МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П.Королёва»

ИНСТИТУТ ИНФОРМАТИКИ И КИБЕРНЕТИКИ

Отчёт по проекту Easy Learning

Выполнила: Пироженкова И.С.

Группа: 6303-010302D

Easy Learning – инновационная образовательная платформа, позволяющая автоматизировать процесс выдачи домашних заданий ученикам

Цель проекта: Создать удобную веб-платформу, которая позволит репетиторам эффективно управлять домашними заданиями, группировать их в комплексы и раздавать ученикам в упорядоченной форме

Основные проблемы, которые решает платформа:

• Разрозненность заданий и отсутствие единого пространства для их хранения.
• Затраты времени на раздачу заданий.
• Необходимость гибкости в управлении учебными материалами.

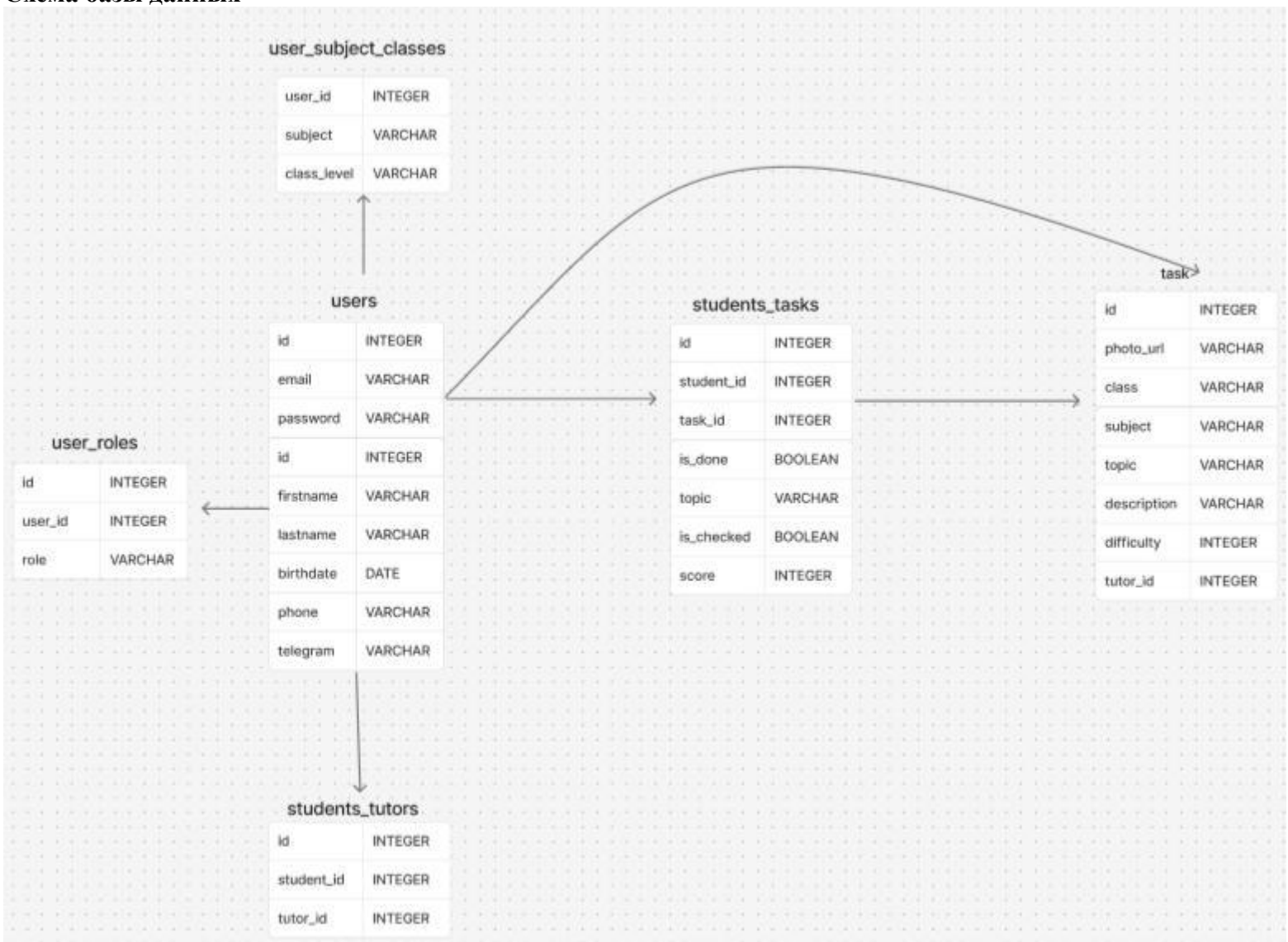
Функциональные требования

• Регистрация и авторизация пользователей (репетиторы, ученики).
• Панель управления для создания, редактирования и удаления заданий.
• Группировка заданий в домашние работы.
• Назначение заданий ученикам.

Лабораторная работа 0

• Разработана логическую схема базы данных
• Определена структура API
• Создан Git-репозиторий

Схема базы данных



Описание базы данных

Таблица users — общая информация о пользователе (и студенте, и тьюторе)

- **id** INTEGER – уникальный идентификатор пользователя
- **email** VARCHAR – электронная почта (логин), уникальное
- **password** VARCHAR – захешированный пароль
- **personalInfo.firstname** VARCHAR – имя
- **personalInfo.lastname** VARCHAR – фамилия
- **personalInfo.birthdate** DATE – дата рождения
- **personalInfo.phone** VARCHAR – номер телефона, уникальное
- **personalInfo.telegram** VARCHAR – ник в Telegram, уникальное

Таблица user_roles — роли пользователя

- **id** INTEGER – PK
- **user_id** INTEGER → users.id
- **role** VARCHAR – STUDENT или TUTOR

Таблица user_subject_classes — сочетания «предмет ↔ класс»

(реализовано как `@ElementCollection` в User.personalInfo)

- **user_id** INTEGER → users.id
- **subject** VARCHAR – ENUM из {MATH, PHYSICS, …, ENGLISH}
- **class_level** VARCHAR – ENUM из {CLASS_1…CLASS_11}

Таблица task — задания, создаваемые тьюторами

- **id** INTEGER – PK
- **photo_url** VARCHAR – ссылка на картинку с условием задачи
- **class** VARCHAR – ENUM ClassLevel (класс, для которого задача)
- **subject** VARCHAR – ENUM Subject
- **topic** VARCHAR – краткий заголовок/тема задачи
- **description** VARCHAR – подробное описание
- **difficulty** INTEGER – сложность (1–10 и т. п.)
- **tutor_id** INTEGER → users.id

Таблица students_tasks — статус выполнения задания учеником

- **id** INTEGER – PK
- **student_id** INTEGER → users.id
- **task_id** INTEGER → task.id
- **is_done** BOOLEAN – пометил ли ученик как «сделал»
- **is_checked** BOOLEAN – проверил ли тьютор
- **score** INTEGER – выставленный балл

Таблица students_tutors — связь «ученик ↔ тьютор»

- **id** INTEGER – PK
- **student_id** INTEGER → users.id (роль STUDENT)
- **tutor_id** INTEGER → users.id (роль TUTOR)

Кардинальности и типы связей

1. **`users` ↔ `user_roles`**

- Тип: **1-к-многим**
- Один пользователь (`users.id`) может иметь несколько записей в `user_roles` (несколько ролей), но каждая запись в `user_roles` принадлежит ровно одному пользователю.

2. **`users` ↔ `user_subject_classes`**

◦ Тип: **1-к-многим**
◦ Один пользователь хранит множество сочетаний «предмет ↔ класс» (`user_subject_classes`), каждое из которых связано с одним `users.id`.

3. **`users` (Tutor) ↔ `task`**

◦ Тип: **1-к-многим**
◦ Один тьютор (`users.id` с ролью TUTOR) может создавать много задач (`task.tutor_id`), но у каждой задачи ровно один автор-тьютор.

4. **`task` ↔ `students_tasks`**

◦ Тип: **1-к-многим**
◦ Одна задача (`task.id`) может встречаться во многих записях `students_tasks` (разные ученики), но каждая запись `students_tasks` ссылается на одну задачу.

5. **`users` (Student) ↔ `students_tasks`**

◦ Тип: **1-к-многим**
◦ Один ученик (`users.id` с ролью STUDENT) может иметь много записей в `students_tasks` (для разных задач), но каждая запись относится к одному ученику.

6. **`students_tasks`**

◦ Фактически реализует **отношение многие-ко-многим** между **учениками** и **задачами**, обогащённое дополнительными атрибутами (`is_done`, `score` и т. д.).

7. **users (Student) ↔ students_tutors**

◦ Тип: **1-к-многим**
◦ Один ученик (`users.id` с ролью STUDENT) может быть связан сразу с несколькими репетиторами через `students_tutors.student_id`.
8. **users (Tutor) ↔ students_tutors**

◦ Тип: **1-к-многим**
◦ Один репетитор (`users.id` с ролью TUTOR) может вести много учеников через `students_tutors.tutor_id`.

9. **students_tutors**

◦ Фактически реализует **отношение многие-ко-многим** между **учениками** и **тьюторами**.

Лабораторная работа 1

• Развернута MySQL в Docker
• Разработаны ORM-модели с использованием Hibernate, настроены миграции
• Настроено хеширование паролей
• Написаны скрипты для заполнения базы данных тестовыми данными
• Реализован функционал для работы с данными в соответствии с тематикой приложения (сервисы, контроллеры, репозитории)

Разработка ORM-моделей

• Для взаимодействия с базой данных были разработаны ORM-модели с помощью Hibernate.
  • Определены основные сущности:

1 **. User** (`users`)

- PK: `id`
- Поля: `email`, `password`, `personalInfo`, коллекция `roles` (через `user_roles`), коллекция `subjectClassPairs` (через `user_subject_classes`), связи на созданные задачи (`tasks`), домашки (`studentsTasks`) и связи ученик–тьютор (`tutors`, `students`).

2. **Role** (`user_roles`)

◦ PK: `id`
◦ FK: `user_id` → `users.id`
◦ Поле: `role` (STUDENT / TUTOR)

3. **SubjectClassPair** (`user_subject_classes`)

◦ PK (составной): `user_id` + `subject` + `class_level`

4. **Task** (`task`)

◦ PK: `id`
◦ FK: `tutor_id` → `users.id`
◦ Поля: `photo_url`, `class_level`, `subject`, `topic`, `description`, `difficulty`

5. **StudentsTasks** (`students_tasks`)

◦ PK: `id`
◦ FK: `student_id` → `users.id`
◦ FK: `task_id` → `task.id`
◦ Поля: `is_done`, `is_checked`, `score`

6. **StudentsTutors** (`students_tutors`)

◦ PK: `id`
◦ FK: `student_id` → `users.id`
◦ FK: `tutor_id` → `users.id`

• Между сущностями настроены связи @OneToMany, @ManyToOne, @ManyToMany, обеспечивающие корректное отображение отношений.

Было реализовано:

• Для безопасности реализовано хеширование паролей с помощью bcrypt – популярного алгоритма, обеспечивающего защиту от атак на пароли.
• Проект создан с помощью Spring Initializr, на базе Spring Boot.
• Для управления структурой базы данных используется Liquibase – инструмент для миграций.

Были разработаны:

• **Контроллеры**:

◦ `AuthController`
◦ `ViewController`

• **Сервисы** (интерфейсы + реализации в `service/impl`):

- `AuthService` → `AuthServiceImpl`
- `TaskService` → `TaskServiceImpl`
- `StudentsTutorsService` → `StudentsTutorsServiceImpl`
- `UserService` → `UserServiceImpl`

• **JPA-репозитории**:

◦ `UserRepository`
◦ `TaskRepository`
◦ `StudentsTasksRepository`
◦ `StudentsTutorsRepository`

Лабораторная работа 2

• Разработаны CRUD-методы для работы с моделями
• Настроены маршруты и обработка запросов
• Для тестирования API использовался Postman (для проверки запросов)

Структура API

Students

GET /api/students/{id}

Получить профиль своего студента (или свои данные). Responses: 200 OK + StudentRDto или StudentNRDto 401 Unauthorized / 403 Forbidden

GET /api/students

Список всех студентов (только для репетиторов). Responses: 200 OK + [StudentNRDto, …] 401 Unauthorized / 403 Forbidden

POST /api/students

Создать нового студента (только репетитор). Responses: 201 Created + StudentNRDto 401 Unauthorized / 403 Forbidden

PUT /api/students/{id}

Обновить данные студента (только репетитор). Request body: StudentRDto Responses: 200 OK + StudentRDto 401 Unauthorized / 403 Forbidden

DELETE /api/students/{id}

Удалить студента. Responses: 204 No Content

Tutors

GET /api/tutor/{id}
Получить профиль тьютора (или свои данные). Responses: 200 OK + TutorRDto или TutorNRDto 401 Unauthorized / 403 Forbidden

GET /api/tutor

Список всех тьюторов (только для репетиторов). Responses: 200 OK + [TutorNRDto, …]

POST /api/tutor

Создать нового тьютора. Request body: TutorNRDto Responses: 201 Created + TutorNRDto

PUT /api/tutor/{id}

Обновить данные тьютора. Request body: TutorRDto Responses: 200 OK + TutorRDto

DELETE /api/tutor/{id}

Удалить тьютора. Responses: 204 No Content

GET /api/tutor/students

Получить список своих студентов. Responses: 200 OK + [StudentNRDto, …]

POST /api/tutor/students/add?studentId={id}

Привязать существующего студента. Responses: 200 OK

Homeworks & Tasks

GET /api/students/{id}/homeworks

Список домашних заданий студента (только свой). Responses: 200 OK + [HomeworkDTO, …] 401 Unauthorized / 403 Forbidden

POST /api/homeworks

Создание домашнего задания. Request body: HomeworkNRDto Responses: 201 Created + HomeworkNRDto

GET /api/homeworks/{id}

Получить домашнее задание по ID. Responses: 200 OK + HomeworkRDto 404 Not Found

PUT /api/homeworks/{id}

Обновить домашнее задание. Request body: HomeworkRDto Responses: 200 OK

DELETE /api/homeworks/{id}

Удалить домашнее задание. Responses: 204 No Content

Лабораторная работа 3

Сделано:

• Регистрация нового пользователя
• Вход в систему и получение JWT-токена
• Проверка валидности токена
• Ограничение доступа к определённым эндпоинтам

Проверка работы аутентификации через Postman

POST /api/v1/auth/login
Вход в систему и получение JWT. Request body (JSON):{ "username": "user@example.com", "password": "secret} Responses: 200 OK + { id, username, accessToken, refreshToken } 401 Unauthorized при неверных учётных данных

POST /api/v1/auth/register/{user-type}

Регистрация нового пользователя. Request body (JSON):{ "email": "newuser@example.com", "password": "password123"} Responses: 200 OK + { email, password, userType } 400 Bad Request при неверном user-type

POST /api/v1/auth/refresh

Обновление accessToken по refresh-токену. Request body: строка с refresh-токеном Responses: 200 OK + { id, username, accessToken, refreshToken } 400 Bad Request / 401 Unauthorized при невалидном токене

Описание модели пользователя и JWT-аутентификации

JWT-система:

При логине или регистрации выдаются два токена: accessToken и refreshToken. accessToken — короткоживущий, используется в заголовке Authorization. refreshToken — долгоживущий, используется для обновления accessToken.

Принцип работы:

Пользователь логинится → получает токены. accessToken передаётся в каждый запрос, где требуется авторизация. JwtTokenFilter извлекает токен и подставляет в SecurityContext.
Контроллеры через SecurityContextHolder знают, кто сделал запрос.

Безопасность: Студенты могут видеть только свои данные.

Модель пользователя:

Student и Tutor имеют email, password (хранится в виде bcrypt-хэша), личную информацию и связи (задания, репетиторы). Для авторизации используются кастомные реализации UserDetails (StudentJwtEntity, TutorJwtEntity).

Лабораторная работа 4

Разработка пользовательского интерфейса

– **Шаблонизатор:** Thymeleaf (Spring)

Layout: `layout.html` описывает общую структуру (header, navbar, footer, подключение JS/CSS) и вставляет в `<main>` фрагменты других страниц через

Основные страницы (templates/*.html)

Главная (index.html)

– Для STUDENT: выводит таблицу «Мои задачи» из ${myTasks} – Для TUTOR: приветственный баннер – Навигация в header с sec:authorize по ролям

Логин (login.html)

• Располагается внутри общего `layout.html` через `layout:fragment="content"`.

• Содержит форму
• После отправки делает POST на /frontend/login, Spring Security проверяет учетные данные.
• Показывает ссылку на страницу регистрации

Регистрация (register.html)

• Также вставляется в layout.html через layout:fragment="content".
• Содержит форму
• POST на /frontend/register, контроллер определяет роль по чекбоксу и создаёт пользователя.
• Снизу ссылка на страницу логина

Личный кабинет (profile.html)

– Форма редактирования ProfileDto (email, пароль, личная информация) – Отображение flash-сообщения об успешном сохранении

Мои студенты (students.html)

– Таблица связей ${students} – Форма добавления по ID:

Мои репетиторы (tutors.html)

– Кликинговая таблица ${tutors} с переходом на /frontend/tutors/{id}

Список задач (tasks.html)

– Группировка по классу/предмету из ${groupedTasks} – Сортировка через ссылки с sort – Кнопка «Назначить» внутри таблицы

Просмотр задачи (task_view.html)

– Вывод полей задачи и изображения ${task.photoUrl}

Создание задачи (task_new.html)

– Форма с th:object="${taskForm}", file-upload и выпадающими списками ${classLevels} и ${subjects}

Все шаблоны подключают общую навигацию через sec:authorize (Spring Security dialect) и используют layout fragments.

Настройка взаимодействия с сервером

• Все страницы построены на серверном рендеринге Thymeleaf и отправляются из `ViewController`.
• Формы (`<form>`) отправляют данные на URL-эндпойнты контроллеров через POST/GET.
• Данные из контроллеров (`Model.addAttribute`) автоматически подставляются в шаблоны через `${…}`.

Работа с аутентификацией

• Spring Security + JWT: accessToken хранится в HTTP-only cookie, Spring автоматически валидирует токен через `JwtTokenFilter`.
• В шаблонах используется атрибут `sec:authorize="hasRole('…')"`, чтобы показывать или скрывать блоки контента в зависимости от роли пользователя.

Тестирование интерфейса

• Ручная проверка в браузере и Postman:
◦ Логин/регистрация → редиректы на фронтенд-страницы.
◦ Переход между разделами (мои задачи, профиль, список студентов и т. д.).

Лабораторная работа 5

Настройка базы данных в Docker

• Создан Docker-контейнер с MySQL:
• В файле docker-compose.yml добавлен сервис mysql, использующий официальный образ MySQL 8.0.

• Для хранения данных используется volume (mysql_data).
• Настроено подключение к базе данных через переменные окружения:
• В секции environment для MySQL заданы переменные MYSQL_ROOT_PASSWORD и MYSQL_DATABASE.
• В настройках Spring (application.yml) прописан адрес: jdbc:mysql://mysql:3306/easy_learning, где mysql — имя контейнера-сервиса из Docker Compose.

Упаковка в Docker
• Написан Dockerfile:
• В корне проекта создан Dockerfile, в котором реализована сборка jar-файла с помощью Gradle и упаковка приложения в лёгкий образ на базе eclipse-temurin:17-jre.
• Создан файл docker-compose.yml для управления сервисами:
• В файле описаны сервисы app и mysql (база данных), а также volumes для хранения данных и загрузок.

Скриншоты работающего приложения в контейнерах:

Настроен .dockerignore для исключения ненужных файлов:

В корне проекта создан файл .dockerignore, в который добавлены каталоги и файлы, не требующиеся для сборки контейнера (.git, .idea, /build, .gradle и т.д.), что ускоряет и оптимизирует процесс сборки.

Описание структуры контейнеризации и настройки окружения

• Проект контейнеризован с помощью Docker:
• Backend и Frontend собирается и запускается через Dockerfile.
• База данных MySQL разворачивается отдельным контейнером.
• Управление всеми сервисами — через docker-compose.yml.
• Настройки окружения (пароли, адреса БД) передаются через .env и переменные окружения, используемые в docker-compose.yml и application.yml.
• Для оптимизации сборки создан .dockerignore, исключающий лишние файлы из образа.

Тестирование работы контейнеров

Запуск всех контейнеров осуществляется с помощью docker compose up: Все сервисы поднимаются одной командой docker compose up --build, обеспечивая автоматическую сборку и запуск backend-приложения и базы данных.

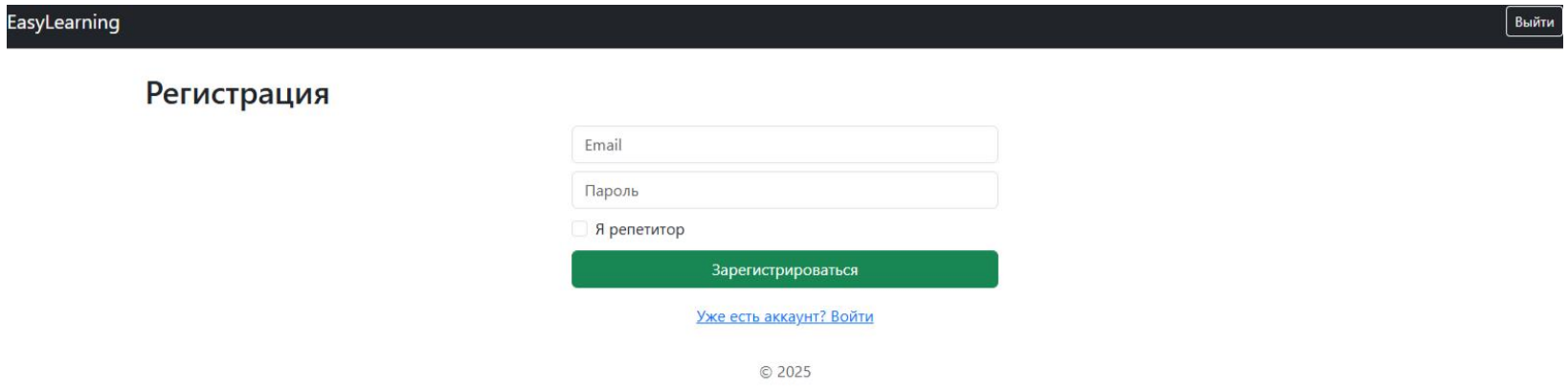
Проверка доступности API и фронтенда:

После запуска контейнеров проверялась работоспособность приложения по адресу http://localhost:8080/, а также осуществлялся доступ к базе данных. Также в логах приложения проверялись корректность запуска и соединения с базой данных.

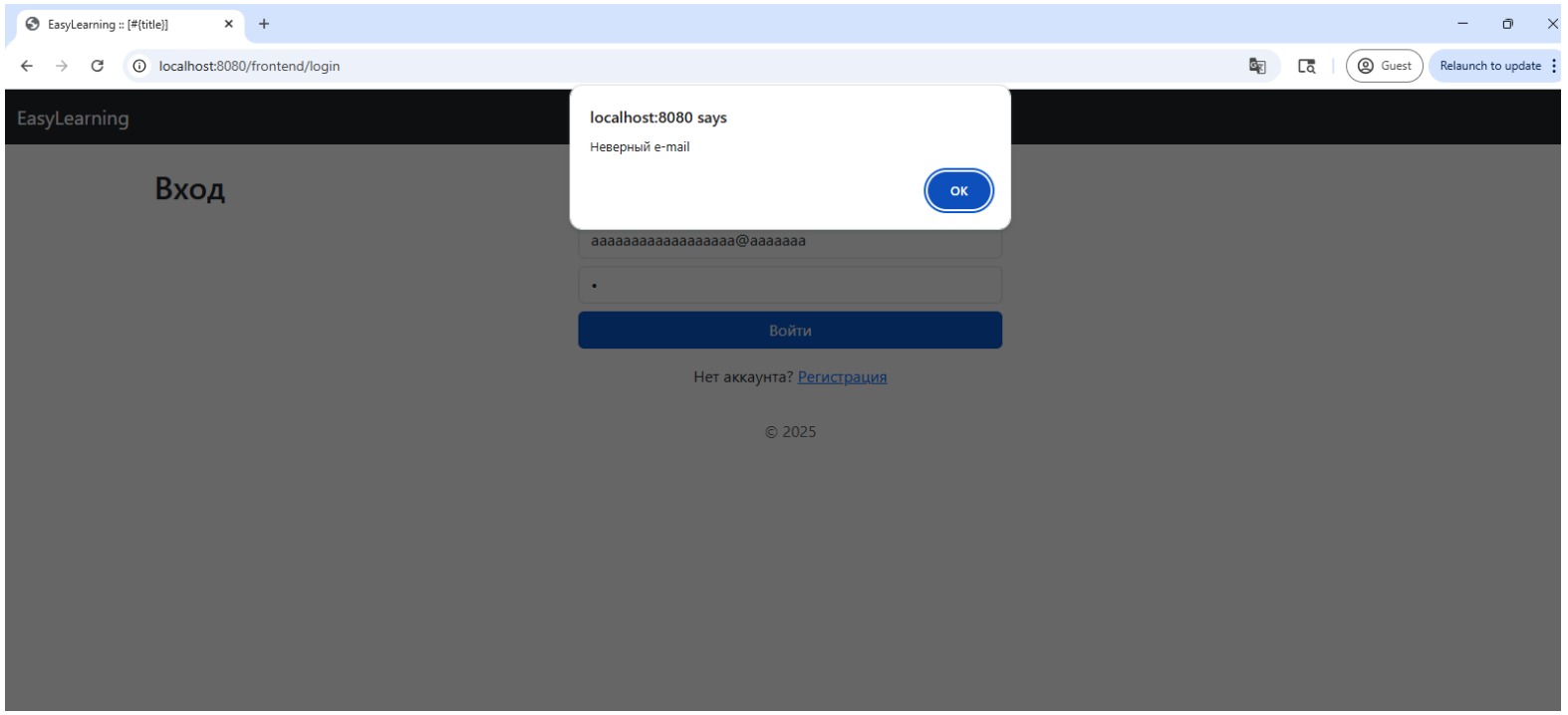
В итоге, вся серверная часть, база данных и сопутствующая инфраструктура теперь полностью автоматизированы и разворачиваются в изолированной среде с помощью Docker, что упрощает деплой и тестирование.

Работающее веб приложение:

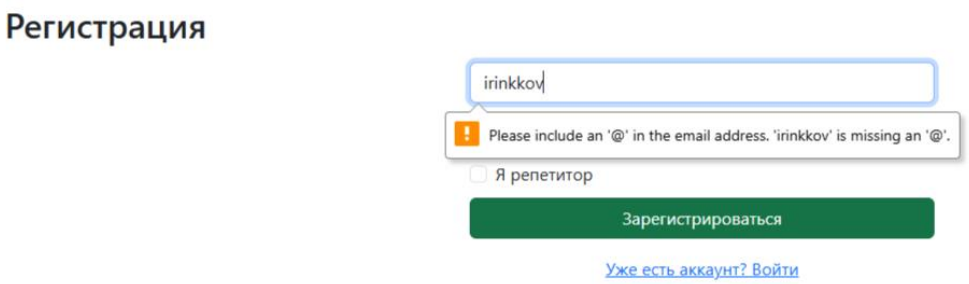
1. Страница регистрации



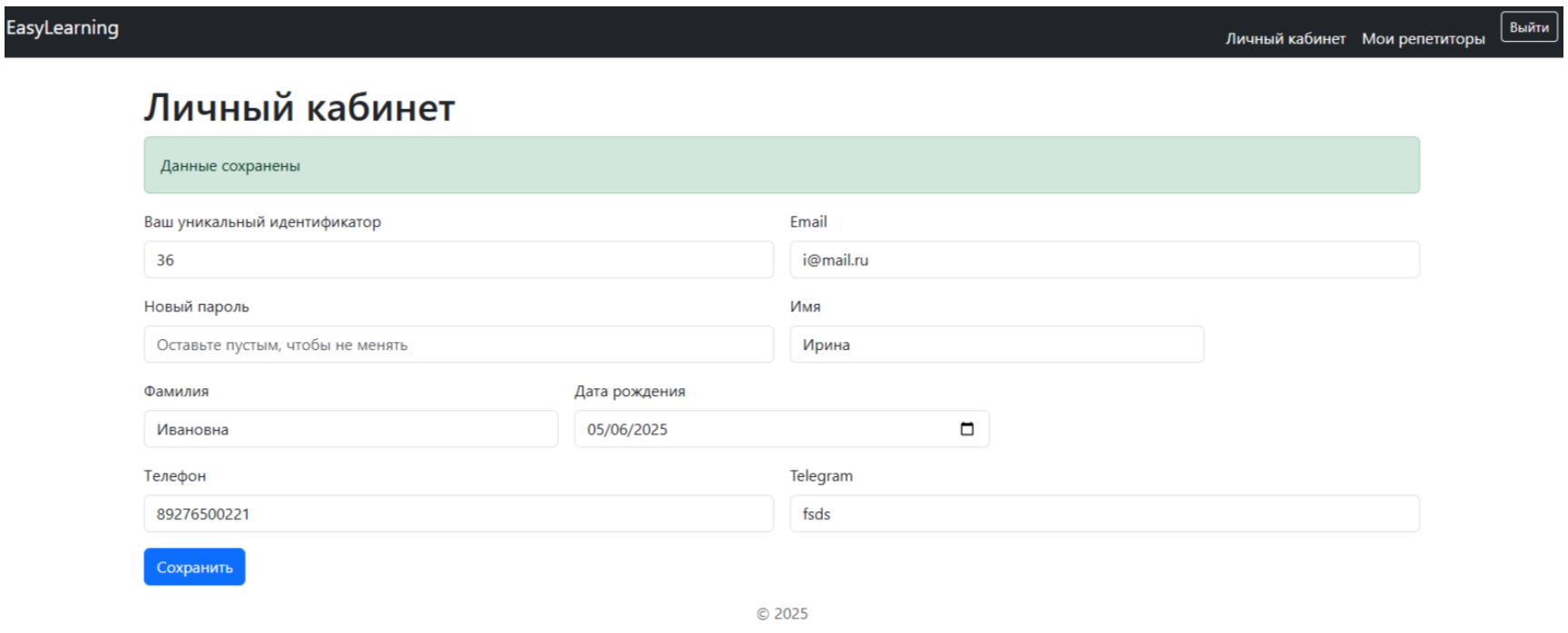
2. Страница входа. Если введён неверный email. Такая же проверка сделана на неверный пароль.



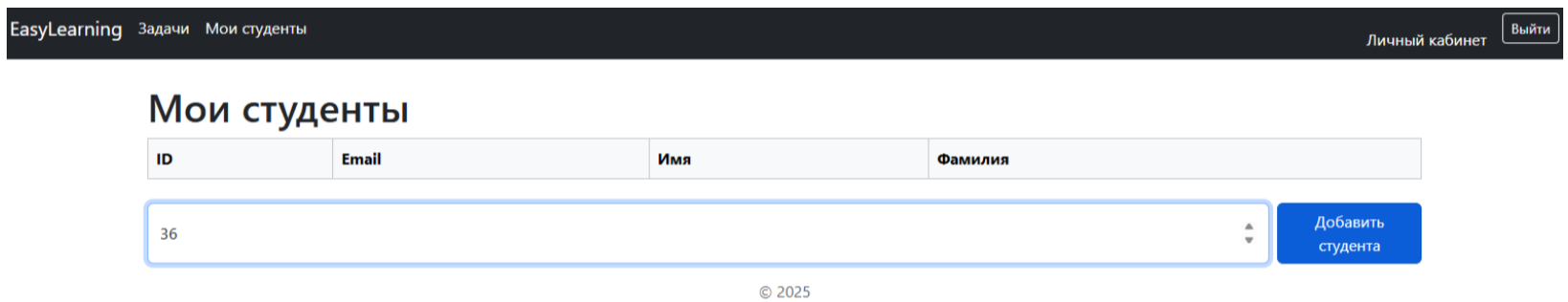
3. Проверка. На некорректный email



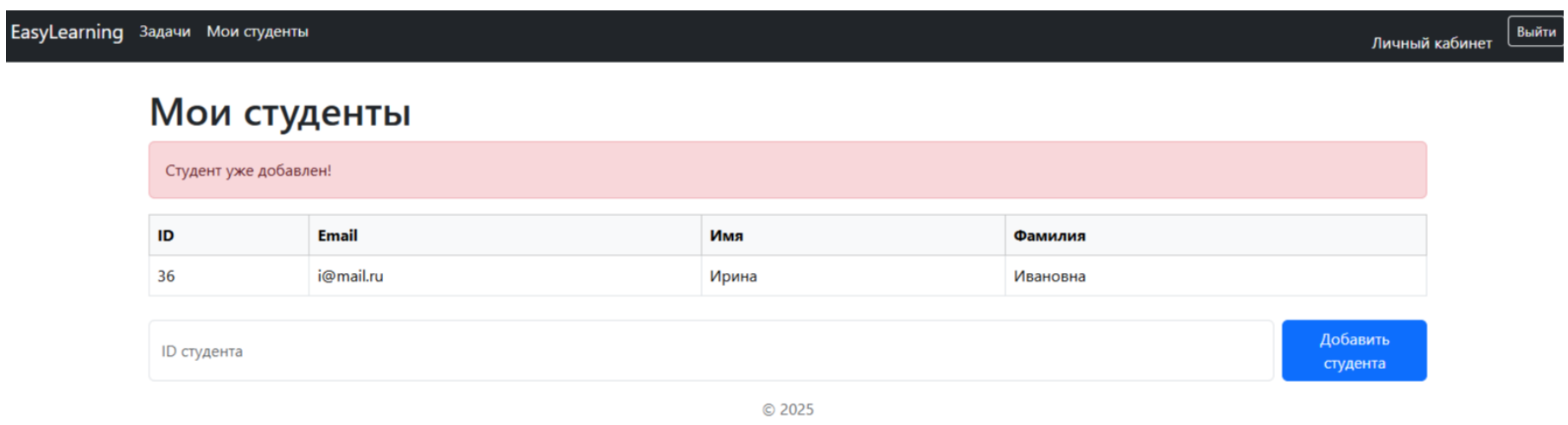
4. Личный кабинет. Пароль можно изменить. Email уникальный, менять нельзя. Все остальные поля спокойно меняются. Страницы одинаковые и у репета, и у студента.



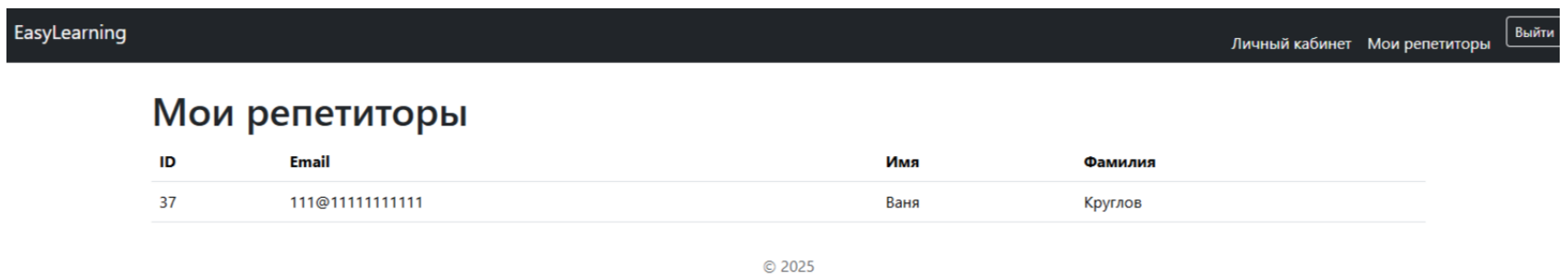
5. Добавление студента на странице репетитора. По id (указан в личном кабинете)



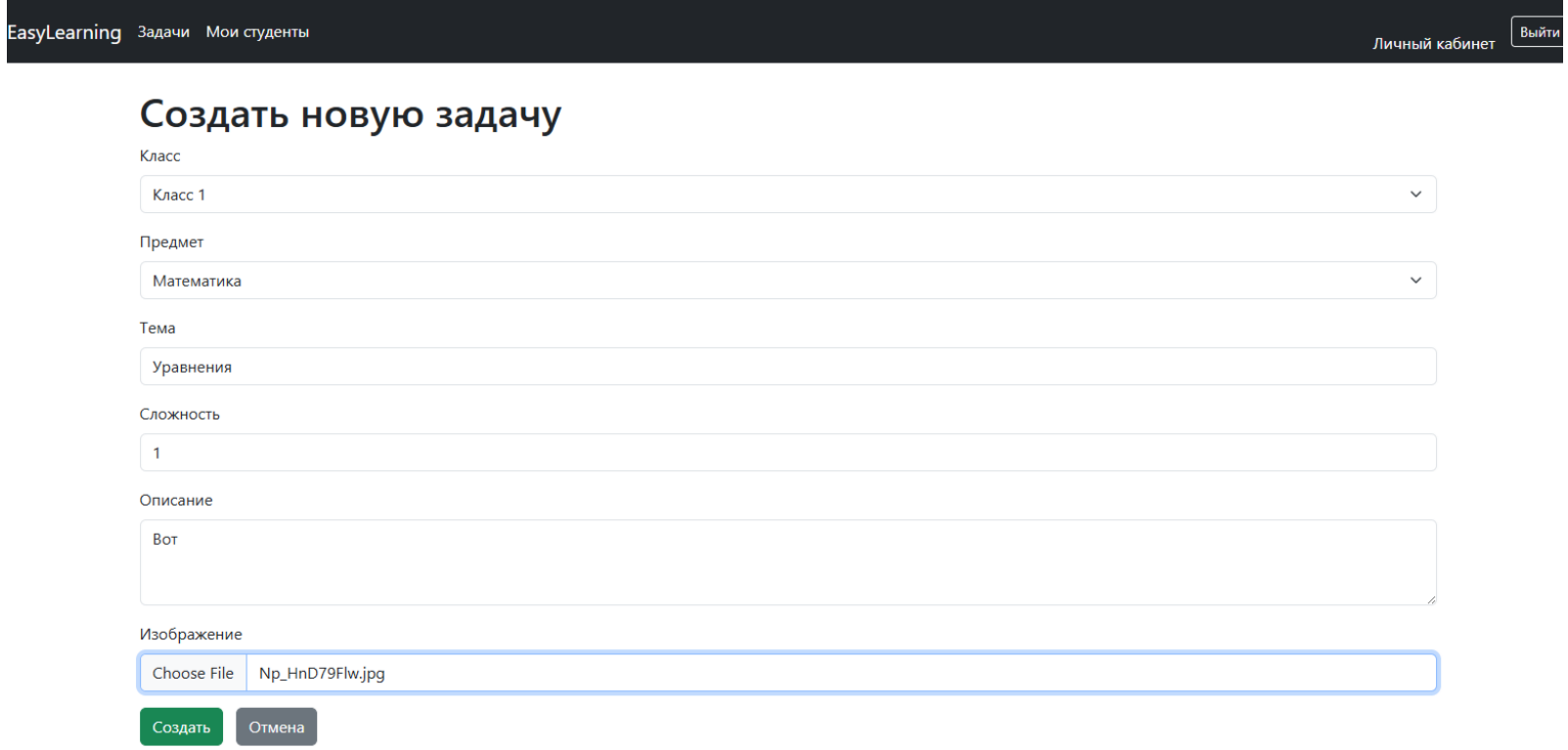
6. Проверка на добавление одного и того же студента. Также есть проверка на добавление несуществующего студента.



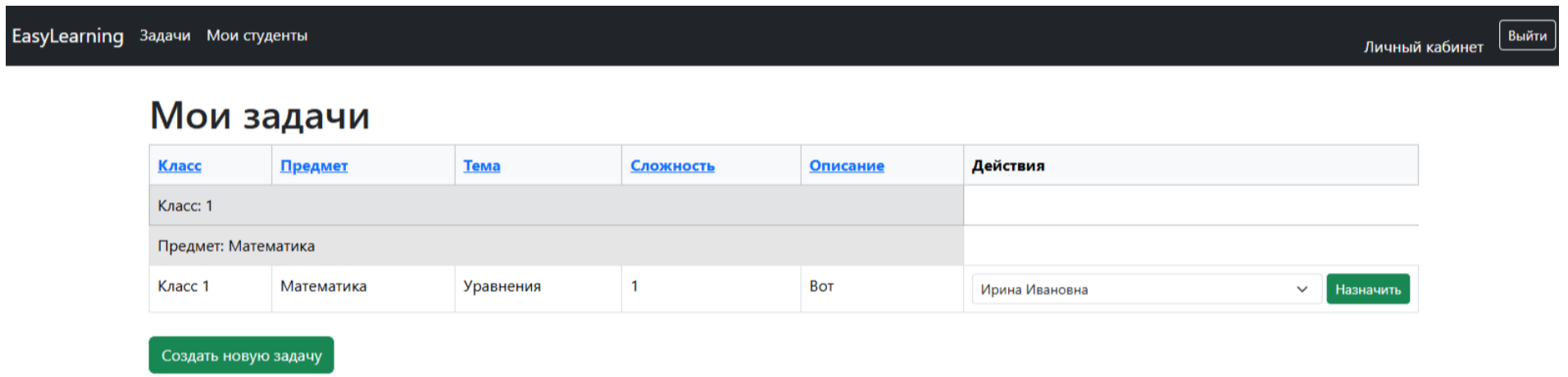
7. У студента на странице появляется его репетитор.



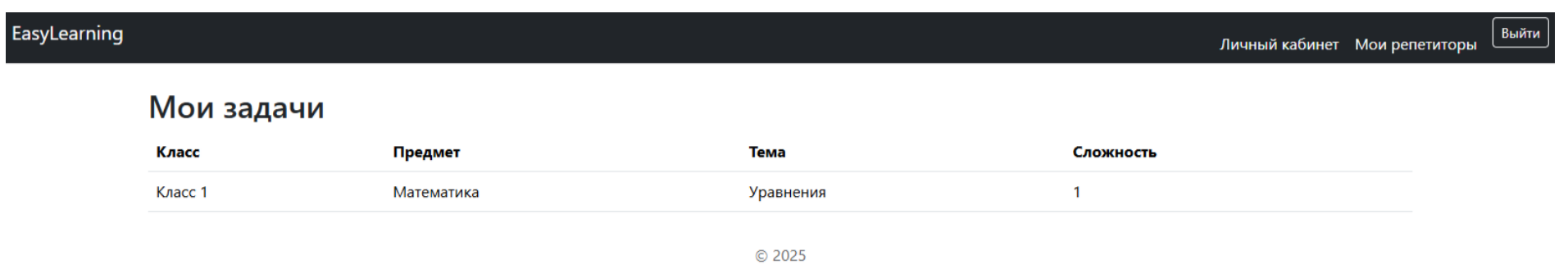
8. Страница репетитора с добавлением новой задачи для студента. Обязательно прикрепление фото задачи.



9. Отображение всех созданных задач. Сбоку у каждой задачи выбираем своих студентов, кому дать задачу. Есть фильтрация по классу, предмету, теме, сложности



10. Отображение у ученика его задачи от репетитора. Возможен просмотр созданной задачи с отображением фото, прикреплённого к задаче



11. Кнопки выход работают. Перебрасывают на станицу входа.