**Содержание**

[Введение 3](#_Toc166081766)

[1 Постановка задачи 4](#_Toc166081767)

[1.1 Обзор аналогичных решений 4](#_Toc166081768)

[1.2 Спецификация требований 6](#_Toc166081769)

[2 Проектирование web-приложения 7](#_Toc166081770)

[2.1 Проектирование вариантов использования 7](#_Toc166081771)

[2.2 Структура web-приложения 7](#_Toc166081772)

[2.3 Проектирование базы данных 8](#_Toc166081773)

[2.4 Проектирование сервера web-приложения 10](#_Toc166081774)

[3 Разработка web-приложения 15](#_Toc166081775)

[3.1 Разработка бэкэнда 15](#_Toc166081776)

[3.2 Разработка фронтенда 17](#_Toc166081777)

[4 Тестирование web-приложения 19](#_Toc166081778)

[5 Руководство пользователя 23](#_Toc166081779)

[Заключение 29](#_Toc166081780)

[Список используемых источников 30](#_Toc166081781)

[Приложение А 31](#_Toc166081782)

Введение

В мире, где технологии развиваются со скоростью мысли, доступ к образованию играет ключевую роль в формировании будущего. Быстро растущий спрос на предоставление образовательных услуг формирует необходимость наличия интернет-платформ, которые позволят людям быстро найти источники информации в удобном для них формате. Особую востребованность в этой сфере приобрели курсы по программированию, как одно из самых восстребованных и быстро развивающихся направлений.

Целью курсового проектирования является разработка распределённого web-приложения, предоставляющего удобную и эффективную платформу для прохождения курсов по программированию. Приложение должно позволять пользователям создавать аккаунты для прохождения курсов. Также приложение должно предоставлять пользователям возможность просматривать доступные курсы и записываться на них, для дальнейшего прохождения. Приложение должно предоставлять администратору возможность размещать, редактировать и удалять курсы, а также просматривать аккаунты пользователей. Помимо базовых функций поиска и фильтрации, пользователь также имеет возможность просматривать курсы на которые он записан и уже прошёл.

В приложении должны быть разграничены возможности администраторов и пользователей. Так, гости должны иметь возможность только просматривать доступные курсы.

В ходе выполнения курсового проектирования будут решены следующие задачи:

– анализ литературы по теме работы;

– изучение требований и определение вариантов использования;

– анализ и проектирование архитектуры приложения, модели базы данных;

– тестирование приложения;

– создание руководства пользователя.

1 Постановка задачи

1.1 Обзор аналогичных решений

В качестве первого аналогичного решения была рассмотрена платформа GreatLearning, на которой предлагаются услуги университетов и преподавателей со всего мира. Она специализируется на предоставлении курсов с самообучением, а не на предоставлении постоянного сопровождения. Пример страницы GreatLearning представлен на рисунке 1.1.

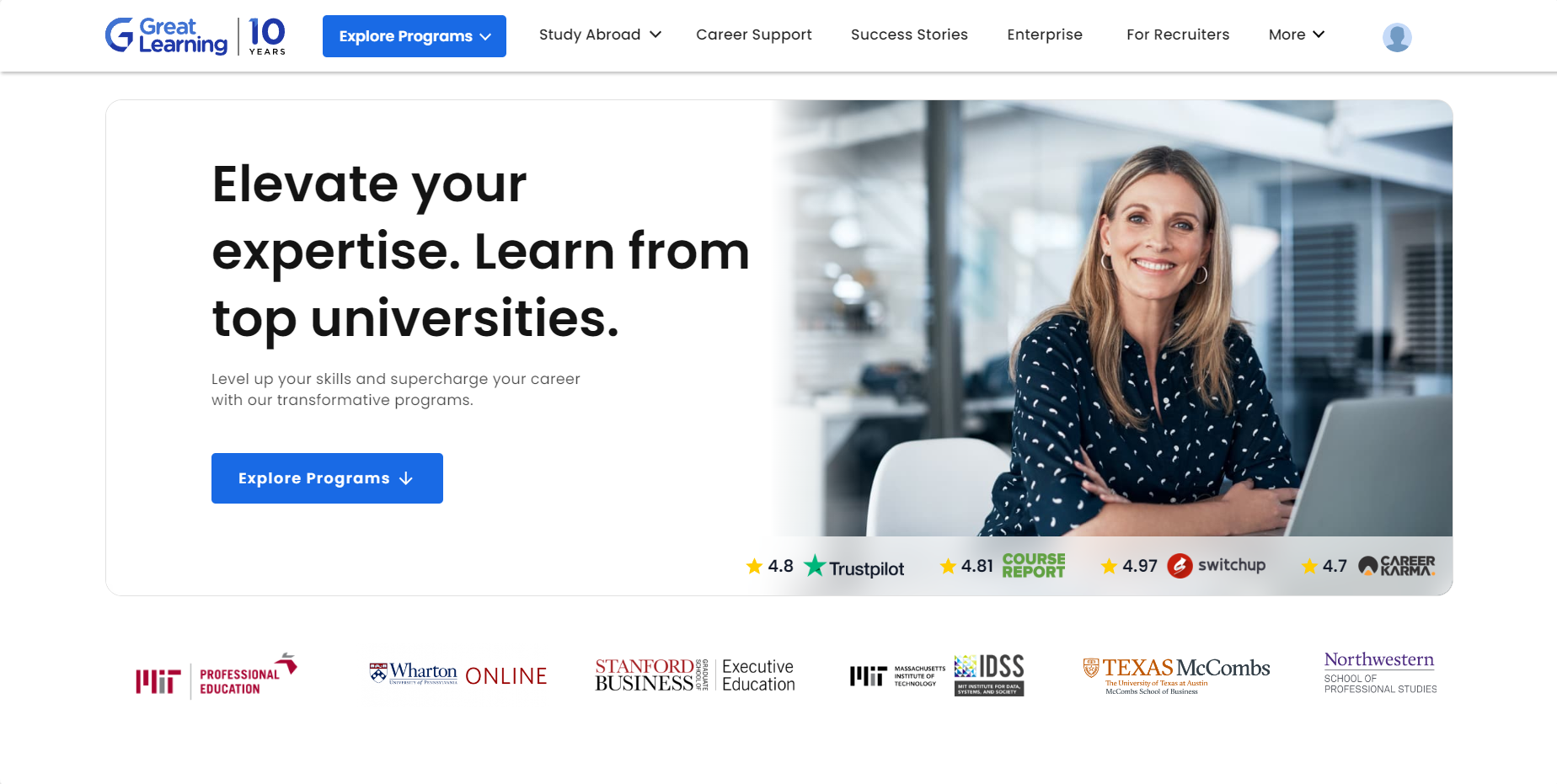


Рисунок 1.1 – Страница GreatLearning

Пользователи могут найти курсы по различным сферам в области информационных технологий, таким как машинное обучение и искусственный интеллект, облачные вычисления, аналитика данных и многое другое. Особенностью GreatLearning является ориентация на самообучение, то есть пользователи ищут и проходят курсы самостоятельно, а решение о прохождении того или иного курса мотивируется описанием, и личными предпочтениями пользователя к изучению конкретной технологии.

Преимуществом платформы являются простой процесс записи на курс, наличие большого количества сертификатов как наград за прохождение курсов. К недостаткам относится недостаточная возможность фильтрации курсов: платформа разбивает их по категориям, и осуществлять фильтрацию внутри категорий можно только по нескольким признакам, таким как время выполнения и уровень навыков исполнителя.

В качестве второго аналогичного решения была рассмотрена платформа OTUS, предлагающая более широкий спектр курсов и вебинаров для клиентов. Платформа предоставляет доступ к множеству уникальных программ в различных областях, включая разработку ПО, управление и так далее. OTUS обеспечивает более сложный процесс регистрации и прохождения курсов, включающий комплексны практически этапы, работу с менторами, присутствует возможность организации и просмотра профиля. Также на платформе присутствует система обратной связи и защиты платежей. Пример страницы платформы представлен на рисунке 1.2.

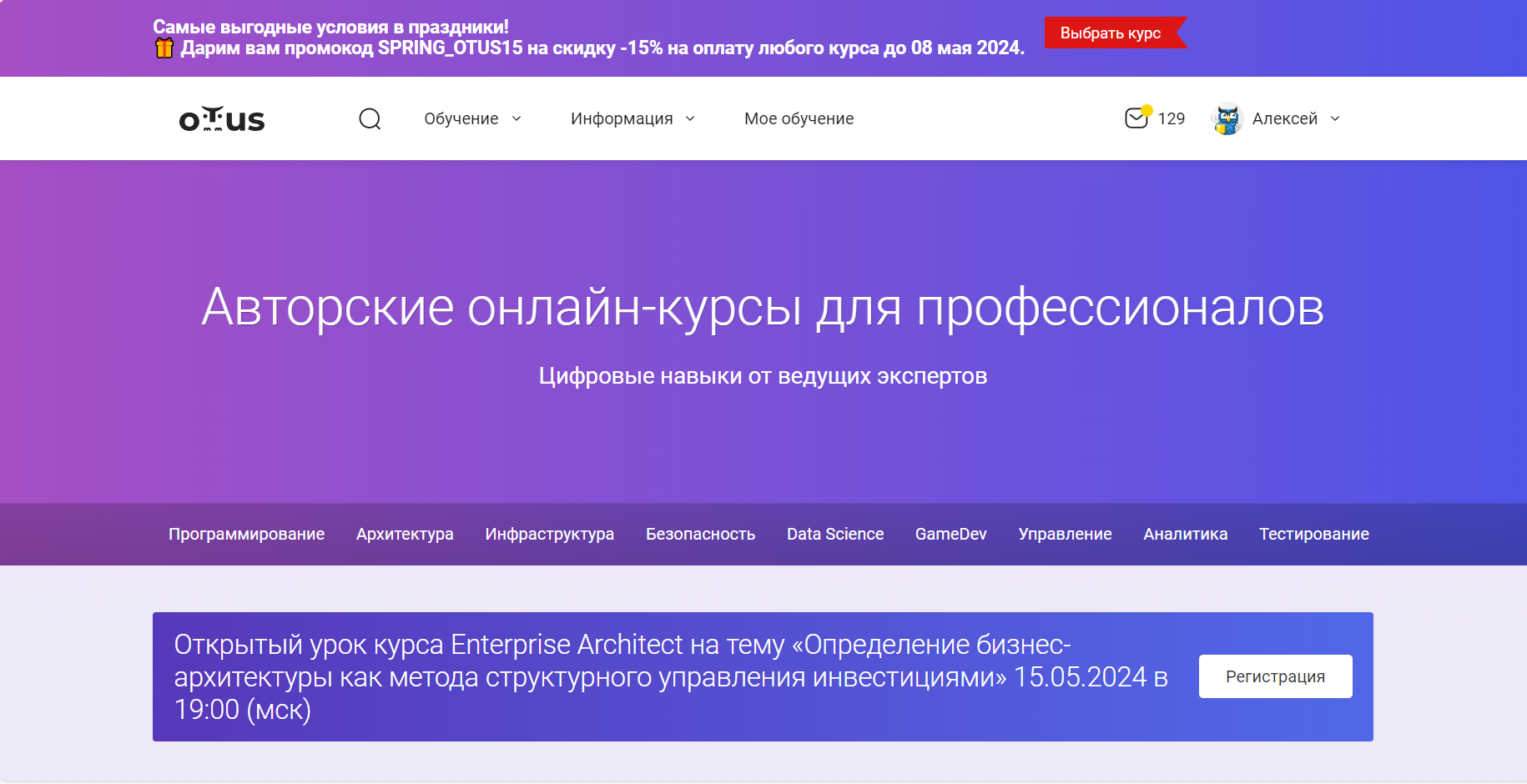


Рисунок 1.2 – Страница OTUS

К преимуществам платформы относится постоянная обратная связь, обеспечивающаяся за счёт постоянно работающей линии пользовательской поддержки, телеграмм ботов и т.д. К недостаткам относится недостаточно интуитивная навигация по сайту и недостаточное количество фильтров, кроме общего определённым областям.

В качестве последнего аналогичного решения была рассмотрена платформа Skillbox – платформа с широким функционалом по поиску курсов и менторов. Пример страницы сайта представлен на рисунке 1.3.

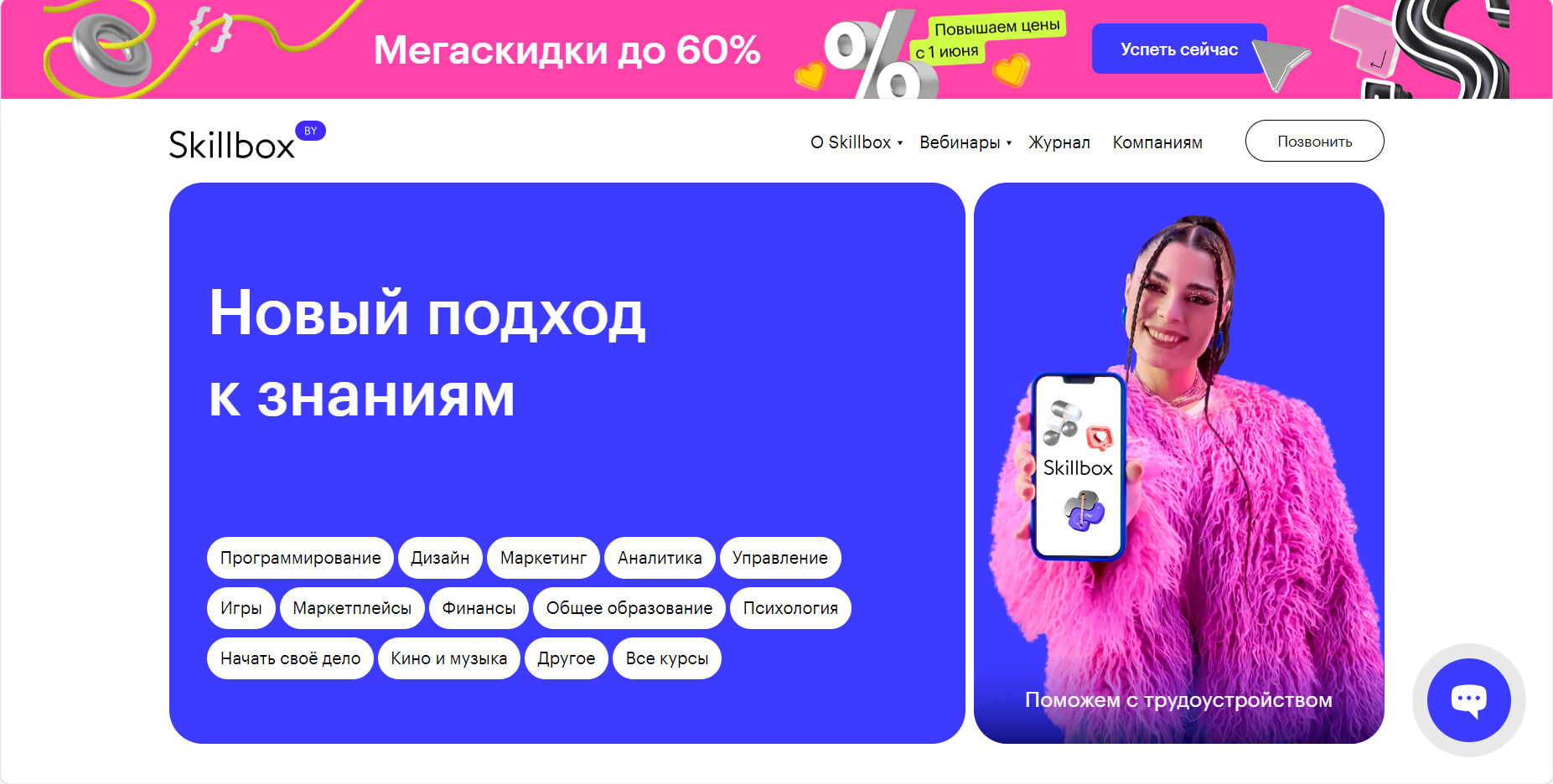


Рисунок 1.3 – Страница Upwork

Она предлагает широкий функционал по поиску курсов и менторов, а также обширные возможности фильтрации по параметрам.

1.2 Спецификация требований

На основе рассмотренных аналогичных решений были сформированы следующие требования к программному продукту:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли администратора и пользователя;
* позволять пользователю просматривать доступные курсы;
* позволять пользователю записаться на конкретный курс;
* позволять администраторам редактировать описание курса;
* позволять пользователю просматривать курсы, на которые он записан;
* предоставлять возможность поиска курса по параметрам;
* позволять администраторам изменять названия курсов;
* позволять администраторам добавлять новые курсы;
* позволять администраторам удалять курсы;
* позволять пользователю проходить курсы;
* позволять пользователю просматривать пройденные курсы.

Всем пользователям, включая гостей, должна быть доступна возможность просматривать курсы и их детали.

Клиенту должна быть доступна возможность просматривать курсы, записываться и проходить их, просматривать пройденные и те на которые записан в данный момент. После записи на курс пользователю приходит на указанную при записи электронную почту письмо с материалами курса.

Администратору должна быть доступна возможность добавлять курсы, изменять и удалять их, получать список пользователей.

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

2 Проектирование web-приложения

2.1 Проектирование вариантов использования

Согласно сформулированным требованиям была создана диаграмма вариантов использования, представленная на рисунке 2.1.

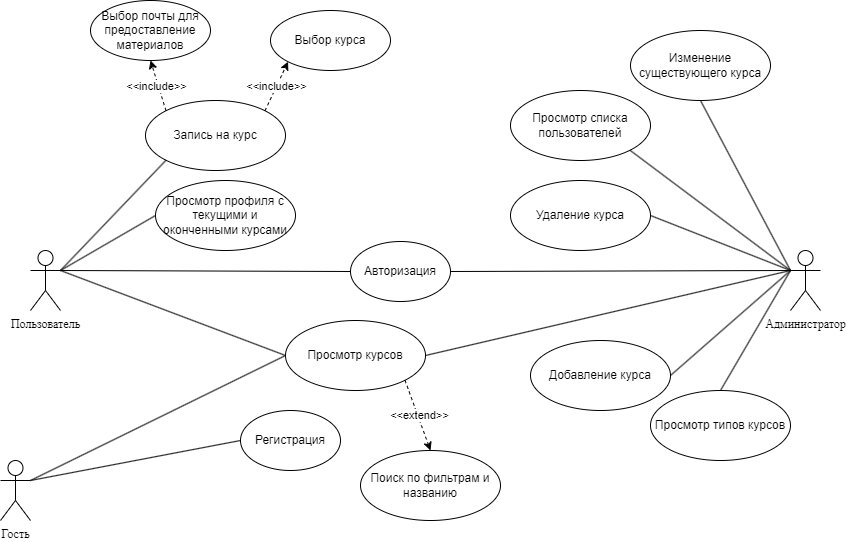


Рисунок 2.1 – Диаграмма вариантов использования

На данной диаграмме все действия, доступные гостю, доступны также всем остальным пользователям.

В свою очередь пользователи могут записываться на курсы, что включает в себя указание почты на которую предоставляются материалы для обучения и выбор курса, также.

Администраторам доступны возможности по созданию, удалению и модификации крусов. Также они могут просматривать список пользователей зарегистрированных на платформе, а также типов, используемых в курсах.

Все пользователи проходят через одинаковые этапы идентификации, аутентификации и авторизации на платформе.

2.2 Структура web-приложения

Обобщённая структура web-приложения представлена на рисунке 2.2.

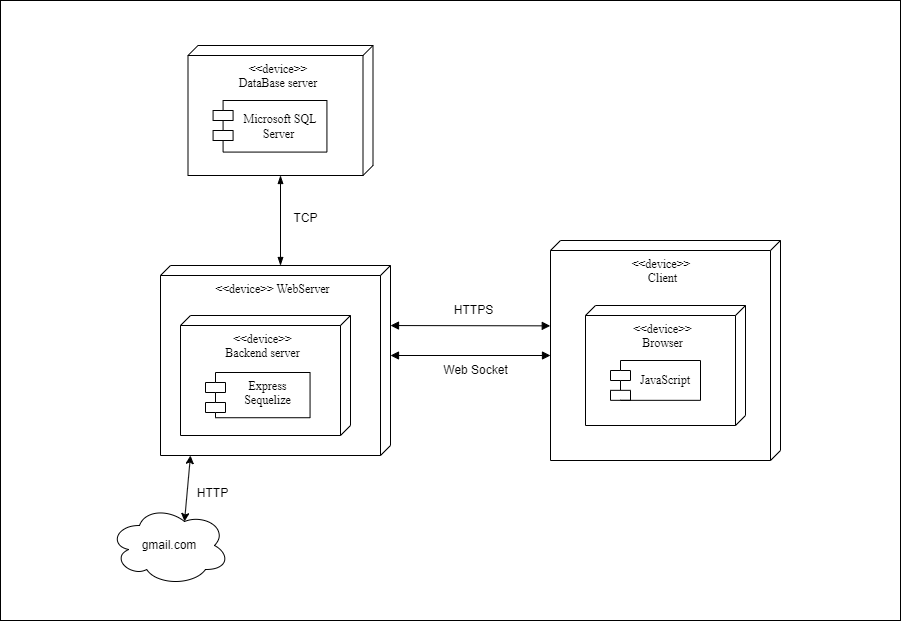


Рисунок 2.2 – Диаграмма развёртывания web-приложения

Согласно данной схеме, клиент и сервер находятся на разных устройствах. Клиент использует браузер для отправки запросов к web-серверу. Клиент и сервер могут обмениться сообщениям по протоколам HTTPS и WebSocket.

Web-сервер обрабатывает запросы при помощи сервера. В случае необходимости сервер отправляет запросы к развёрнутой локально базе данных, под управлением СУБД MSSQL.

Для выполнения запросов к базе данных используется ORM Sequelize.

2.3 Проектирование базы данных

Согласно схеме вариантов использования была создана база данных. Её логическая схема представлена на рисунке 2.3. Описание моделей представлено в приложении А.

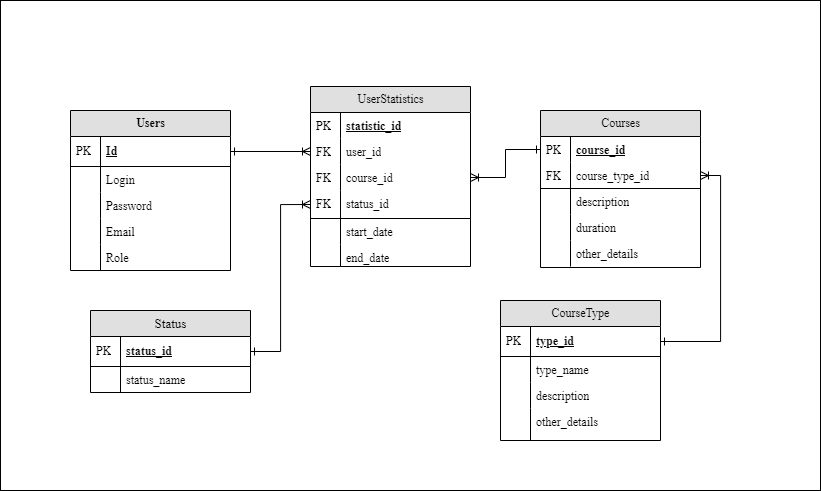


Рисунок 2.3 – Логическая схема базы данных

База данных содержит пять таблиц, хранящих информацию о пользователях, курсах, типах курсов, статусах курсов и статистических данных.

Таблица Users хранит информацию о зарегистрированных пользователях. Описание её столбцов представлено в таблице 2.1.

Таблица 2.1 – Описание таблицы Users

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| Id | integer | идентификатор пользователя, первичный ключ |
| Login | nvarchar(50) | логин пользователя |
| Password | nvarchar(255) | хеш пароля пользователя |
| Role | integer | является ли пользователь администратором |
| Email | nvarchar(50) | адрес электронной почты пользователя |
| SessionId | integer | идентификатор пользовательской сессии |

Таблица CoursesTypes хранит информацию о типах курсов на платформе. Описание её столбцов представлено в таблице 2.2.

Таблица 2.2 – Описание таблицы CoursesTypes

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| type\_id | integer | идентификатор типа, первичный ключ |
| Type\_name | nvarchar(50) | наименование типа |
| description | nvarchar(500) | описание типа курса |
| Other\_details | nvarchar(1000) | дополнительное описание типа курса |

Таблица Courses хранит информацию о курсах размещённых на платформе. Описание её столбцов представлено в таблице 2.3.

Таблица 2.3 – Описание таблицы Courses

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| course\_id | integer | идентификатор курса, первичный ключ |
| course\_name | nvarchar(250) | наименование курса |
| description | nvarchar(500) | описание курса |
| duration | float | продолжительность курса |
| course\_type\_id | integer | идентификатор типа курса, внешний ключ |
| other\_details | nvarchar(1000) | подробное описание курса |

Таблица UserStatistics хранит информацию курсах на которые подписан пользователь, и их состояния. Описание её столбцов представлено в таблице 2.4.

Таблица 2.4 – Описание таблицы UserStatistics

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| statistic\_id | integer | идентификатор статистической единица, первичный ключ |
| user\_id | integer | идентификатор пользователя, внешний ключ |
| course\_id | integer | идентификатор курса внешний ключ |
| status\_id | integer | идентификатор статуса курса, внешний ключ |
| start\_date | date | дата начала курса |
| end\_date | date | дата окончания курса |

2.4 Проектирование сервера web-приложения

Для обработки запросов применяется восемь роутеров, каждый из которых обрабатывает запросы к определённым адресам. Так, index.js служит для обработки запросов по перенаправлению на другие роутеры. Он предоставляет обработку для главной страницы, открываемой по умолчанию с методом GET.

Роутер adminRouter отвечает за обработку запросов администратора. Они все должны начинаться с /admin. Список представляемых обработчиков представлен в таблице 2.5.

Таблица 2.5 – Cписок представляемых обработчиков роутера adminRouter

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес | Требует авторизации | Метод | Описание |
| / | Да | GET | Возвращает страницу администратора |
| /users | Да | GET | Возвращает всех пользователей зарегистрированных на платформе, с данными об идентификаторе и почте |
| /types | Да | GET | Возвращает все типы курсов, с названием типа, идентификатором, описанием и дополнительной информацией типа. |
| /courses | Да | GET | Возвращает все курсы, с названием курса, идентификатором, продолжительностью, описанием и технологиями необходимыми для его прохождения |
| /editcourse/:id | Да | GET | Получает страницу для редактирования курса по его идентификатору |
| /addcourse | Да | GET | Возвращает страницу для добавления нового курса. Включающую название, описание, навыки для прохождения, продолжительность и тип курса |
| /addcourse | Да | POST | Добавляет новый курс |
| /updatecourse/:id | Да | POST | Обновляет информацию о курсе по его идентификатору. Включает название, описание, навыки для прохождения курса, его продолжительность в часах, и тип курса |
| /deletecourse/:id | Да | DELETE | Удаляет курс по его идентификатору. |

Роутер authRouter предназначен для обработки запросов на авторизацию. Все запросы к нему должны начинаться с /auth. Список представляемых обработчиков представлен в таблице 2.6.

Таблица 2.6 – Cписок представляемых обработчиков роутера authRouter

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес | Требует авторизации | Метод | Описание |
| /auth/login | Нет | POST | Проверяет переданные учетные данные, аутентифицирует пользователя и перенаправляет на главную страницу или на сохраненный URL после успешной аутентификации. Если пользователь с указанным именем не найден, перенаправляет на страницу регистрации. Если пароль не совпадает, возвращает ошибку 401 |
| /auth/register | Нет | POST | Регистрация нового пользователя. Проверяется наличие кандидата с таким же именем пользователя. |

Продолжение таблицы 2.6

|  |  |  |  |
| --- | --- | --- | --- |
| /auth/logout | Да | GET | Пользователь выходит из системы. Текущая сессия уничтожается, и пользователь перенаправляется на главную страницу |
| /auth/status | Да | GET | Получение статуса аутентификации. Возвращает информацию о том, аутентифицирован ли текущий пользователь |

Роутер courseRouter предназначен для обработки запросов пользователей на получении информации о курсах размещённых на платформе. Они все должны ныачинаться с /courses. Методы не требуют авторизации, так как предоставляют общедоступную информацию о курсах и их деталях. Список представляемых обработчиков представлен в таблице 2.7.

Таблица 2.7 –­ Список представляемых обработчиков роутера courseRouter

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес | Требует авторизации | Метод | Описание |
| /courses | Нет | GET | Получает все курсы. Если в запросе указан параметр type, то возвращаются курсы определенного типа, в противном случае возвращаются все курсы |
| /courses/:id | Нет | GET | Получает информацию о конкретном курсе по его идентификатору. Если курс с указанным идентификатором не найден, возвращается ошибка 404. Возвращает также название типа курс |

Роутер entryRouter предназначен для обработки запросов пользователей по записи на курсы. Они все должны начинаться с /courses. Список представляемых обработчиков представлен в таблице 2.8.

Таблица 2.8 ­– Список представляемых обработчиков роутера entryRouter

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес | Требует авторизации | Метод | Описание |
| /entry | Да | GET | Получает информацию о записи на курс. Если не указан параметр courseId, перенаправляет на страницу курсов. Если пользователь не аутентифицирован, сохраняет URL текущей страницы и перенаправляет на страницу входа. Возвращает страницу с информацией о курсе, на который записан пользователь |

Продолжение таблицы 2.8

|  |  |  |  |
| --- | --- | --- | --- |
| /entry/:id | Да | POST | Добавляет запись пользователя на курс. Проверяет, не записан ли пользователь уже на этот курс. Если запись не найдена, создает новую запись в базе данных и отправляет письмо с подтверждением на указанный адрес электронной почты. В случае успеха перенаправляет на страницу с курсами. |

Роутер homeRouter отвечает за отображение главной страницы приложения. Он загружает информацию о типах курсов из базы данных и передает ее в шаблон главной страницы для отображения. Все запросы должны начинаться с /. Метод контроллера homeRouter не требует авторизации, так как отвечает только за отображение главной страницы приложения. Список представляемых обработчиков представлен в таблице 2.9.

Таблица 2.9 ­– Список представляемых обработчиков роутера homeRouter

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес | Требует авторизации | Метод | Описание |
| / | Да | GET | Получает информацию о записи на курс. Если не указан параметр courseId, перенаправляет на страницу курсов. Если пользователь не аутентифицирован, сохраняет URL текущей страницы и перенаправляет на страницу входа. Возвращает страницу с информацией о курсе, на который записан пользователь |

Роутер profileRouter отвечает за отображение информации о профиле пользователя. Он предоставляет доступ к различным страницам профиля, таким как основная страница профиля, страница завершенных курсов и страница курсов в процессе. Также позволяет обновлять статус завершения курса. Все запросы должны начинаться с /profile. Список представляемых обработчиков представлен в таблице 2.10.

Таблица 2.10 ­– Список представляемых обработчиков роутера profileRouter

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес | Требует авторизации | Метод | Описание |
| /profile | Да | GET | Отображает информацию о профиле пользователя. Если пользователь не аутентифицирован, перенаправляет на страницу регистрации. |

Продолжение таблицы 2.10

|  |  |  |  |
| --- | --- | --- | --- |
| /profile/finished | Да | GET | Отображает список завершенных курсов пользователя. Загружает информацию о завершенных курсах пользователя из базы данных и передает ее в шаблон страницы для отображения. |
| /profile/inprocess | Да | GET | Отображает список курсов, на которые пользователь записан и которые он еще не завершил. Загружает информацию о курсах в процессе пользователя из базы данных и передает ее в шаблон страницы для отображения. |
| /profile/update/:courseId | Да | PUT | Обновляет статус курса пользователя на "завершенный". Принимает параметр courseId, по которому находит запись о курсе пользователя в базе данных и обновляет ее статус на "завершенный". |

Роутер userRouter отвечает за обработку запросов, связанных с пользователями. В частности, он предоставляет возможность регистрации нового пользователя и отображения страницы регистрации. Все запросы должны начинаться с /register. Список представляемых обработчиков представлен в таблице 2.11.

Таблица 2.11 ­– Список представляемых обработчиков роутера userRouter

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес | Требует авторизации | Метод | Описание |
| /register | Нет | GET | Отображает страницу регистрации. |
| /register | Нет | POST | Регистрирует нового пользователя. Принимает данные пользователя из тела запроса и создает новую запись в базе данных. Перенаправляет на страницу входа в случае успешной регистрации. |

Для обработки сессий пользователей используется промежуточное ПО для управления сеансами в Express.js, оно позволяет создаватьи управлять сеансами пользователей в приложении. В процессе разработки приложения были созданы роутеры и контроллеры, которые полностью охватывает определённый в требованиях функционал.

3 Разработка web-приложения

3.1 Разработка бэкэнда

Для разработки бэкэнда был использован веб-фреймворк для Node.js – Express.js, в качестве среды для выполнения JavaScript на сервере используется Node.js. На листинге ниже мы настраиваем конфигурацию нашего приложения, для управления сеансами мы используем expressSession,подключаем маршруты из модуля router, используем Handlebars как шаблонизатор.

|  |
| --- |
| app.use(expressSession({  secret: 'SECRET',  resave: false,  saveUninitialized: false  }));  app.use(bodyParser.urlencoded({ extended: false }));  app.engine('.hbs', hbs.engine);  app.set('view engine', '.hbs');  app.use(express.static(\_\_dirname + '/public'));  app.use(bodyParser.json());  app.use('/', router); |

Листинг 3.1 – Простые обработчики запросов

Для обработки запросов использовался router, который перенаправлял запросы на другие роутеры

|  |
| --- |
| const Router = require("express");  const router = new Router();  const homeRouter = require("./homeRouter");  const authRouter = require("./authRouter")  const courseRouter = require("./courseRouter")  const profileRouter = require("./profileRouter")  const entryRouter = require("./entryRouter")  const adminRouter = require("./adminRouter");  router.use("/", homeRouter);  router.use("/auth", authRouter);  router.use("/courses", courseRouter);  router.use("/profile", profileRouter);  router.use("/entry", entryRouter);  router.use("/admin", adminRouter);  module.exports = router; |

Листинг 3.2 – Фрагмент роутера index.js

Для работы с базой данный была выбрана ORM Sequelize. Сначала были созданы модели, соответствующие таблицам базы данных, затем они экспортировались для инициализации в один файл. Это было сделано с целью обращения ко всем моделяем по одному пути. Ниже представлен файл инициализации всех моделей бд.

|  |
| --- |
| const users = require("./users");  const courses = require("./courses");  const status = require("./status");  const statistics = require("./statistics");  const courseTypes = require("./types");  const { DataTypes } = require('sequelize');  function initModels(sequelize) {  const Users = users(sequelize, DataTypes);  const Courses = courses(sequelize, DataTypes);  const Status = status(sequelize, DataTypes);  const Statistics = statistics(sequelize, DataTypes);  const CourseTypes = courseTypes(sequelize, DataTypes);  Statistics.belongsTo(Users, { foreignKey: 'user\_id' });  Users.hasMany(Statistics, { foreignKey: 'user\_id' });  return {Users,Courses,Status,Statistics,CourseTypes};  }  module.exports = initModels;  module.exports.initModels = initModels;  module.exports.default = initModels; |

Листинг 3.3 – Фрагмент файла initModels.js

Каждому роутеру соответсвует свой контроллер, который и обрабатывает запросы, например, пользователь решил перейти к курсам, в index.js роутер перенаправляет запрос, где url содержит /courses в данном случае courseRouter, его содержание в листинге 3.3.

|  |
| --- |
| const { Router } = require('express');  const coursesRouter = new Router();  const coursesController = require('../controllers/coursesController');  coursesRouter  .get('/', coursesController.getAllCourses)  .get('/:id', coursesController.getOneCourse)  module.exports = coursesRouter; |

Листинг 3.4 – Фрагмент роутера courseRouter.js

Далее запрос поступал на контроллер, в котором все запросы и обрабытываются. В данном случае у нас естьдва метода для получения списка всех курсов и получение курса по его идентификатору. В каждом методе мы обрашаемся к нашим моделям для получения данных из базы данных, обрабатываем запросы, полученные сервером, далее идет валидация данных и затем отправляем необходимый ответ пользователю. Пример метода в контроллере курсов представлен ниже в листинге 3.5.

|  |
| --- |
| async getOneCourse(req, res) {  try {  const courseId = req.params.id;  const course = await models.Courses.findByPk(courseId, { include: [models.CourseTypes], raw: true });  if (!course) {  return res.status(404).send('Курс не найден');  }  const type = await models.CourseTypes.findOne({ where: { type\_id: course.course\_type\_id }, raw: true });  res.render('./layouts/courseDetails.hbs', { layout: "courseDetails.hbs", course: course, type: type });  } catch (error) {  console.error('Ошибка при получении курса:', error);  res.status(500).send('Произошла ошибка при получении курса'); |

Листинг 3.5 – Фрагмент контроллера courseController.js

Таким образом была разработана система обработки запросов клиента на сервер, подобным образом были реализованы остальные роутеры и контроллеры.

3.2 Разработка фронтенда

Для реализации фронтенда были использованы express-handlebars. Файлы были разделены на layout и partials, то есть разделение повторяющихся блоков на каждой странице и индивидуальных страниц. Ниже приведен листинг хедера.

|  |
| --- |
| <header><nav><ul>  <li><a href="/courses">Курсы</a></li>  <li><a href="/profile">Ваш профиль</a></li>  <li id="login-logout"><a id="login-logout-link" href="/auth/login">Вход</a></li>  </ul></nav></header>  <script>  document.addEventListener('DOMContentLoaded', function() {  const loginLogoutLink = document.getElementById('login-logout-link');  const xhr = new XMLHttpRequest();xhr.onreadystatechange = function() {  if (xhr.readyState === XMLHttpRequest.DONE) {  if (xhr.status === 200) {  const isAuthenticated = JSON.parse(xhr.responseText).isAuthenticated;  if (isAuthenticated) {  loginLogoutLink.textContent = 'Выход';  loginLogoutLink.href = '/auth/logout';  } else {loginLogoutLink.textContent = 'Вход';  loginLogoutLink.href = '/auth/login';}} else {console.error('Произошла ошибка при выполнении AJAX-запроса');}}};  xhr.open('GET', '/auth/status', true); xhr.send();});</script> |

Листинг 3.6 – Фрагмент header.hbs

В коде выше создается объект XMLHttpRequest для выполнения асинхронных HTTP-запросов, устанавливается обработчик события readystatechange, который срабатывает при изменении состояния объекта XMLHttpRequest.

Выполняется GET-запрос на URL /auth/status, для получения информации о статусе аутентификации пользователя: пользователь не зарегистрирован/логинин, то отображается ссылка для входа в свой аккаунт, иначе предлагает выйти из него.

В листинге 3.5 демонстрируется реализация этого хедера на странице с курсами, так же вывод всех направлений курсов.

|  |
| --- |
| {{> header}}  <select id="course-type-select">  <option value="{{typeIn.type\_id}}">{{typeIn.type\_name}}</option>  <option value="0">Все типы</option>  {{#each courseTypes}}  <option value="{{this.type\_id}}">{{this.type\_name}}</option>  {{/each}}  </select>  <script>  document.getElementById('course-type-select').addEventListener('change', async (event) => {  const selectedType = event.target.value;  if (selectedType && selectedType != 0) {  const currentUrl = new URL(window.location.href);  currentUrl.searchParams.set('type', selectedType);  window.location.href = currentUrl.toString();  } else {  // Если выбран пункт "Все типы", перезагрузить страницу без параметра type  window.location.href = '/courses';  }  }); </script> |

Листинг 3.7 – Фрагмент course.hbs

Таким образом был реализован фронтент данного веб-приложения. На основании всех рассмотренных примеров и фрагментов кода, можно сделать вывод что интерфейс приложения соответствует запрашиваемому уровню, обеспечивает необходимый функционал, представлен в классической цветовой гамме.

4 Тестирование web-приложения

Для тестирования web-приложения использовалось ручное тестирование и тестирование через Postman. Были проведены тесты, которые бы помогли найти неисправность или некоректность работы системы. Сначала пробуем обратиться в Postman к профилю в роли гостя. Результат теста показан на рисунке 4.1.

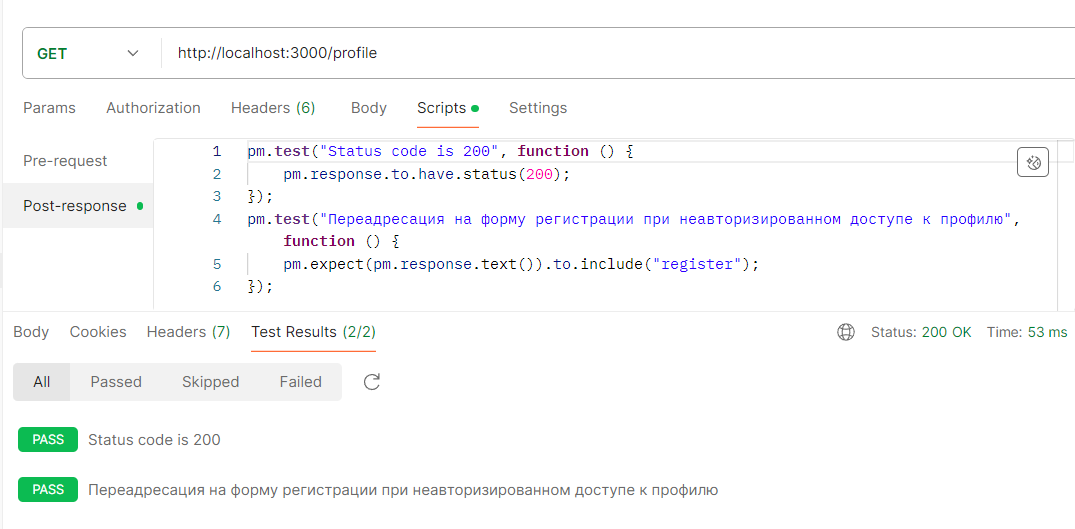


Рисунок 4.1 – Перенаправления на форму регистрации

Далее будем тестировать функционал администратора. Проверяем обработчики на возможность не аутентифицированного и не авторизованного доступа. Результат тестирования приведён на рисунках 4.2 и 4.3 соответственно.

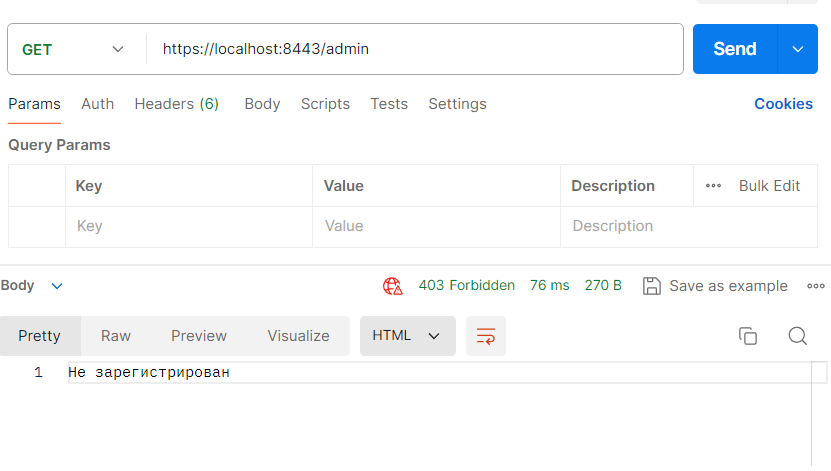


Рисунок 4.2 – Сообщение при попытке получения доступа без аутентификации

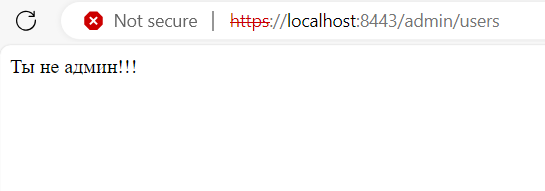


Рисунок 4.3 – Сообщение при попытке получения доступа без необходимой роли

Далее переходим к тестированию панелей по редактированию и добавлению курсов. Логика приложения допускает возможность публикации курсов с одинаковыми названиями, однако время затрачиваемое на прохождение курса ограничено диапазоном чисел с плавающей точкой в диапазоне от 1 до 999 часов.  
Результаты тестирования по добавлению отрицательного числа приведены на рисунке 4.4.

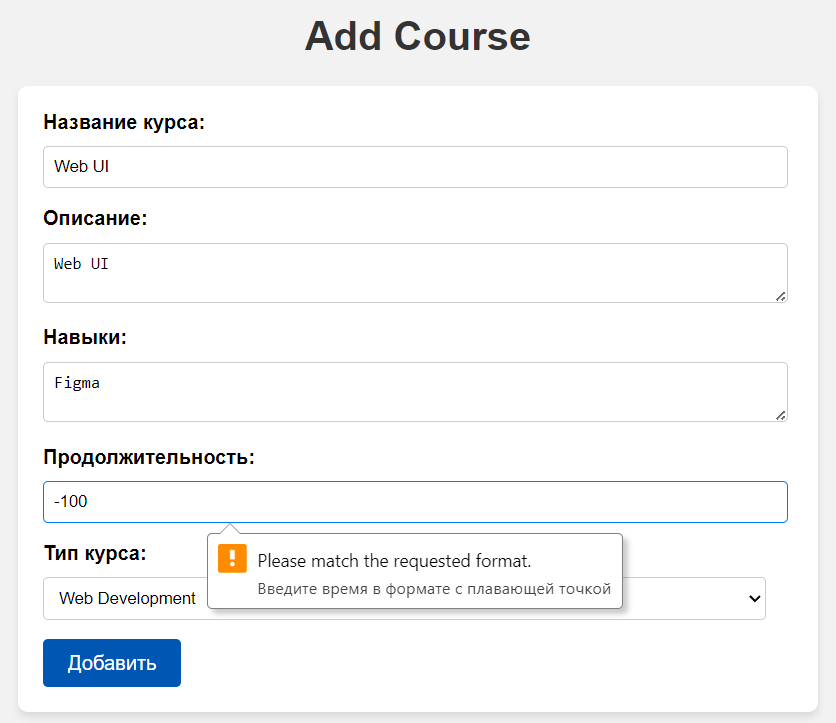


Рисунок 4.4 – Уведомление при попытке ввода некорректной продолжительности курса.

Далее тестируем возможность администратора открыть для редактирования не существующий курс. Результаты тестирования приведены на рис 4.5.

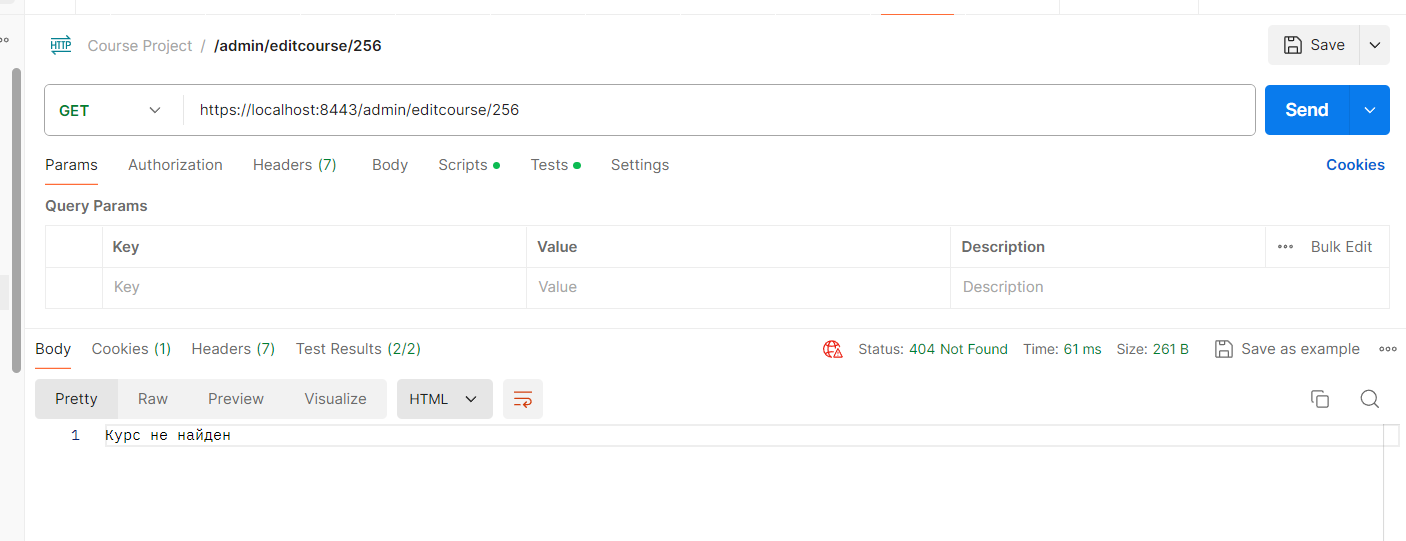


Рисунок 4.5 – Сообщение о попытке получения не существующего курса

Далее протестируем ввод некорректного пароля на форме логина. Результаты тестирования представлены на рисунке 4.6.

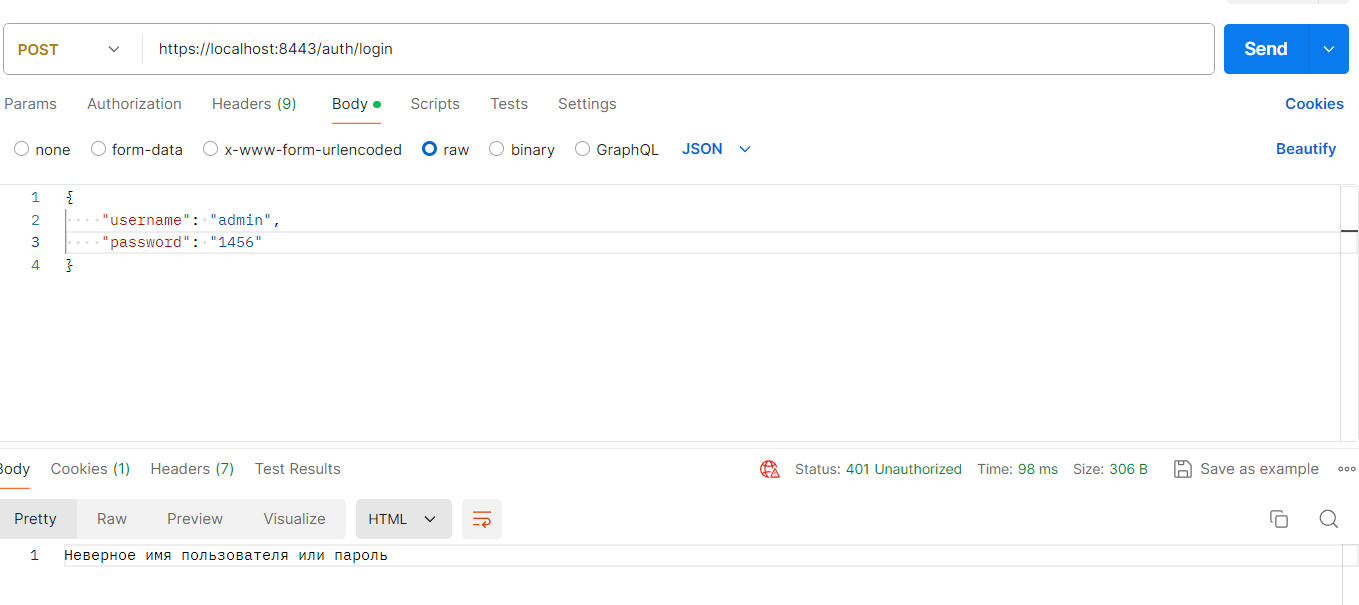


Рисунок 4.6 – Сообщение о вводе неверных данных на форме входа

Далее протестируем отправку материалов для прохождения курса, результаты тестирования представлены на рисунке 4.7, предоставляющем информацию со стороны отправителя, и 4.7 б), с информацией со стороны получателя.

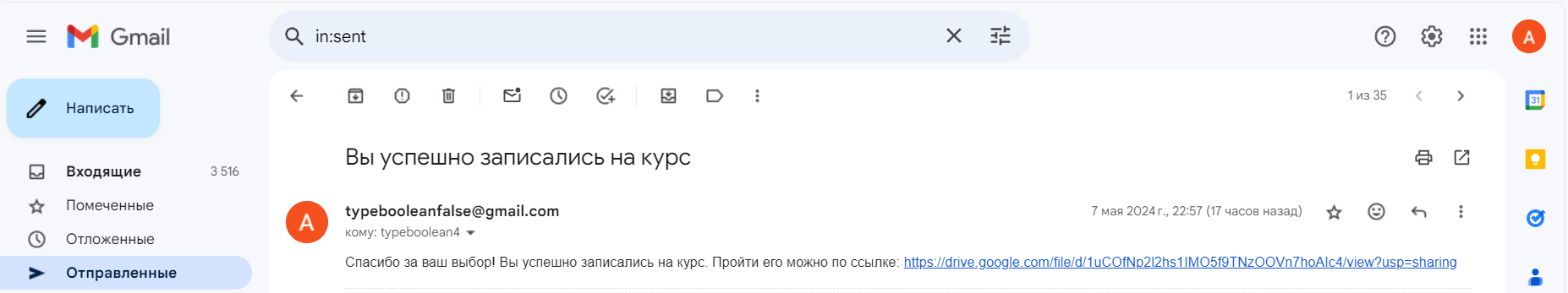


Рисунок 4.7 а) – Информация со стороны отправителя

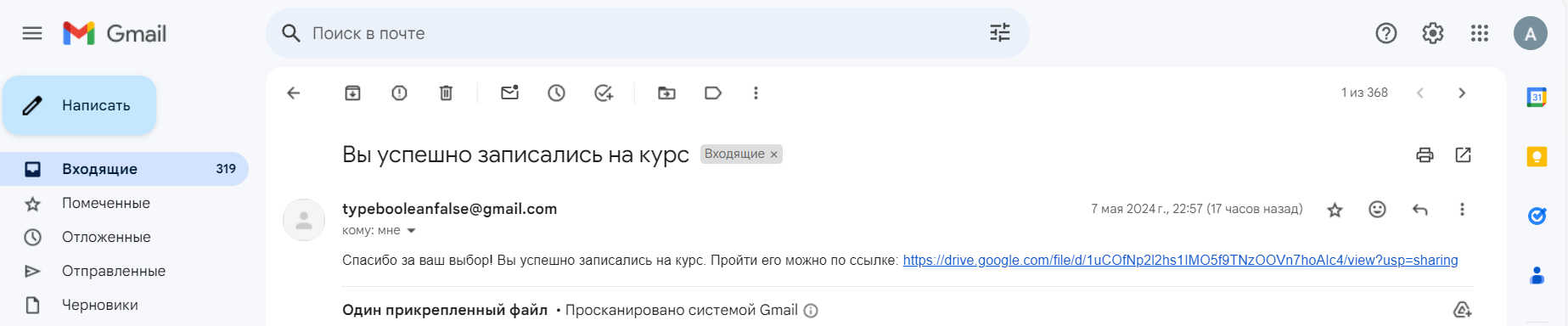


Рисунок 4.7 б) – Информация со стороны получателя

В ходе тестирования веб-приложения “Курсы по программированию” были выявлены и успешно устранены ряд проблем. Это включает в себя несоответствия между данными, запрашиваемыми клиентом, и данными, возвращаемыми сервером, а также ошибки в исходном коде.  
 Были проведены тщательные проверки на различных уровнях приложения, включая интерфейс пользователя, бизнес-логику и взаимодействие с базой данных. Это обеспечило глубокое понимание функциональности приложения и его поведения в различных сценариях.  
 Ошибки в исходном коде, которые могли привести к непредвиденному поведению приложения, были успешно исправлены. Это улучшило общую стабильность приложения и обеспечило более плавный пользовательский опыт.  
 Несмотря на то, что были обнаружены и исправлены некоторые проблемы, важно отметить, что процесс тестирования - это непрерывный процесс. По мере развития приложения и добавления новых функций, необходимо продолжать тестирование для обеспечения его надежности и эффективности.  
 В целом, результаты тестирования демонстрируют прочность и надежность веб-приложения “Курсы по программированию”. Это подтверждает его готовность к использованию конечными пользователями, обеспечивая им качественный и эффективный инструмент для изучения программирования.

5 Руководство пользователя

Web-приложение курсов программирования поддерживает с пользователями с такими ролями как гость, администратор и зарегистрированный пользователь.

При первом открытии сайта гость видит страницу типов, представленную на рисунке 5.1.

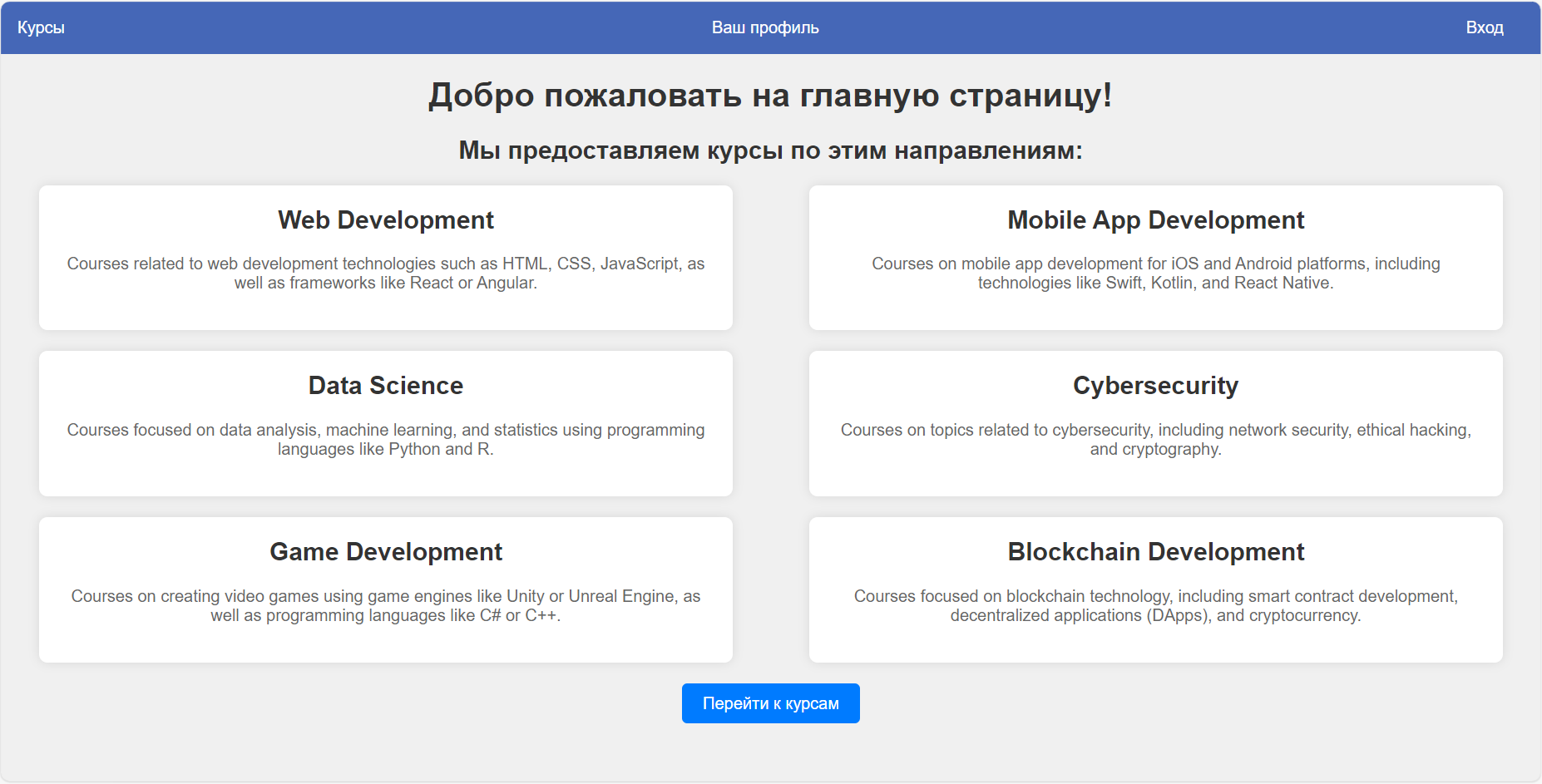


Рисунок 5.1 – Страница направлений курсов

Гостевой пользователь может просмотреть страницу, на которой отображены курсы, размещённые на платформе. На странице гость может ознакомиться с предоставляемыми курсами, получить более подробное опсание каждого из них.

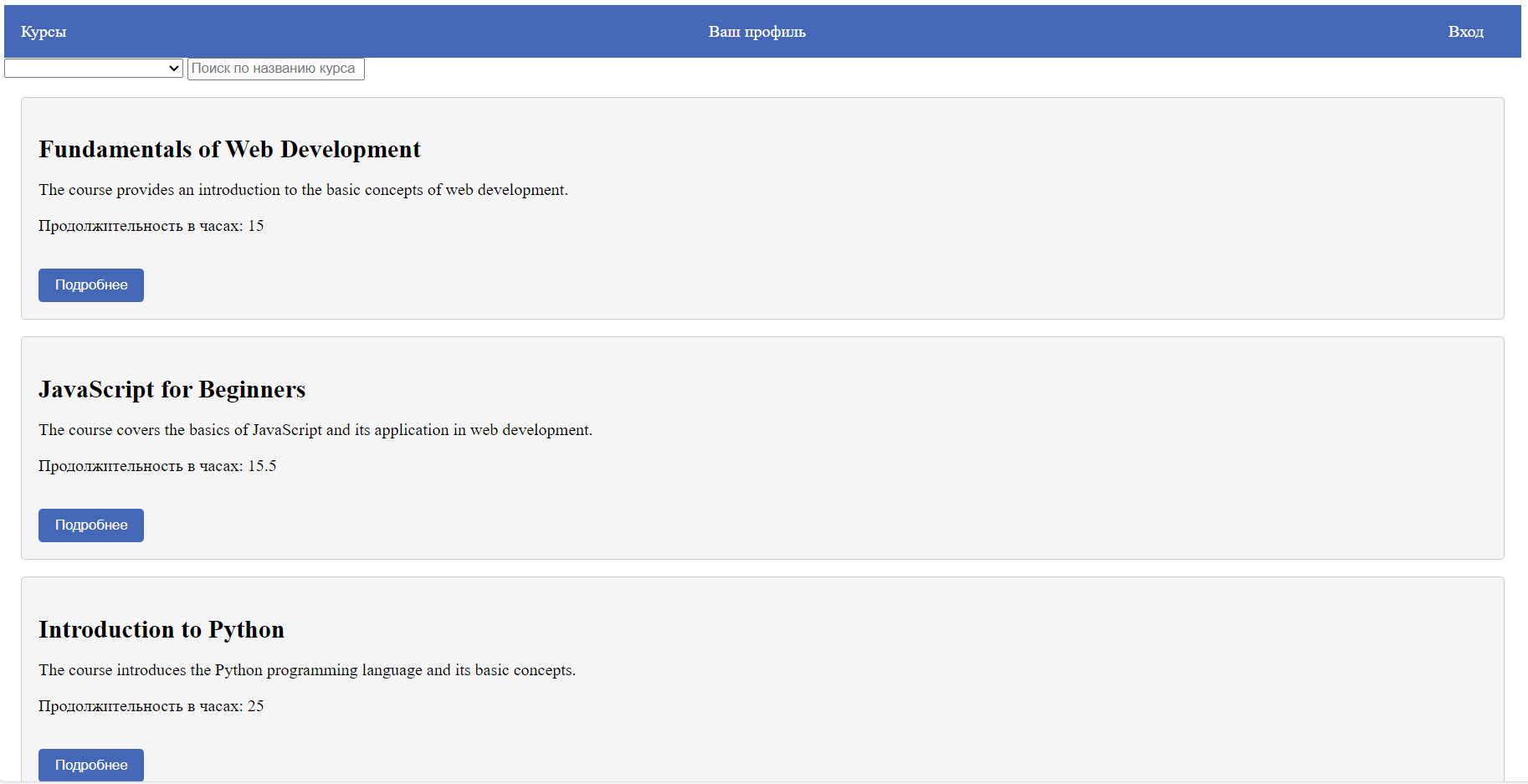


Рисунок 5.2 – Страница со списком курсов

Воспользовавшись элементом интерфейса для входа, или же перейдя на страницу профиля, гость может произвести регистрацию аккаунта, соответствующий интерфейс представлен на рисунке 5.3.

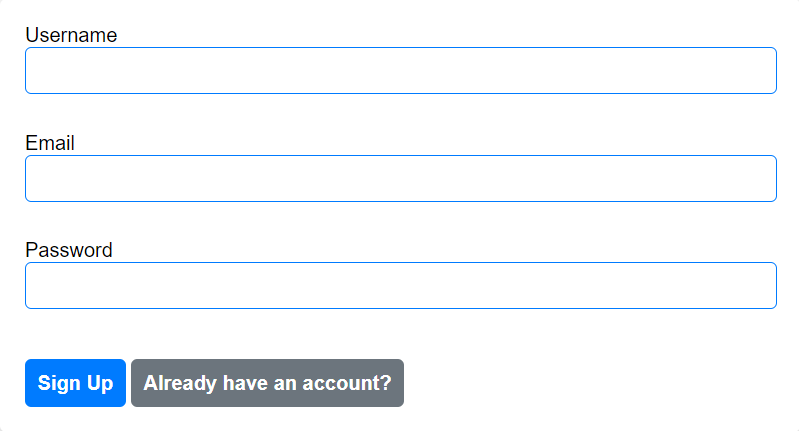


Рисунок 5.3 – Форма регистрации в приложении

Если пользователь создал аккаунт, то используя одноимённый элемент интерфейса, он может произвести вход в свой аккаунт, используя имя пользователя и пароль указанные при регистрации. Соответствующая форма представлена на рисунке 5.4.

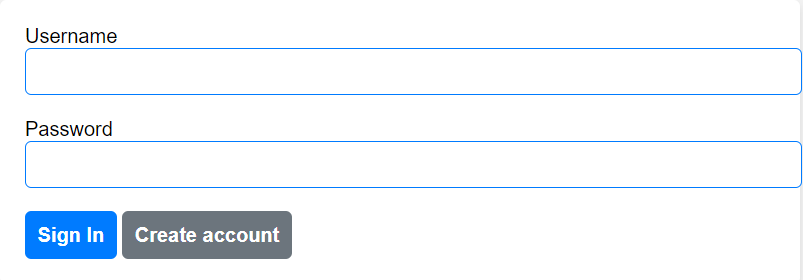


Рисунок 5.4 – Форма входа в аккаунт

Для записи на курс необходимо нажать на кнопку записаться. Если пользователь вошёл в свой аккаунт, его перенаправит на форму записи, иначе на регистрацию, после чего он сможет продолжить. Форма записи представлена на рисунке 5.5.

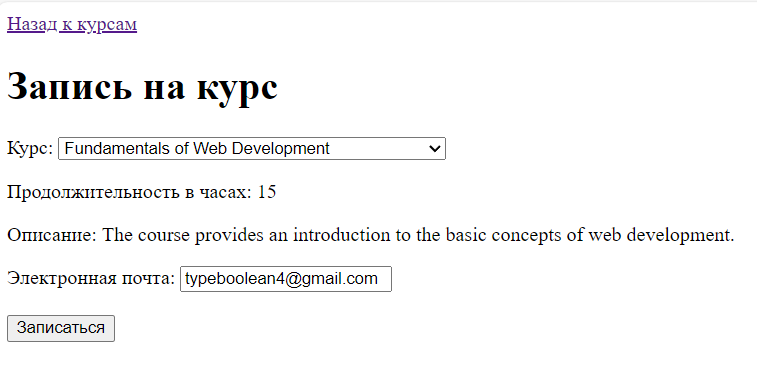


Рисунок 5.5 – Форма записи на курс

Пользователь может поменять курс и так же электронную почту, на которую пользователю прийдет материал для прохождения курсов.

Пользователь, у которого есть аккаунт и он совершил вход, может перейти в свой профиль, в котором он может посмотреть курсы, на которые он записан 5.6 а).

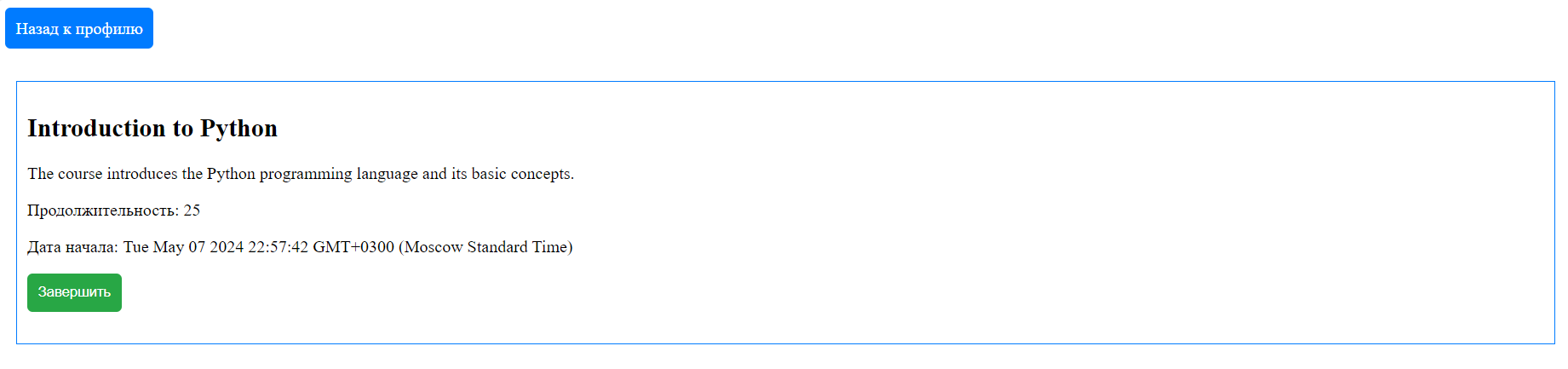


Рисунок 5.6 а) – Страница с курсами на которые пользователь записан

Если по какой-то причине пользователь предпочтёт закончить курс раньше, не пройдя все материалы, он может воспользоваться кнопкой «Завершить» на странице с курсами на которые он записан, после чего принудительно завершённый курс будет перемещён в раздел завершённых курсов в профиле пользователя. Интерфейс этого раздела представлен на рисунке 5.6 б).

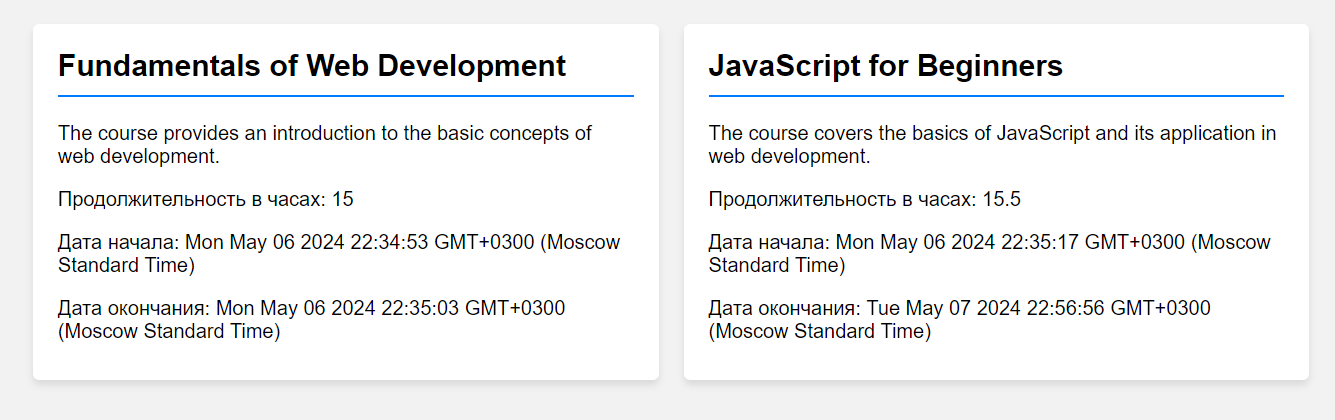


Рисунок 5.6 б) – Страница с завершёнными курсами

Администратор приложения может просматривать всех зарегистрированных пользователей как представлено на рисунке 5.7.

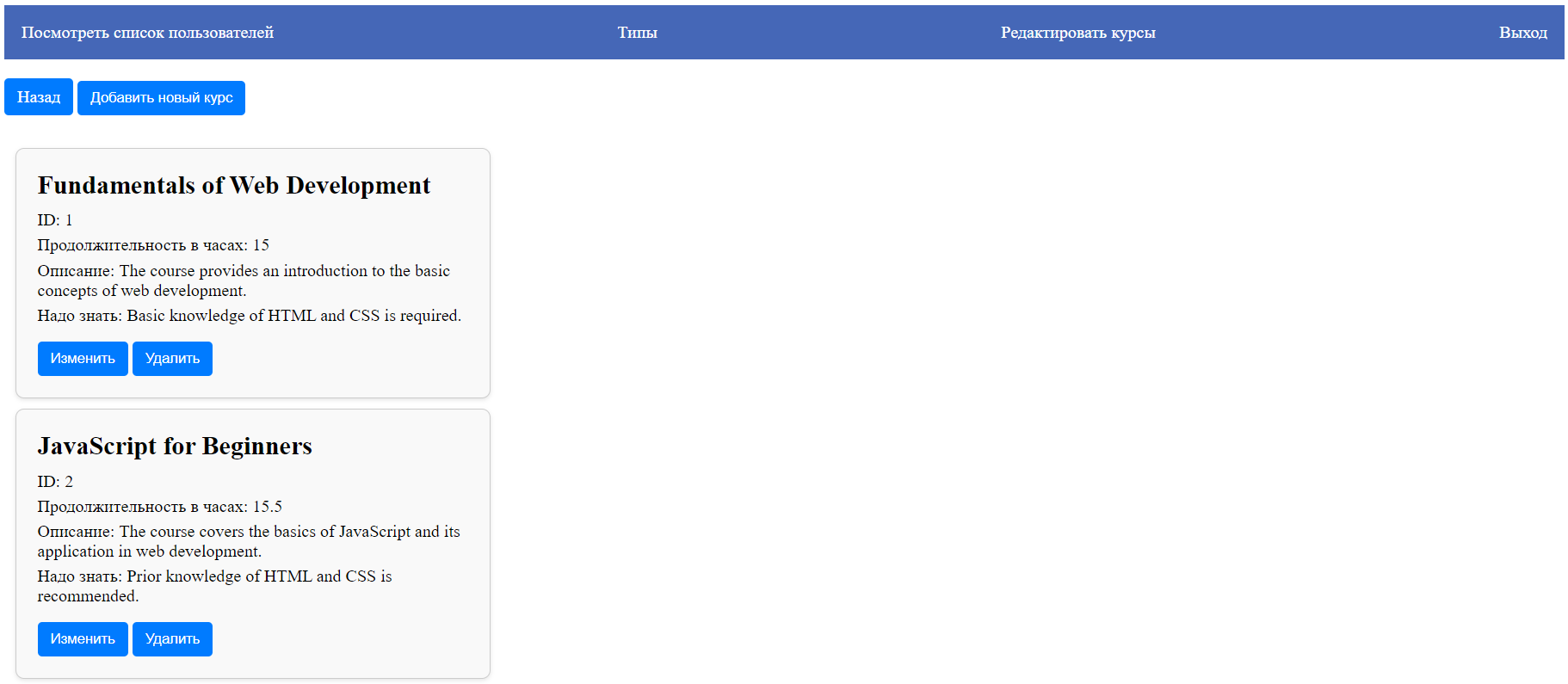


Рисунок 5.7 – Страница с пользователями платформы

Также в функционал администратора входит размещение новых курсов, до этого не представленных на платформе платформе. Предназначенный для этой цели интерфейс продемонстрирован на рисунке 5.8.

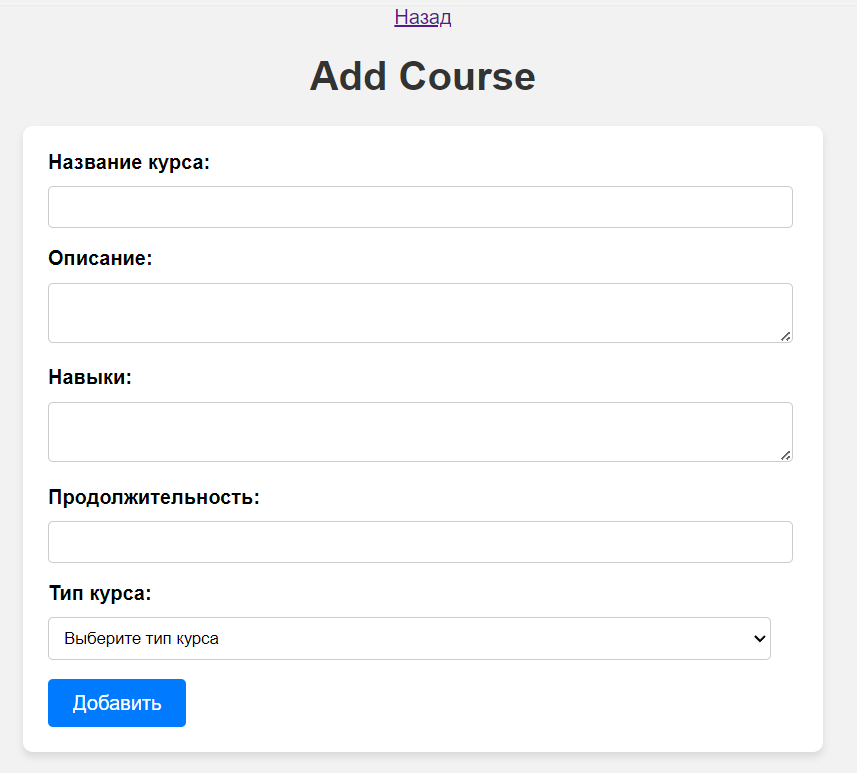


Рисунок 5.8 – Страница добавления курсов

Также администратор может редактировать курсы, которые уже были размещенные на платформе. На рисунке 5.9 приведён соответствующий интерфейс.

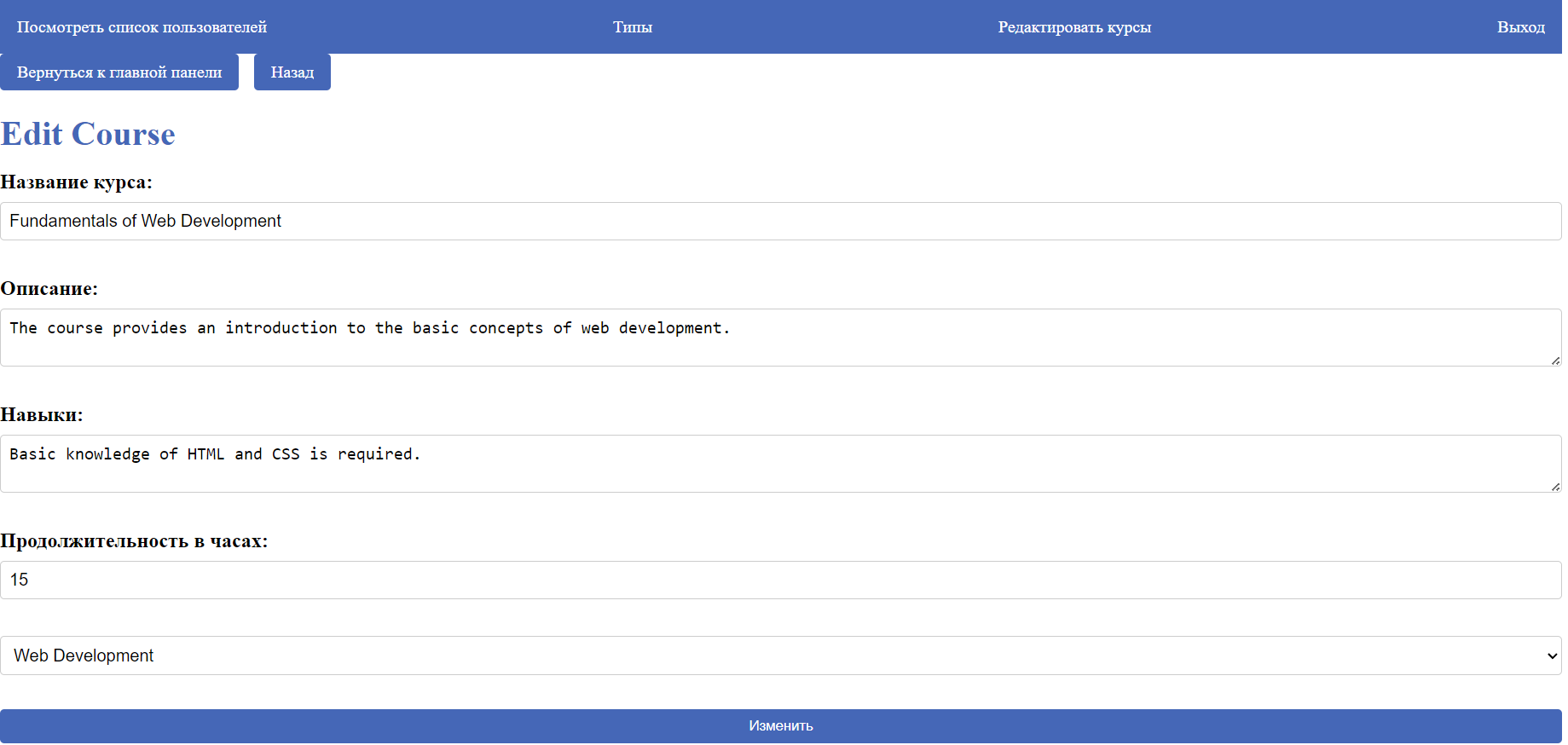


Рисунок 5.9 – Страница изменения курса

В качестве основного администраторского интерфейса для просмотра курсов размещённых на платформе, а также доступа к функционалу с их удалением и редактированием, используется интерфейс приведённый на рисунке 5.10.

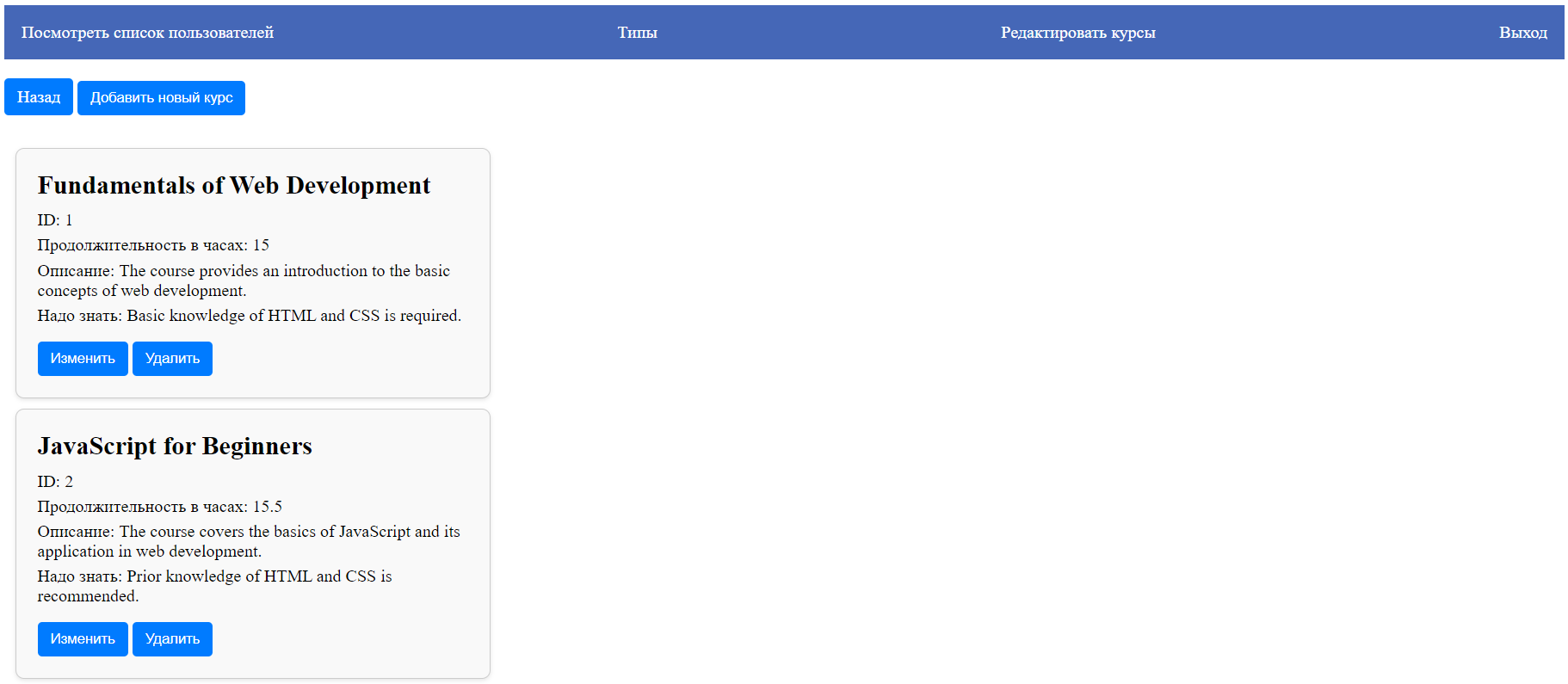


Рисунок 5.10 – Страница с курсами

Также в список функций администратора входит просмотр типов курсов, которые размещены на платформе, с дополнительной информацией в виде их идентификации в базе данных. Интерфейс используемый для этого представлен на рисунке 5.11.

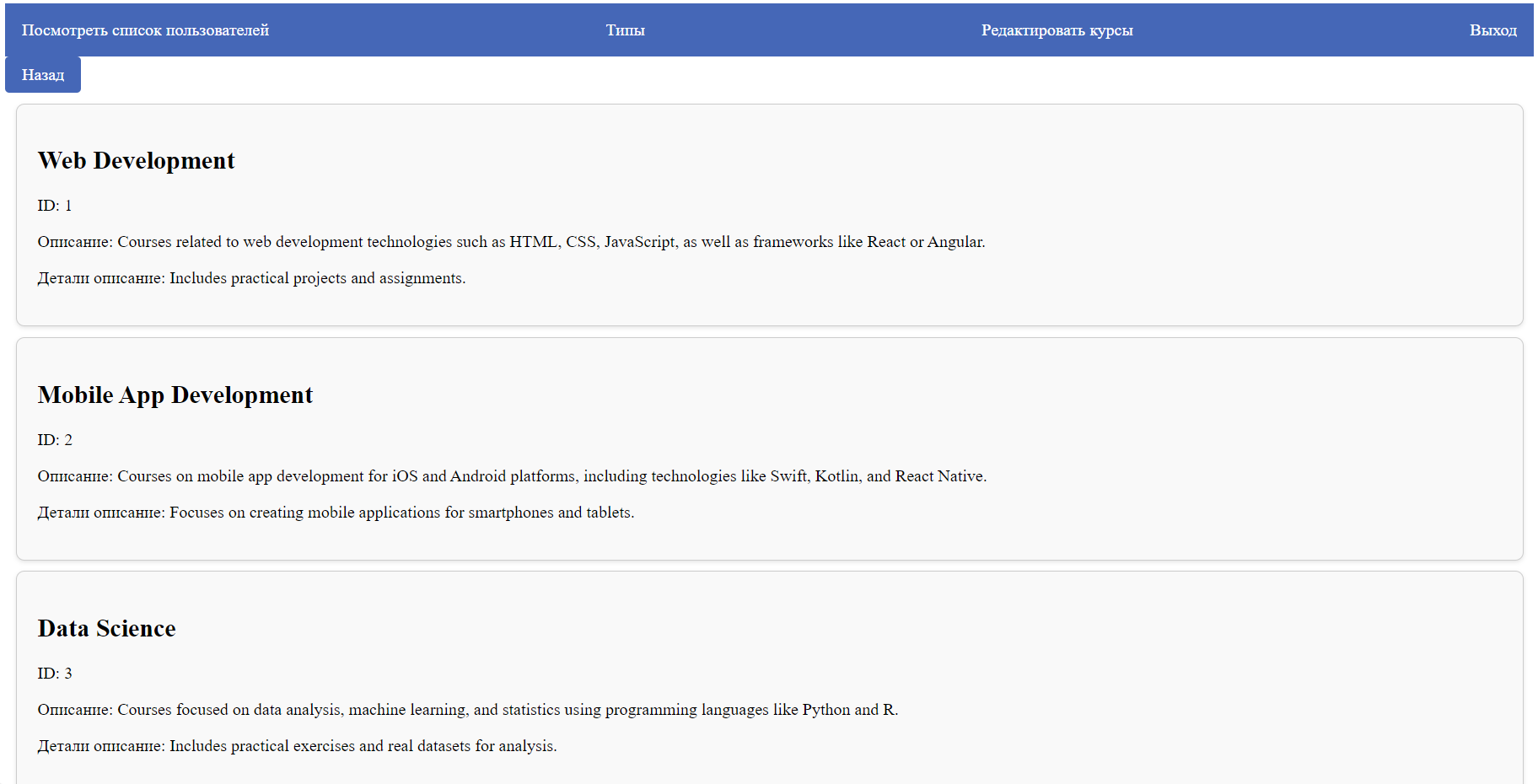


Рисунок 5.11 – Страница с типами курсов

Веб-приложение “Курсы по программированию” представляет собой мощный и удобный инструмент для изучения программирования. Оно разработано таким образом, чтобы обеспечить простоту использования и эффективность обучения.

Пользователи могут легко зарегистрироваться, введя свое имя пользователя, адрес электронной почты и пароль. После регистрации они получают доступ к широкому спектру курсов по программированию, каждый из которых содержит подробные уроки и практические задания.

Приложение также обеспечивает безопасность данных пользователей, используя современные методы шифрования и защиты. Пользователи могут быть уверены, что их личная информация и результаты обучения хранятся в безопасности.

Заключение

При выполнении курсового проекта было создано приложение поиска и предложения работы. Сервер был создан при помощи платформы Node.js, языка программирования JavaScript. Web-сайт был реализован при помощи express-handlebars. Файлы были разделены на layout и partials, то есть разделение повторяющихся блоков на каждой странице и индивидуальных страниц. База данных была реализована в СУБД MS SQL Server. Были реализованы все функциональные требования, а именно:

* обеспечение возможности регистрации и авторизации;
* поддержку ролей администратора и пользователя;
* просмотр доступных курсов пользователем;
* запись на конкретный курс пользователем;
* редактирование описания курса администратором;
* просмотр пользователем списка курсов на которые он записан;
* предоставление возможности поиска курса по параметрам;
* поддержка изменения названия курса администратором;
* добавление администратором новых курсов;
* удаление курсов администратором;
* прохождение курсов пользователем;
* просмотр пользователем пройденных курсов.

Также были реализованы WebSocket-сервер и поддержка протокола HTTPS.

В ходе разработки был уделен особый акцент на разработке backend приложения. Это обеспечивает быстрое и плавное взаимодействие пользователя с приложением, что в свою очередь улучшает общий пользовательский опыт.

Кроме того, был реализован интуитивно понятный и простой в использовании интерфейс. Это обеспечивает легкость навигации и доступности функций приложения для всех пользователей, независимо от их технического опыта или знаний.

Была проведена серия тестов для проверки устойчивости и надежности приложения в различных ситуациях. Результаты этих тестов показали, что приложение способно обрабатывать запросы в соответствии со всеми установленными требованиями.

В целом, результаты проекта демонстрируют успешное применение теоретических знаний на практике и подтверждают готовность приложения к реальной эксплуатации.

Список используемых источников

1. Node.js [Электронный ресурс] – Режим доступа: https://nodejs.org/docs/latest/api/ – Дата доступа: 30.02.2024.
2. Руководство по Node.js [Электронный ресурс] – Режим доступа: https://metanit.com/web/nodejs/ – Дата доступа: 25.03.2024
3. Handlebars.js [Электронный ресурс] – Режим доступа: https://handlebarsjs.com/guide/ – Дата доступа: 11.04.2024
4. SQL Server Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver16. – Дата доступа: 13.04.2024

Приложение А

Листинг определения моделей для ORM Sequelize

|  |
| --- |
| const { initModels } = require('../models/initModels');  const { DataTypes } = require('sequelize');  const { Sequelize } = require("sequelize");  const connection = new Sequelize('CourseWork', 'cw', '1111', {  host: 'localhost',  dialect: 'mssql',  port: 1433,  pool: {  min: 0,  max: 10  }  });  const models = initModels(connection);  module.exports = { models, connection };  // connection.sync({ alter: true,force: true }) // Use { force: true } to drop existing tables and re-create them  // .then(() => {  // console.log('Database synchronized successfully.');  // })  // .catch(err => {  // console.error('Unable to synchronize the database:', err);  // });  connection  .authenticate()  .then(() => {  console.log('Connection to the database has been established successfully.');  })  .catch(err => {  console.error('Unable to connect to the database:', err);  }); |
| module.exports = function(sequelize, DataTypes) {  return sequelize.define('Courses', {  course\_id: {  type: DataTypes.INTEGER,  primaryKey: true,  autoIncrement: true  },  course\_name: {  type: DataTypes.STRING(250),  allowNull: false  },  description: {  type: DataTypes.STRING(500)  },  duration: {  type: DataTypes.FLOAT,  allowNull: false  },  course\_type\_id: {  type: DataTypes.INTEGER,  allowNull: false  },  other\_details: {  type: DataTypes.STRING(1000)  }  }, {  sequelize,  tableName: 'Courses',  timestamps: false  });  };    function formatDate(timeString) {  const date = new Date(timeString);  const hours = String(date.getHours()).padStart(2, '0');  const minutes = String(date.getMinutes()).padStart(2, '0');  const seconds = String(date.getSeconds()).padStart(2, '0');  return `${hours}:${minutes}:${seconds}`;  } |
| module.exports = function(sequelize, DataTypes) {  return sequelize.define('UserStatistics ', {  statistic\_id: {  type: DataTypes.INTEGER,  primaryKey: true,  autoIncrement: true  },  user\_id: {  type: DataTypes.INTEGER,  allowNull: false  },  course\_id: {  type: DataTypes.INTEGER,  allowNull: false  },  status\_id: {  type: DataTypes.INTEGER,  allowNull: false  },  start\_date: {  type: DataTypes.DATE,  allowNull: false  },  end\_date: {  type: DataTypes.DATE,  allowNull: true  },  }, {  sequelize,  tableName: 'UserStatistics',  timestamps: false  });  }; |
| module.exports = function(sequelize, DataTypes) {  return sequelize.define('Status', {  status\_id: {  type: DataTypes.INTEGER,  primaryKey: true,  autoIncrement: true  },  status\_name: {  type: DataTypes.STRING(90), // пройден, в процессе, недоступен, доступен  allowNull: false  }  }, {  sequelize,  tableName: 'Status',  timestamps: false  });  }; |
| module.exports = function(sequelize, DataTypes) {  return sequelize.define('CourseTypes', {  type\_id: {  type: DataTypes.INTEGER,  primaryKey: true,  autoIncrement: true  },  type\_name: {  type: DataTypes.STRING(50),  allowNull: false  },  description: {  type: DataTypes.STRING(500)  },  other\_details: {  type: DataTypes.STRING(1000)  }  }, {  sequelize,  tableName: 'CourseTypes',  timestamps: false  });  }; |
| module.exports = function(sequelize, DataTypes) {  return sequelize.define('Users', {  ID: {  type: DataTypes.INTEGER,  primaryKey: true,  autoIncrement: true  },  Login: {  type: DataTypes.STRING(50),  allowNull: false  },  Role: {  type: DataTypes.INTEGER  },  Password: {  type: DataTypes.STRING,  allowNull: false  },  Email: {  type: DataTypes.STRING(50),  allowNull: false  },  SessionId: {  type: DataTypes.INTEGER, // Предполагается, что идентификатор сессии - строка  allowNull: true // Может быть пустым, если пользователь не авторизован  }  }, {  sequelize,  tableName: 'Users',  timestamps: false  });  }; |
| const users = require("./users");  const courses = require("./courses");  const status = require("./status");  const statistics = require("./statistics");  const courseTypes = require("./types");  const { DataTypes } = require('sequelize');  function initModels(sequelize) {  const Users = users(sequelize, DataTypes);  const Courses = courses(sequelize, DataTypes);  const Status = status(sequelize, DataTypes);  const Statistics = statistics(sequelize, DataTypes);  const CourseTypes = courseTypes(sequelize, DataTypes);  Statistics.belongsTo(Users, { foreignKey: 'user\_id', onDelete: 'CASCADE' });  Users.hasMany(Statistics, { foreignKey: 'user\_id', onDelete: 'CASCADE' });  Statistics.belongsTo(Courses, { foreignKey: 'course\_id', onDelete: 'CASCADE' });  Courses.hasMany(Statistics, { foreignKey: 'course\_id', onDelete: 'CASCADE' });  Statistics.belongsTo(Status, { foreignKey: 'status\_id', onDelete: 'CASCADE' });  Status.hasMany(Statistics, { foreignKey: 'status\_id', onDelete: 'CASCADE' });  // Связи для таблицы Courses и CourseTypes с каскадным удалением  Courses.belongsTo(CourseTypes, { foreignKey: 'course\_type\_id', onDelete: 'CASCADE' });  CourseTypes.hasMany(Courses, { foreignKey: 'course\_type\_id', onDelete: 'CASCADE' });  return {  Users,  Courses,  Status,  Statistics,  CourseTypes  };  }  module.exports = initModels;  module.exports.initModels = initModels;  module.exports.default = initModels; |