult

from .utils import get\_number\_of\_passed\_tests

def save\_to\_database(

task: Task,

code: str,

response: ContesterResponse,

language: str,

partner: t.Optional[User] = None

) -> None:

"""Saves all information about user's submission in database"""

submission = Submission(

task\_id=task.id,

language=language,

passed\_tests=response.passed\_tests,

source\_code=code

)

db.session.add(submission)

db.session.commit()

current\_user.submissions.append(submission)

if partner:

partner.submissions.append(submission)

db.session.commit()

for result in response.tests:

test\_result = TestResult(

test\_id=result.test.id,

submission\_id=submission.id,

success=result.success,

message=result.message,

user\_output=result.user\_output

)

db.session.add(test\_result)

db.session.commit()

def load\_from\_database(submission: Submission) -> ContesterResponse:

""" Returns `ContesterResponse` object with all data of submission"""

results\_array = []

for result in submission.test\_results:

results\_array.append(SingleTestResult(

message=result.message,

success=result.success,

user\_output=result.user\_output,

test=result.test

))

return ContesterResponse(

language=languages.get\_language(submission.language, object\_only=True),

tests=results\_array,

passed\_tests=get\_number\_of\_passed\_tests(results\_array),

time='None sec'

)

***tester/exceptions.py***

# General exception of testing

class ContesterError(Exception):

def \_\_init\_\_(self):

self.message = 'Testing system error'

# Server response exception

class ApiServiceError(ContesterError):

def \_\_init\_\_(self):

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

self.message = 'Server Response Error'

# Execution exception

class ExecutionError(ContesterError):

def \_\_init\_\_(self):

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

self.message = 'Execution Error'

# Wrong answer exception

class WrongAnswerError(ContesterError):

def \_\_init\_\_(self):

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

self.message = 'Wrong Answer'

# Wrong answer exception

class TimeOutError(ContesterError):

def \_\_init\_\_(self):

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

self.message = 'Time Limit Error'

***tester/languages.py***

import typing as t

from dataclasses import dataclass

from flask import current\_app as app

from flask import url\_for

from app.utils.singleton import SingletonBaseClass

@dataclass(frozen=True)

class Language:

name: str

fullname: str

compiler: str

mode: str

icon\_name: str = 'default'

is\_default: bool = False

@property

def icon\_url(self) -> str:

with app.app\_context(), app.test\_request\_context():

return url\_for('static', filename=f'images/svg/language\_icons/{self.icon\_name}.svg')

@dataclass(frozen=True)

class Languages(metaclass=SingletonBaseClass):

# Dictionary with programming languages (name, compiler, CodeMirror mode)

dictionary = {

'cpp': Language(

name='C++',

fullname='GNU C++ 11.1',

compiler='gcc-11.1.0',

mode='text/x-c++src',

icon\_name='cpp'

),

'csharp': Language(

name='C#',

fullname='C# Mono 6.12',

compiler='mono-6.12.0.122',

mode='text/x-csharp',

icon\_name='csharp'

),

'python': Language(

name='Python 3',

fullname='Python 3.8.9',

compiler='cpython-3.8.9',

mode='text/x-python',

icon\_name='python',

is\_default=True

),

'pypy': Language(

name='Pypy 3',

fullname='Pypy 3.7 (7.3.4)',

compiler='pypy-3.7-v7.3.4',

mode='text/x-python',

icon\_name='python'

),

'pascal': Language(

name='Pascal',

fullname='Free Pascal 3.2.0',

compiler='fpc-3.2.0',

mode='text/x-pascal',

)}

\_language\_not\_found = Language(

name='Not found',

fullname='Not found',

compiler='Not found',

mode='Not found'

)

def get\_language(self, language: str, object\_only: t.Optional[bool] = False) -> t.Union[dict, Language]:

current\_language = self.dictionary.get(language, None)

if current\_language is None:

success = False

language\_object = self.\_language\_not\_found

else:

success = True

language\_object = current\_language

if object\_only:

return language\_object

return {

'success': success,

'language': language\_object

}

languages = Languages()

try:

app.jinja\_env.globals['languages'] = languages

except RuntimeError:

print('Cannot set jinja2 global variable')

***tester/types.py***

import typing as t

from app.models import Test

from .languages import Language

class SingleTestResult(t.NamedTuple):

message: str

success: bool

test: Test

user\_output: str

class ContesterResponse(t.NamedTuple):

language: Language

tests: t.Iterable[SingleTestResult]

passed\_tests: int

time: str

***tester/utils.py***

import typing as t

import platform

from functools import wraps

from asyncio.proactor\_events import \_ProactorBasePipeTransport

from .types import SingleTestResult

def get\_number\_of\_passed\_tests(tests: t.Iterable[SingleTestResult]) -> int:

return len([result for result in tests if result.success])

def silence\_event\_loop\_closed(func):

@wraps(func)

def wrapper(self, \*args, \*\*kwargs):

try:

return func(self, \*args, \*\*kwargs)

except RuntimeError as e:

if str(e) != 'Event loop is closed':

raise

return wrapper

if platform.system() == 'Windows':

\_ProactorBasePipeTransport.\_\_del\_\_ = silence\_event\_loop\_closed(\_ProactorBasePipeTransport.\_\_del\_\_)

***blueprints/admin/forms.py***

from flask\_wtf import FlaskForm

from wtforms import StringField, TextAreaField, SelectField, BooleanField, FieldList, FormField, SubmitField

from wtforms.validators import DataRequired

class TestForm(FlaskForm):

stdin = TextAreaField('Ввод', validators=[DataRequired()])

stdout = TextAreaField('Вывод', validators=[DataRequired()])

is\_hidden = BooleanField('Скрыть', default=True)

class TopicForm(FlaskForm):

grade\_id = SelectField('Класс', choices=[], validators=[DataRequired()])

name = StringField('Название', validators=[DataRequired()])

submit = SubmitField()

class TaskForm(FlaskForm):

grade\_id = SelectField('Класс', choices=[], validators=[DataRequired()])

topic\_id = SelectField('Тема', choices=[], validators=[DataRequired()])

name = StringField('Название', validators=[DataRequired()])

condition = TextAreaField('Условие', validators=[DataRequired()])

example\_stdin = TextAreaField('Ввод', validators=[DataRequired()])

example\_stdout = TextAreaField('Вывод', validators=[DataRequired()])

tests = FieldList(FormField(TestForm), min\_entries=1)

submit = SubmitField('Создать задачу')

***blueprints/admin/routes.py***

from enum import Enum

from datetime import date

from sqlalchemy import asc, desc, func, not\_

from flask import current\_app as app

from flask import render\_template, request, redirect, url\_for, abort

from flask\_login import current\_user

from flask\_breadcrumbs import register\_breadcrumb

from app import db

from app.blueprints.admin import admin

from app.models import User, Topic, Task, Submission, Report

from app.utils.forms import init\_grades\_select, init\_topics\_select

from .forms import TopicForm, TaskForm

class ActionType(str, Enum):

CREATE = 'create'

EDIT = 'edit'

@admin.before\_request

def admin\_role\_required():

if not current\_user.is\_authenticated:

return redirect(url\_for('auth.login\_page'))

if not current\_user.is\_admin:

abort(403)

@admin.add\_app\_template\_global

def check\_for\_reports():

if db.session.query(Report).first():

return True

return False

@admin.route('/')

@register\_breadcrumb(admin, '.admin', 'Админ панель')

def home\_page():

page = request.args.get('table\_page', type=int, default=1)

submission\_table = {

'submissions': (

db.session.query(Submission).filter(

func.date(Submission.submission\_date) == date.today()

).order\_by(

desc(Submission.submission\_date)

).paginate(

per\_page=app.config['RECORDS\_PER\_PAGE'], page=page, error\_out=False

)

),

'show\_task': True,

'show\_users': True

}

return render\_template('admin/admin.html', title='Админ панель', \*\*submission\_table)

@admin.route('/students', methods=['GET', 'POST'])

def students\_page():

number = request.args.get('number')

letter = request.args.get('letter')

users = db.session.query(User).filter(

User.grade.has(number=number),

User.grade\_letter == letter

).order\_by(

asc(User.surname)

).all()

table\_data = []

for user in users:

solved\_today = db.session.query(Submission).filter(

func.date(Submission.submission\_date) == date.today(),

not\_(Submission.test\_results.any(success=False)),

Submission.users.any(id=user.id)

).count()

table\_data.append({

'user': user,

'solved\_today': solved\_today

})

return render\_template('admin/students.html', title=f'Ученики {number}{letter} класса', table\_data=table\_data)

@admin.route('/task/create', methods=['GET', 'POST'])

@register\_breadcrumb(admin, '.admin.create\_task', 'Создание задачи')

def create\_task\_page():

form = TaskForm()

init\_grades\_select(form=form)

init\_topics\_select(form=form)

return render\_template('admin/task.html', title='Создать задачу', form=form, action=ActionType.CREATE)

@admin.route('/task/edit', methods=['GET', 'POST'])

@register\_breadcrumb(admin, '.admin.edit\_task', 'Редактирование задачи')

def edit\_task\_page():

task\_id = request.args.get('id')

task = db.session.query(Task).get\_or\_404(task\_id)

form = TaskForm(

obj=task,

grade\_id=task.topic.grade\_id,

condition=task.text,

example\_stdin=task.example.example\_input,

example\_stdout=task.example.example\_output

)

init\_grades\_select(form=form)

init\_topics\_select(form=form, grade\_id=task.topic.grade\_id)

return render\_template('admin/task.html', title='Редактировать задачу', form=form, action=ActionType.EDIT)

@admin.route('/topic/create', methods=['GET', 'POST'])

@register\_breadcrumb(admin, '.admin.create\_topic', 'Создание темы')

def create\_topic\_page():

form = TopicForm()

init\_grades\_select(form=form)

return render\_template('admin/topic.html', title='Создать тему', form=form, action=ActionType.CREATE)

@admin.route('/topic/edit', methods=['GET', 'POST'])

@register\_breadcrumb(admin, '.admin.edit\_topic', 'Редактирование темы')

def edit\_topic\_page():

topic\_id = request.args.get('id')

topic = db.session.query(Topic).get\_or\_404(topic\_id)

form = TopicForm(obj=topic)

init\_grades\_select(form=form)

return render\_template('admin/topic.html', title='Редактировать тему', form=form, action=ActionType.EDIT)

@admin.route('/reports', methods=['GET', 'POST'])

@register\_breadcrumb(admin, '.admin.reports', 'Жалобы')

def reports\_page():

reports = db.session.query(Report).all()

return render\_template('admin/reports.html', title='Жалобы', reports=reports)

***blueprints/api/admin.py***

"""

Module with Admin APIs

"""

from flask import request, make\_response, jsonify, url\_for

from app import db

from app.models import Grade, Topic, Task, Example, load\_user

from app.blueprints.api import api

from .utils import send\_alert, tests\_to\_orm\_objects

@api.route('/topics', methods=['POST'])

def get\_topics():

data = request.json

grade = db.session.query(Grade).filter(Grade.id == data['grade\_id']).first()

topics\_array = []

for topic in grade.topics:

topics\_array.append({

'id': topic.id,

'name': topic.name

})

return jsonify({'topics': topics\_array})

@api.route('/admin/topic', methods=['POST'])

def create\_topic():

data = request.json

topic = Topic(

grade\_id=data['grade\_id'],

name=data['name'].strip()

)

topic.set\_translit\_name()

if not db.session.query(Topic).filter(Topic.translit\_name == topic.translit\_name).first():

db.session.add(topic)

db.session.commit()

return send\_alert(True, 'Тема успешно создана!')

else:

return send\_alert(False, 'Тема с таким именем уже существует')

@api.route('/admin/topic/<topic\_id>', methods=['PUT'])

def update\_topic(topic\_id):

data = request.json

try:

topic = db.session.query(Topic).get(topic\_id)

# Updating general info

topic.grade\_id = data['grade\_id']

topic.name = data['name']

db.session.commit()

return send\_alert(True, 'Тема успешно обновлена!')

except Exception:

return send\_alert(False, 'Не удалось обновить тему')

@api.route('/admin/task', methods=['POST'])

def create\_task():

data = request.json

# Task

task = Task(

topic\_id=data['path']['topic\_id'],

name=data['info']['name'].strip(),

text=data['info']['condition'].strip()

)

task.set\_translit\_name()

# Checking whether a task with this name already exists

topic = db.session.query(Topic).get(data['path']['topic\_id'])

translit\_names = [task\_.translit\_name for task\_ in topic.tasks]

if task.translit\_name in translit\_names:

return send\_alert(False, 'Задача с таким именем уже существует')

else:

db.session.add(task)

db.session.commit()

# Example

example = Example(

task\_id=task.id,

example\_input=data['example']['stdin'],

example\_output=data['example']['stdout']

)

db.session.add(example)

# Tests

tests\_to\_orm\_objects(data['tests'], task)

return send\_alert(True, 'Задача успешно создана')

@api.route('/admin/task/<task\_id>', methods=['PUT'])

def update\_task(task\_id):

data = request.json

try:

task = db.session.query(Task).get(task\_id)

# Updating general info

task.topic\_id = data['path']['topic\_id']

task.name = data['info']['name'].strip()

task.text = data['info']['condition'].strip()

task.set\_translit\_name()

# Updating example

task.example.example\_input = data['example']['stdin']

task.example.example\_output = data['example']['stdout']

# Deleting old tests

for test in task.tests:

db.session.delete(test)

db.session.commit()

# Adding new tests

tests\_to\_orm\_objects(data['tests'], task)

return send\_alert(True, 'Задача успешно обновлена')

except Exception:

return send\_alert(False, 'Не удалось обновить задачу')

@api.route('/admin/task', methods=['DELETE'])

def delete\_task():

data = request.json

task = db.session.query(Task).get(data['task\_id'])

topic\_url = url\_for(

'problems.topic\_page',

grade\_number=task.topic.grade.number,

topic\_translit\_name=task.topic.translit\_name

)

db.session.delete(task)

db.session.commit()

return make\_response(jsonify({'success': True, 'redirect\_url': topic\_url}), 200)

@api.route('/admin/user/<user\_id>', methods=['DELETE'])

def delete\_user(user\_id):

user = load\_user(user\_id)

db.session.delete(user)

db.session.commit()

return make\_response(jsonify({'success': True, 'redirect\_url': url\_for('home\_page')}), 200)

***blueprints/api/auth.py***

"""

Module with Auth APIs

"""

from flask import request, session, make\_response, jsonify, render\_template, url\_for

from flask\_login import login\_user, logout\_user, current\_user

from app import db, serializer

from app.models import User, Role

from app.blueprints.api import api

from app.utils.email import send\_email

from .utils import send\_alert

@api.route('/auth/sign-up', methods=['POST'])

def signup():

data = request.json

if db.session.query(User).filter(User.email == data['email']).first():

return send\_alert(False, 'Пользователь с этой почтой уже зарегистрирован!')

if data['password'] != data['password\_again']:

return send\_alert(False, 'Пароли не совпадают')

user = User(

name=data['firstname'].strip().capitalize(),

surname=data['lastname'].strip().capitalize(),

email=data['email'].strip(),

role\_id=db.session.query(Role).filter(Role.name == 'user').first().id,

grade\_id=data['grade'],

grade\_letter=data['letter'],

)

user.set\_password(data['password'])

db.session.add(user)

db.session.commit()

next\_url = session['next\_url'] or url\_for('home\_page')

return make\_response(jsonify({'success': True, 'redirect\_url': next\_url}), 200)

@api.route('/auth/login', methods=['POST'])

def login():

data = request.json

user = db.session.query(User).filter(User.email == data['email']).first()

# Error

if not user:

return send\_alert(False, 'Неверная почта или пароль')

# Success

elif user.check\_password(data['password']):

login\_user(user)

if 'next\_url' in session:

next\_url = session['next\_url']

else:

next\_url = url\_for('home\_page')

return make\_response(jsonify({'success': True, 'redirect\_url': next\_url}), 200)

# Error

else:

return send\_alert(False, 'Неверная почта или пароль')

@api.route('/auth/logout', methods=['POST'])

def logout():

logout\_user()

return make\_response(jsonify({'redirect\_url': url\_for('home\_page')}), 200)

@api.route('/auth/confirm-email', methods=['PUT'])

def confirm\_email():

try:

email = current\_user.email

token = serializer.dumps(email, salt='confirm-email')

link = url\_for('auth.confirm\_email', token=token, \_external=True)

print(email, token, link)

send\_email(subject='Подтвердите адрес электронной почты',

recipients=[email],

html\_body=render\_template('auth/confirm\_email.html', link=link))

except Exception:

return send\_alert(False, 'При отправке письма произошла ошибка')

return send\_alert(True, 'Письмо с подтверждением отправлено')

***blueprints/api/task.py***

"""

Module with Task APIs

"""

from flask import request, make\_response, jsonify, render\_template

from flask\_login import current\_user

from app import db

from app.contester import contester, ContesterResponse

from app.models import Report, load\_user

from app.blueprints.api import api

from app.utils.db import get\_task

from .utils import send\_alert

def render\_solution\_failure(message: str):

return make\_response(jsonify({

'status': 'FAILED',

'result': render\_template(

'responses/solution/failure.html',

message=message

)

}), 200)

def render\_solution\_success(response: ContesterResponse):

return make\_response(jsonify({

'status': 'OK',

'result': render\_template(

'responses/solution/success.html',

response=response

)

}), 200)

@api.route('/task/solution', methods=['POST'])

def send\_solution():

# Checking if user is authenticated

if not current\_user.is\_authenticated:

return render\_solution\_failure('Для отправки решений необходимо войти в систему')

data = request.json

# Checking if user's or partners profile is verified

partner = load\_user(data['partner\_id']) # Partner

if not current\_user.verified:

return render\_solution\_failure('Для отправки решений вам необходимо подтвердить свою почту')

if partner:

if not partner.verified:

return render\_solution\_failure('Для отправки решений вашему партнеру необходимо подтвердить свою почту')

path = data['path']

task = get\_task(path['grade'], path['topic'], path['task']) # Task

code = data['code'].strip() # Code

response = contester.run\_tests(

code=code,

language=data['lang'],

task=task,

partner=partner

)

if response is not None:

return render\_solution\_success(response)

return render\_solution\_failure('Что-то пошло не так!')

@api.route('/task/report', methods=['POST'])

def send\_report():

data = request.json

try:

path = data['path']

task = get\_task(path['grade'], path['topic'], path['task'])

report = Report(

user\_id=current\_user.id,

task\_id=task.id,

text=data['text']

)

db.session.add(report)

db.session.commit()

return send\_alert(True, 'Жалоба успешно отправлена')

except Exception:

return send\_alert(False, 'Не удалось отправить жалобу')

@api.route('/task/report', methods=['DELETE'])

def delete\_report():

data = request.json

try:

report = db.session.query(Report).get(data['report\_id'])

db.session.delete(report)

db.session.commit()

return send\_alert(True, 'Проблема отмечена как решенная')

except Exception:

return send\_alert(False, 'Не удалось выполнить действие')

***blueprints/api/utils.py***

"""

Module with API utils

"""

import typing as t

from flask import make\_response, jsonify

from app import db

from app.models import Task, Test

def send\_alert(success: bool, message: str):

"""Returns typed response than can be processes by sendDefaultAjax function in JS file"""

return make\_response(jsonify(

{

'success': success,

'message': message

}

), 200)

def tests\_to\_orm\_objects(tests: dict, task: Task) -> t.Iterable[Test]:

"""Saves tests to database and returns list of ORM Test objects"""

tests\_zip = zip(tests['stdin\_list'], tests['stdout\_list'], tests['is\_hidden\_list'])

tests\_list = []

for stdin, stdout, is\_hidden in tests\_zip:

test = Test(

task\_id=task.id,

stdin=stdin,

stdout=stdout,

is\_hidden=is\_hidden

)

tests\_list.append(test)

db.session.add(test)

db.session.commit()

return tests\_list

***blueprints/auth/forms.py***

from flask\_wtf import FlaskForm

from wtforms import PasswordField, StringField, SubmitField, SelectField

from wtforms.fields.html5 import EmailField

from wtforms.validators import DataRequired, Length

class SignUpForm(FlaskForm):

first\_name = StringField('Имя', validators=[DataRequired()])

last\_name = StringField('Фамилия', validators=[DataRequired()])

grade = SelectField('Класс', choices=[])

letter = SelectField('Буква', choices=['А', 'Б', 'В', 'Г'])

email = EmailField('Email', validators=[DataRequired()])

password = PasswordField('Пароль', validators=[DataRequired(), Length(min=6)])

password\_again = PasswordField('Повторите пароль', validators=[DataRequired()])

submit = SubmitField('Зарегистрироваться')

class LoginForm(FlaskForm):

email = EmailField('Email', validators=[DataRequired()])

password = PasswordField('Password', validators=[DataRequired()])

submit = SubmitField('Войти')

***blueprints/auth/routes.py***

from flask import render\_template, redirect, url\_for

from flask\_login import current\_user, login\_required

from itsdangerous import SignatureExpired, BadSignature

from app import db, serializer

from app.blueprints.auth import auth

from app.models import Grade

from .forms import LoginForm, SignUpForm

@auth.route('/login', methods=['GET', 'POST'])

def login\_page():

form = LoginForm()

return render\_template('auth/login.html', title='Вход', form=form)

@auth.route('/signup', methods=['GET', 'POST'])

def signup\_page():

grades = db.session.query(Grade).all()

grades\_list = [(grade.id, grade.number) for grade in grades]

form = SignUpForm()

form.grade.choices = grades\_list

return render\_template('auth/sign\_up.html', title='Регистрация', form=form)

@auth.route('/confirm-email/<string:token>')

@login\_required

def confirm\_email(token):

if not current\_user.verified:

try:

serializer.loads(token, salt='confirm-email', max\_age=3600)

current\_user.verified = True

db.session.commit()

except SignatureExpired:

return render\_template(

'auth/after\_confirm.html',

title='Подтверждение почты',

status='Время подтверждения истекло',

success=False

)

except BadSignature:

return render\_template(

'auth/after\_confirm.html',

title='Подтверждение почты',

status='Несуществующий токен подтверждения!',

success=False

)

return render\_template(

'auth/after\_confirm.html',

title='Подтверждение почты',

status='Ваша почта подтверждена!',

success=True

)

return redirect(url\_for('home\_page'))

***blueprints/errors/handlers.py***

from flask import render\_template, make\_response

from werkzeug.exceptions import HTTPException

from app.blueprints.errors import errors

ERRORS\_DATA = {

403: {

'description': 'У вас не доступа к этой странице'

},

404: {

'description': 'Данной страницы не существует'

},

500: {

'description': 'Внутренняя ошибка сервера'

}

}

DEFAULT\_ERRORS\_DATA = {

'description': 'Что-то пошло не так'

}

@errors.app\_errorhandler(HTTPException)

def handle\_exception(e):

data = ERRORS\_DATA.get(e.code, DEFAULT\_ERRORS\_DATA)

return make\_response(render\_template('errors/error.html', code=e.code, data=data), e.code)

***blueprints/problems/routes.py***

from flask import current\_app as app

from flask import render\_template, redirect, url\_for, request, abort

from flask\_login import current\_user, login\_required

from flask\_breadcrumbs import register\_breadcrumb

from sqlalchemy import desc

from app import db

from app.blueprints.problems import problems

from app.models import Grade, Topic, Submission

from app.contester.languages import languages

from app.utils.routes import grade\_compliance\_required, admin\_required

from app.utils.db import get\_task

import app.utils.breadcrumbs as bc

@problems.route('/redirect', methods=['GET'])

@login\_required

def redirect\_page():

if current\_user.is\_admin:

return redirect(url\_for('problems.all\_grades\_page'))

return redirect(url\_for('problems.grade\_page', grade\_number=current\_user.grade.number))

@problems.route('/', methods=['GET'])

@login\_required

@admin\_required

def all\_grades\_page():

return render\_template('problems/all\_grades.html', title='Все классы', grades=db.session.query(Grade).all())

@problems.route('/grade-<int:grade\_number>', methods=['GET'])

@register\_breadcrumb(problems, '.grade', '', dynamic\_list\_constructor=bc.view\_grade\_dlc)

@login\_required

@grade\_compliance\_required

def grade\_page(grade\_number):

grade = db.session.query(Grade).filter(Grade.number == grade\_number).first\_or\_404()

topics = grade.topics.all()

context = {

'grade': grade,

'topics': topics,

}

return render\_template('problems/grade.html', title=f'{grade.number} класс', \*\*context)

@problems.route('/grade-<int:grade\_number>/<string:topic\_translit\_name>', methods=['GET'])

@register\_breadcrumb(problems, '.grade.topic', '', dynamic\_list\_constructor=bc.view\_topic\_dlc)

@login\_required

@grade\_compliance\_required

def topic\_page(grade\_number, topic\_translit\_name):

grade = db.session.query(Grade).filter(Grade.number == grade\_number).first\_or\_404()

topic = db.session.query(Topic).filter(Topic.translit\_name == topic\_translit\_name).first\_or\_404()

tasks = topic.tasks.all()

print(tasks)

context = {

'grade': grade,

'topic': topic,

'tasks': tasks

}

return render\_template('problems/topic.html', title=topic.name, \*\*context)

@problems.route(

'/grade-<int:grade\_number>/<string:topic\_translit\_name>/<string:task\_translit\_name>',

defaults={'tab': 'problem'},

methods=['GET']

)

@problems.route(

'/grade-<int:grade\_number>/<string:topic\_translit\_name>/<string:task\_translit\_name>/<string:tab>',

methods=['GET']

)

@register\_breadcrumb(problems, '.grade.topic.task', '', dynamic\_list\_constructor=bc.view\_task\_dlc)

@login\_required

@grade\_compliance\_required

def task\_page(grade\_number, topic\_translit\_name, task\_translit\_name, tab):

task = get\_task(grade\_number, topic\_translit\_name, task\_translit\_name)

topic = task.topic

global\_context = {

'task': task,

'topic': topic,

}

page = request.args.get('table\_page', type=int, default=1)

if tab == 'problem':

# Tab with problem (name, condition, editor etc)

local\_context = {

'language\_dict': languages.dictionary

}

return render\_template(

'problems/tabs/problem.html',

title=f'{task.name} - Задача',

\*\*global\_context,

\*\*local\_context

)

elif tab == 'submissions':

# Tab with user's submissions

local\_context = {

'submissions': current\_user.submissions.filter(

Submission.task\_id == task.id

).paginate(

per\_page=app.config['RECORDS\_PER\_PAGE'], page=page, error\_out=False

)

}

return render\_template(

'problems/tabs/submissions.html',

title=f'{task.name} - Мои отправки',

\*\*global\_context,

\*\*local\_context

)

elif tab == 'all-submissions':

# Tab with all submissions

if current\_user.is\_admin:

local\_context = {

'submissions': task.submissions.order\_by(

desc(Submission.submission\_date)

).paginate(

per\_page=app.config['RECORDS\_PER\_PAGE'], page=page, error\_out=False

),

'show\_users': True

}

return render\_template(

'problems/tabs/all\_submissions.html',

title=f'{task.name} - Все отправки',

\*\*global\_context,

\*\*local\_context

)

abort(403)

abort(404)