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

Лабораторная работа 0

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.

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

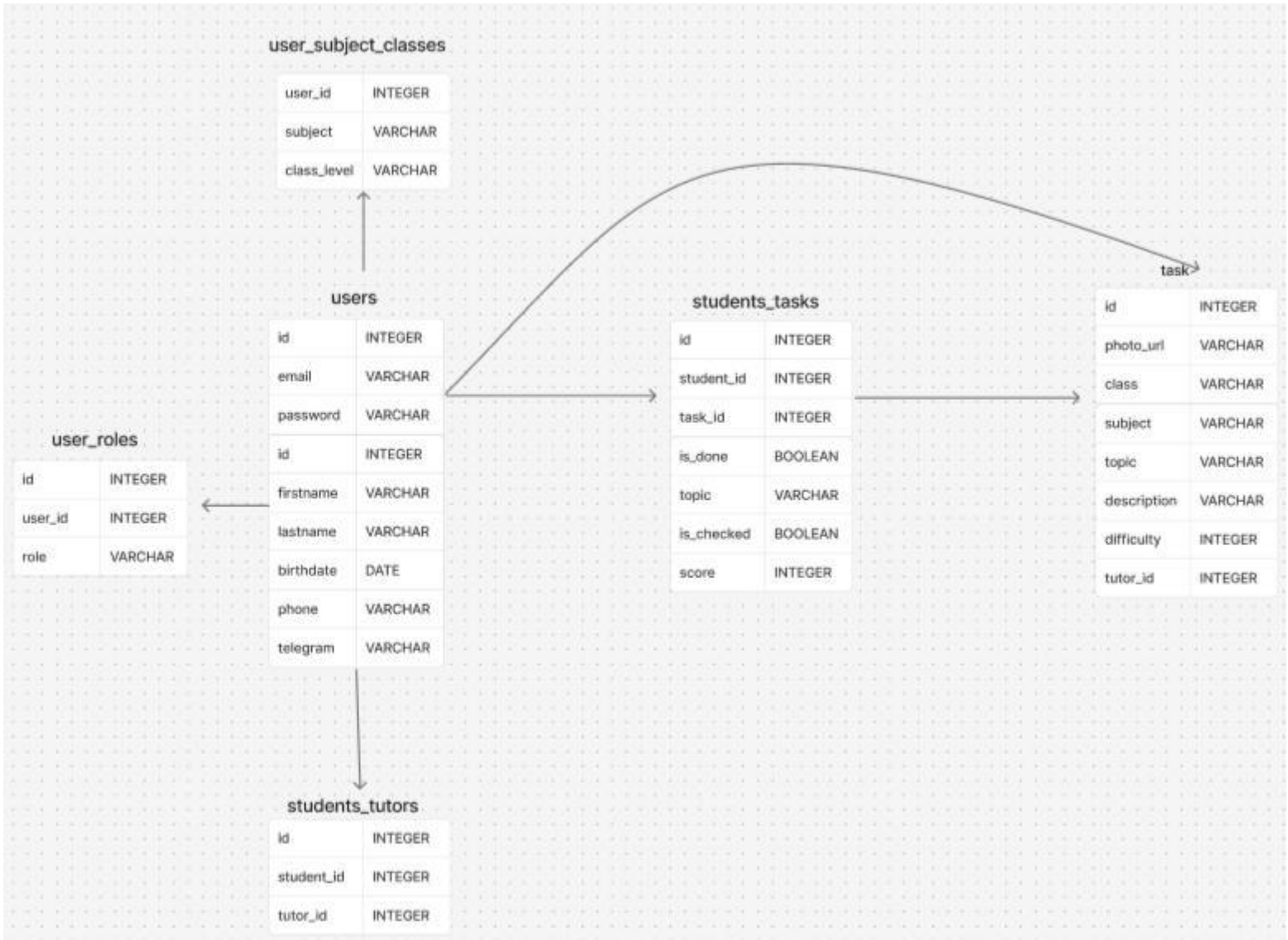

Рисунок 1 – Схема базы данных

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.

- **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`).

```java
@Getter
@Setter
@NoArgsConstructor
public class User {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;
  @Column(nullable = false, unique = true)
  private String email;
  @Column(nullable = false)
  private String password;
  @ElementCollection(fetch = FetchType.EAGER)
  @CollectionTable(name = "user_roles", joinColumns = @JoinColumn(name = "user_id"))
  @Enumerated(EnumType.STRING)
  @Column(name = "role")
  private Set<Role> roles = new HashSet<>();
  // Встраиваемая персональная информация
  @Embedded
  private PersonalInfo personalInfo;
  // Задания, созданные репетитором
  @OneToMany(mappedBy = "tutor", cascade = CascadeType.ALL, orphanRemoval = true)
  private Set<Task> tasks = new HashSet<>();
  // Связи ученик-домзадание
  @OneToMany(mappedBy = "student", cascade = CascadeType.ALL, orphanRemoval = true)
  private Set<StudentsTasks> studentsTasks = new HashSet<>();
  // Связи ученик-тьютор (роль STUDENT)
  @OneToMany(mappedBy = "student", cascade = CascadeType.ALL, orphanRemoval = true)
  private Set<StudentsTutors> tutors = new HashSet<>();

  @OneToMany(mappedBy = "tutor", cascade = CascadeType.ALL, orphanRemoval = true)
  private Set<StudentsTutors> students = new HashSet<>();
}
```

Рисунок 2 – ORM модель User

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

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

Рисунок 3 – ORM модель Role

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

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

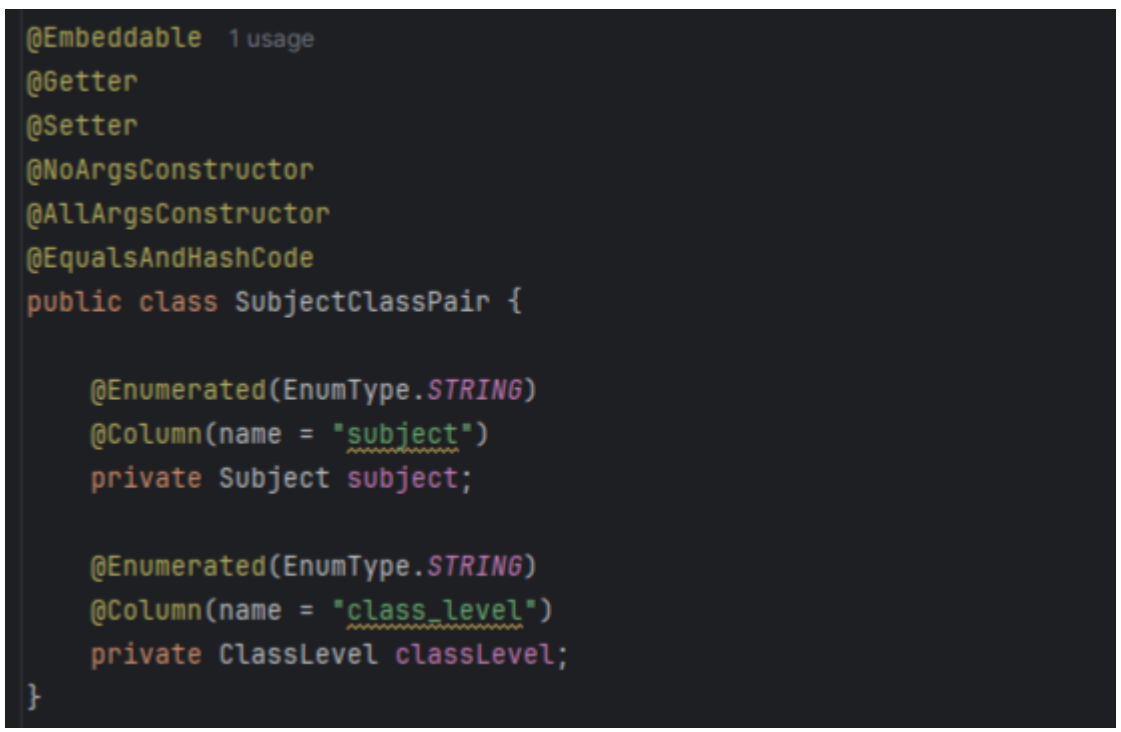

Рисунок 4 – ORM модель Role

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

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

```java
@Entity
@Table(name = "task")
@Getter
@Setter
@NoArgsConstructor
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "photo_url", nullable = false)
    private String photoUrl;
    @Enumerated(EnumType.STRING)
    @Column(name = "class", nullable = false)
    private ClassLevel className;
    @Enumerated(EnumType.STRING)
    @Column(name = "subject", nullable = false)
    private Subject subject;
    @Column(name = "topic", nullable = false)
    private String topic;
    @Column(name = "description", nullable = false)
    private String description;
    @Column(name = "difficulty", nullable = false)
    private Integer difficulty;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "tutor_id")
    private User tutor;

    @OneToMany(mappedBy = "task", fetch = FetchType.LAZY, cascade = CascadeType.ALL, orphanRemoval = true)
    private Set<StudentsTasks> studentsTasks = new HashSet<>();
}
```

Рисунок 5 – ORM модель Task

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

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

```java
@Entity
@Table(name = "students_tasks")
@Getter
@Setter
@NoArgsConstructor
public class StudentsTasks {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "student_id")
    private User student;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "task_id")
    private Task task;
    private Boolean isDone;
    private Boolean isChecked;
    private Integer score;
    public void setStudent(User student) {
        this.student = student;
        if (student != null && student.getStudentsTasks() != null) student.getStudentsTasks().add(this);
    }
    public void setTask(Task task) {
        this.task = task;
        if (task != null && task.getStudentsTasks() != null) task.getStudentsTasks().add(this);
    }
}
```

Рисунок 6 – ORM модель SrudentsTasks

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

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

```java
@Entity
@Table(name = "students_tutors")
@Getter
@Setter
@NoArgsConstructor
public class StudentsTutors {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;
  @ManyToOne
  @JoinColumn(name = "student_id")
  private User student;
  @ManyToOne
  @JoinColumn(name = "tutor_id")
  private User tutor;
  public void setStudent(User student) {
    this.student = student;
    if (student != null && student.getTutors() != null) student.getTutors().add(this);
  }
  public void setTutor(User tutor) {
    this.tutor = tutor;
    if (tutor != null && tutor.getStudents() != null) tutor.getStudents().add(this);
  }
}
```

Рисунок 7 – ORM модель SrudentsTutors

• 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:

• **Контроллер** `ViewController`

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

- `TaskService` → `TaskServiceImpl`

```
@Service
@RequiredArgsConstructor
public class TaskServiceImpl implements TaskService {

  private final TaskRepository taskRepository;
//  private final Path uploadDir = Paths.get(System.getProperty("user.dir"), "uploads");
  private final Path uploadDir = Paths.get( first: "/app/uploads"); 2 usages
  @Override  no usages
  public Task createTask(Task task) { return taskRepository.save(task); }
  @Override  1 usage
  public Task createTaskWithFile(Task task, MultipartFile file) throws IOException {
    String photoUrl = saveFile(file);
    task.setPhotoUrl(photoUrl);
    return taskRepository.save(task);
  }
  @Override  1 usage
  public Task getTaskByIdWithAllRelations(Integer id) {
    return taskRepository.findByIdWithAllRelations(id)
            .orElseThrow(() -> new RuntimeException("Task not found"));
  }
  @Override  no usages
  public List<Task> getAllTasks() { return taskRepository.findAll(); }
  @Override  no usages
  public Task updateTask(Integer id, Task updatedTask, MultipartFile file) throws IOException {
    Task existingTask = getTaskByIdWithAllRelations(id);
    existingTask.setClassName(updatedTask.getClassName());
    existingTask.setSubject(updatedTask.getSubject());
    existingTask.setTopic(updatedTask.getTopic());
    existingTask.setDifficulty(updatedTask.getDifficulty());
    existingTask.setTutor(updatedTask.getTutor());
    if (file != null && !file.isEmpty()) {
```

Рисунок 8 – TaskServiceImpl

```java
public byte[] getTaskPhoto(Integer taskId) throws IOException {
  Task task = taskRepository.findById(taskId)
          .orElseThrow(() -> new RuntimeException("Task not found"));
  Path path = Paths.get(task.getPhotoUrl());
  return Files.readAllBytes(path);
}
@Override  no usages
public void deleteTask(Integer id) { taskRepository.deleteById(id); }
private String saveFile(MultipartFile file) throws IOException {  2 usages
  // Получаем расширение
  String original = file.getOriginalFilename();
  String ext = original != null && original.contains(".")
          ? original.substring(original.lastIndexOf( ch: '.'))
          : "";
  String uniqueName = UUID.randomUUID() + ext;
  // Создаём директорию (если ещё нет)
  Files.createDirectories(uploadDir);
  // Абсолютный путь для сохранения
  Path filePath = uploadDir.resolve(uniqueName);
  // Сохраняем файл
  file.transferTo(filePath.toFile());
  // Возвращаем URL для доступа
  return "/uploads/" + uniqueName;
}
@Override  1 usage
public Task getTaskById(Integer id) {
  return taskRepository.findById(id)
          .orElseThrow(() -> new RuntimeException("Task not found with id: " + id));
}
```

Рисунок 9 – TaskServiceImpl

- `StudentsTutorsService` → `StudentsTutorsServiceImpl`

```java
@Override  2 usages
public List<User> getStudentsForTutor(Integer tutorId) {
  User tutor = userRepo.findById(tutorId)
          .orElseThrow(() -> new RuntimeException("Tutor not found: " + tutorId));
  return tutor.getStudents().stream()  Stream<StudentsTutors>
          .map(StudentsTutors::getStudent)  Stream<User>
          .collect(Collectors.toList());
}

@Override  1 usage
@Transactional
public void addStudentToTutor(Integer tutorId, Integer studentId) {
  User tutor = userRepo.findById(tutorId).orElseThrow(() -> new RuntimeException("Tutor not found"));
  User student = userRepo.findById(studentId).orElseThrow(() -> new RuntimeException("Student not found"));

  // Проверка: уже существует такая связь?
  boolean exists = linkRepo.existsByTutorAndStudent(tutor, student);
  if (exists) {
    throw new IllegalStateException("Студент уже привязан к этому репетитору");
  }

  StudentsTutors link = new StudentsTutors();
  link.setTutor(tutor);
  link.setStudent(student);
  linkRepo.save(link);
}
```

Рисунок 10 – StudentsTutorsServiceImpl

```java
@Override 1 usage
@Transactional
public void assignTaskToStudent(Integer tutorId, Integer taskId, Integer studentId) {
  // проверяем, что этот студент действительно привязан к этому тьютору
  User tutor = userRepo.findById(tutorId).orElseThrow();
  User student = userRepo.findById(studentId).orElseThrow();
  boolean linked = tutor.getStudents().stream() Stream<StudentsTutors>
          .map(StudentsTutors::getStudent) Stream<User>
          .anyMatch( User s -> s.getId().equals(studentId));
  if (!linked) {
    throw new RuntimeException("Студент не привязан к репетитору");
  }

  Task task = taskRepo.findById(taskId).orElseThrow();
  // создаём запись StudentsTasks
  StudentsTasks st = new StudentsTasks();
  st.setStudent(student);
  st.setTask(task);
  st.setIsDone(false);
  st.setIsChecked(false);
  st.setScore(null);
  studentsTasksRepo.save(st);
}
```

Рисунок 11 – StudentsTutorsServiceImpl

- `UserService` → `UserServiceImpl`

```java
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
  private final UserRepository userRepository;
  private final PasswordEncoder passwordEncoder;
  /*** Сохраняет пользователя с захешированным паролем.*/
  @Override
  @Transactional
  public User create(User user) {
    // Хешируем пароль перед сохранением
    user.setPassword(passwordEncoder.encode(user.getPassword()));
    return userRepository.save(user);
  }
  /*** Ищет пользователя по email.*/
  @Override 5 usages
  public Optional<User> findByEmail(String email) { return userRepository.findByEmail(email); }
  @Override 1 usage
  public Optional<User> findById(Integer id) { return userRepository.findById(id); }
  @Override 1 usage
  public boolean existsByEmail(String email) { return userRepository.existsByEmail(email); }
  @Override 1 usage
  @Transactional
  public User updateProfile(ProfileDto dto) {
    User user = userRepository.findById(dto.getId())
            .orElseThrow(() -> new RuntimeException("User not found: " + dto.getId()));
    // email
    user.setEmail(dto.getEmail());
    // пароль, если указали
    if (dto.getPassword() != null && !dto.getPassword().isBlank()) {
      user.setPassword(passwordEncoder.encode(dto.getPassword()));
    }
```

Рисунок 12 – UserServiceImpl

Лабораторная работа 2

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

• Написны Dto и Muppper

Основные Dto

```java
@Data  8 usages
public class ProfileDto {
  private Integer id;
  @Email(message = "Некорректный email")
  private String email;
  private String password;
  private String firstname;
  private String lastname;
  @DateTimeFormat(pattern = "yyyy-MM-dd")
  private LocalDate birthdate;
  @Pattern(regexp = "\\d{11}", message = "Телефон должен состоять из 11 цифр")
  private String phone;
  private String telegram;
}
```

Рисунок 13 – ProfileDto

```java
@Getter  2 usages
@Setter
@NoArgsConstructor
@Data
public class TutorPersonalInfoDto {
  private String firstname;
  private String lastname;
  private LocalDate birthdate;
  private String phone;
  private String telegram;
}
```

Рисунок 14 – TutorPersonalInfoDto

```java
@Data  9 usages
public class TaskNRDto {
  private Integer id;
  private String photoUrl;
  private String className;
  private String subject;
  private String description;
  private String topic;
  private Integer difficulty;
  private Integer tutorId;
}
```

Рисунок 15 – TaskNRDto

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).

```java
@Component
@RequiredArgsConstructor
public class JwtTokenFilter extends GenericFilterBean {
    private final JwtTokenProvider jwtTokenProvider;
    @Override  no usages
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
            throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) req;
        String token = null;
        String authHeader = request.getHeader( s: "Authorization");
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            token = authHeader.substring( beginIndex: 7);
        }
        // Если в заголовке нет — ищем в куках
        if (token == null && request.getCookies() != null) {
            for (Cookie c : request.getCookies()) {
                if ("accessToken".equals(c.getName())) {
                    token = c.getValue();
                }
            }
        }
        if (token != null && jwtTokenProvider.validate(token)) {
            Authentication auth = jwtTokenProvider.getAuthentication(token);
            SecurityContextHolder.getContext().setAuthentication(auth);
        }

        chain.doFilter(req, res);
    }
}
```

Рисунок 16 – JwtTokenFilter

```java
@Component
@RequiredArgsConstructor
public class JwtTokenProvider {
    private final JwtProperties props;
    private SecretKey key;  5 usages
    private final UserDetailsService userDetailsService;
    @PostConstruct
    public void init() { key = Keys.hmacShaKeyFor(props.getSecret().getBytes()); }
    public String createAccessToken(Authentication auth) {  2 usages
        UserJwtEntity user = (UserJwtEntity) auth.getPrincipal();
        Instant now = Instant.now();
        // Кладём только названия ролей!
        var roles = user.getAuthorities().stream() Stream<capture of extends GrantedAuthority>
                .map( capture of extends GrantedAuthority a -> a.getAuthority()) Stream<String>
                .collect(Collectors.toList());
        return Jwts.builder()
                .setSubject(user.getUsername())
                .claim( s: "id", user.getId())
                .claim( s: "roles", roles) // <-- теперь список строк, а не объектов
                .setIssuedAt(Date.from(now))
                .setExpiration(Date.from(now.plus(props.getAccess(), ChronoUnit.HOURS)))
                .signWith(key, SignatureAlgorithm.HS256)
                .compact();
    }
    public String createRefreshToken(Authentication auth) {  2 usages
        UserJwtEntity user = (UserJwtEntity) auth.getPrincipal();
        Instant now = Instant.now();
        return Jwts.builder()
            .setSubject(user.getUsername())
            .claim( s: "id", user.getId())
            .setIssuedAt(Date.from(now))
            .setExpiration(Date.from(now.plus(props.getRefresh(), ChronoUnit.DAYS)))
```

Рисунок 17 – JwtTokenProvider

```
            .compact();
}
public String createRefreshToken(Authentication auth) {  2 usages
    UserJwtEntity user = (UserJwtEntity) auth.getPrincipal();
    Instant now = Instant.now();
    return Jwts.builder()
        .setSubject(user.getUsername())
        .claim( s: "id", user.getId())
        .setIssuedAt(Date.from(now))
        .setExpiration(Date.from(now.plus(props.getRefresh(), ChronoUnit.DAYS)))
        .signWith(key, SignatureAlgorithm.HS256)
        .compact();
}
public boolean validate(String token) {  2 usages
    try {
        return Jwts.parserBuilder().setSigningKey(key).build()
            .parseClaimsJws(token).getBody().getExpiration().after(new Date());
    } catch (Exception e) {
        return false;
    }
}
public Authentication getAuthentication(String token) {  2 usages
    var claims = Jwts.parserBuilder().setSigningKey(key).build()
        .parseClaimsJws(token).getBody();
    String username = claims.getSubject();
    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
    return new UsernamePasswordAuthenticationToken(userDetails,  credentials: "", userDetails.getAuthorities());
}
```

Рисунок 18 – JwtTokenProvider

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

```
<!-- для студента – список его задач -->
<div sec:authorize="hasRole('STUDENT')">
  <h2>Мои задачи</h2>
  <table class="table table-hover">
    <thead>
    <tr>
      <th>Класс</th>
      <th>Предмет</th>
      <th>Тема</th>
      <th>Сложность</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="task : ${myTasks}"
        th:onclick="|window.location='@{/frontend/tasks/{id}(id=${task.id})}'|"
        style="...">
      <td th:text="${task.className.displayName}"></td>
      <td th:text="${task.subject.displayName}"></td>
      <td th:text="${task.topic}"></td>
      <td th:text="${task.difficulty}"></td>
    </tr>
    </tbody>
  </table>
</div>
<!-- для тьютор  -->
<div sec:authorize="!hasRole('STUDENT')">
  <div class="text-center py-5">
    <h1 class="display-4">Добро пожаловать в EasyLearning!</h1>
  </div>
```

Рисунок 19 – index.html

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

```
<body>
<section layout:fragment="content">
    <h2 class="mb-3">Вход</h2>
    <form id="loginForm" method="post" class="vstack gap-2 col-md-4 mx-auto">
        <input class="form-control" type="email" name="username" placeholder="Email" required>
        <input class="form-control" type="password" name="password" placeholder="Пароль" required>
        <button class="btn btn-primary" type="submit">Войти</button>
    </form>
    <p class="mt-3 text-center">Нет аккаунта? <a th:href="@{/frontend/register}">Регистрация</a></p>
</section>
</body>
```

Рисунок 20 – login.html

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

```
<body>
<section layout:fragment="content">
    <h2 class="mb-3">Регистрация</h2>
    <form id="registerForm" method="post" class="vstack gap-2 col-md-4 mx-auto">
        <input class="form-control" type="email"    name="email"    placeholder="Email" required>
        <input class="form-control" type="password" name="password" placeholder="Пароль" required>
        <div class="form-check">
          <input class="form-check-input" type="checkbox" name="tutor" id="tutorChk">
          <label class="form-check-label" for="tutorChk">Я&NBSPрепетитор</label>
        </div>
        <button class="btn btn-success" type="submit">Зарегистрироваться</button>
    </form>
    <p class="mt-3 text-center"><a th:href="@{/frontend/login}">Уже есть аккаунт? Войти</a></p>
</section>
</body>
```

Рисунок 21 – register.html

Личный кабинет (profile.html)

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

```
<body>
<section layout:fragment="content">
    <h1>Личный кабинет</h1>
    <div th:if="${success}" class="alert alert-success" th:text="${success}"></div>
    <form th:action="@{/frontend/profile}" th:object="${profile}" method="post" class="row g-3">
        <input type="hidden" th:field="*{id}"/>
        <div class="col-md-6">
            <label class="form-label">Ваш уникальный идентификатор</label>
            <input type="text"
                   th:field="*{id}"
                   class="form-control"
                   readonly/>
        </div>
        <div class="col-md-6">
            <label class="form-label">Email</label>
            <input type="email" th:field="*{email}" class="form-control" readonly/>
            <div th:if="${#fields.hasErrors('email')}" class="text-danger">
                <span th:errors="*{email}"></span>
            </div>
        </div>
        <div class="col-md-6">
            <label class="form-label">Новый пароль</label>
            <input type="password" th:field="*{password}" class="form-control"
                   placeholder="Оставьте пустым, чтобы не менять"/>
        </div>
        <div class="col-md-4">
            <label class="form-label">Имя</label>
            <input type="text" th:field="*{firstname}" class="form-control" required/>
        </div>
        <div class="col-md-4">
            <label class="form-label">Фамилия</label>
```

Рисунок 22 – profile.html

Мои студенты (students.html)

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

```
<h1>Мои студенты</h1>
<!-- Здесь сообщение об ошибке -->
<div th:if="${error}" class="alert alert-danger" th:text="${error}"></div>
<table class="table table-bordered">
    <thead class="table-light">
    <tr>
        <th>ID</th>
        <th>Email</th>
        <th>Имя</th>
        <th>Фамилия</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="student : ${students}">
        <td th:text="${student.id}">1</td>
        <td th:text="${student.email}">mail@example.com</td>
        <td th:text="${student.personalInfo.firstname}">Иван</td>
        <td th:text="${student.personalInfo.lastname}">Иванов</td>
    </tr>
    </tbody>
</table>
<form th:action="@{/frontend/students}" method="post" class="d-flex gap-2 mt-4">
    <input type="number"
           name="studentId"
           class="form-control"
           placeholder="ID студента"
           required />
    <button type="submit" class="btn btn-primary">Добавить студента</button>
</form>
```

Рисунок 23 – students.html

Мои репетиторы (tutors.html)

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

```
<body>
<section layout:fragment="content">
    <h1>Мои репетиторы</h1>
    <table class="table table-hover">
        <thead>
        <tr>
            <th>ID</th>
            <th>Email</th>
            <th>Имя</th>
            <th>Фамилия</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="tutor : ${tutors}"
            th:onclick="|window.location='@{/frontend/tutors/{id}(id=${tutor.id})}'|"
            style="...">
            <td th:text="${tutor.id}">1</td>
            <td th:text="${tutor.email}">tutor@mail.com</td>
            <td th:text="${tutor.personalInfo.firstname}">Анна</td>
            <td th:text="${tutor.personalInfo.lastname}">Петрова</td>
        </tr>
        </tbody>
```

Рисунок 24 – tutors.html

Список задач (tasks.html)

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

```
        <th:block th:each="subjectEntry : ${classEntry.value}">
            <tr class="table-active">
                <td colspan="5" th:text="'  Предмет: ' + ${subjectEntry.key}"></td>
            </tr>
            <tr th:each="task : ${subjectEntry.value}"
                th:onclick="|window.location='@{/frontend/tasks/{id}(id=${task.id})}'|"
                style="...">
                <td th:text="${task.className.displayName}"></td>
                <td th:text="${task.subject.displayName}"></td>
                <td th:text="${task.topic}"></td>
                <td th:text="${task.difficulty}"></td>
                <td th:text="${task.description}"></td>
                <td onclick="event.stopPropagation()">
                    <form th:action="@{/frontend/tasks/assign}" method="post" class="d-flex">
                        <input type="hidden" name="taskId" th:value="${task.id}"/>
                        <select name="studentId"
                                class="form-select form-select-sm me-1"
                                required>
                            <option th:each="stu : ${studentsList}"
                                    th:value="${stu.id}"
                                    th:text="${stu.personalInfo.firstname + ' ' + stu.personalInfo.lastname}">
                            </option>
                        </select>
                        <button type="submit" class="btn btn-sm btn-success">Назначить</button>
                    </form>
```

Рисунок 25 – tasks.html

Просмотр задачи (task_view.html)

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

```
<h1>Просмотр задачи</h1>
<div class="mb-3">
    <img th:src="@{${task.photoUrl}}" alt="Фото задачи" class="img-fluid"/>
</div>
<dl class="row">
    <dt class="col-sm-3">Класс</dt>
    <dd class="col-sm-9" th:text="${task.className.displayName}">CLASS_1</dd>
    <dt class="col-sm-3">Предмет</dt>
    <dd class="col-sm-9" th:text="${task.subject.displayName}">MATH</dd>
    <dt class="col-sm-3">Тема</dt>
    <dd class="col-sm-9" th:text="${task.topic}">Algebra</dd>
    <dt class="col-sm-3">Сложность</dt>
    <dd class="col-sm-9" th:text="${task.difficulty}">3</dd>
    <dt class="col-sm-3">Описание</dt>
    <dd class="col-sm-9" th:text="${task.description}">Описание задачи</dd>
</dl>
<a class="btn btn-secondary" th:href="@{/frontend/tasks}">Назад к списку</a>
</section>
```

Рисунок 26 – task_view.html

Создание задачи (task_new.html)

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

```
<form th:action="@{/frontend/new}"
      method="post"
      enctype="multipart/form-data"
      th:object="${taskForm}">
    <!-- Выпадающий список для ClassLevel -->
    <div class="mb-3">
        <label for="className" class="form-label">Класс</label>
        <select id="className"
                th:field="*{className}"
                class="form-select"
                required>
            <option th:each="cl : ${classLevels}"
                    th:value="${cl}"
                    th:text="${cl.displayName}"></option>
        </select>
    </div>
    <!-- Выпадающий список для Subject -->
    <div class="mb-3">
        <label for="subject" class="form-label">Предмет</label>
        <select id="subject"
                th:field="*{subject}"
                class="form-select"
                required>
            <option th:each="sub : ${subjects}"
                    th:value="${sub}"
                    th:text="${sub.displayName}"></option>
        </select>
    </div>
```

Рисунок 27 – task_new.html

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:

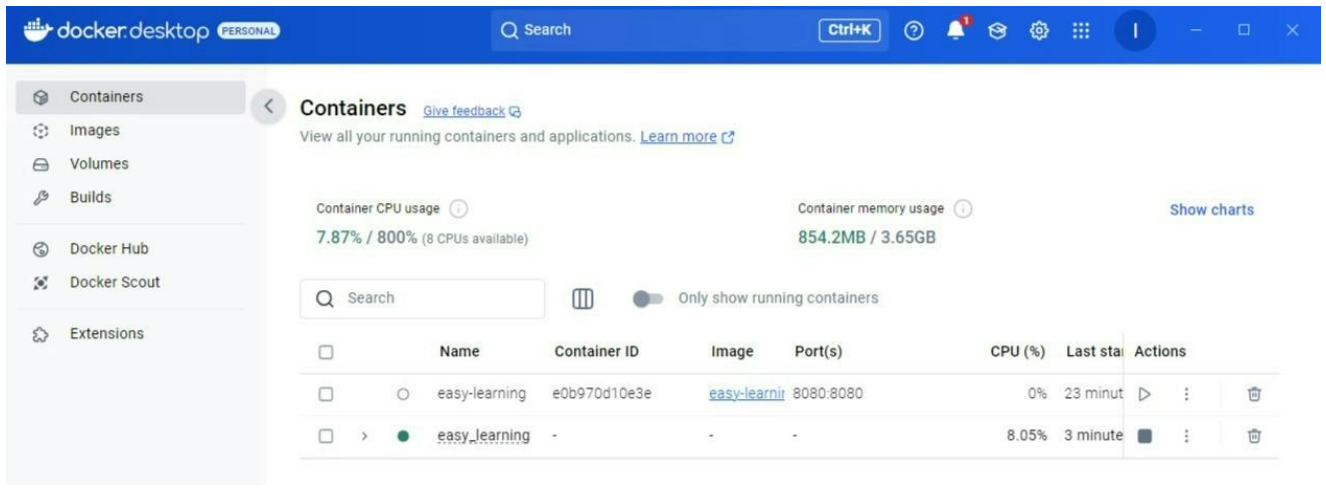

Рисунок 27 – Docker

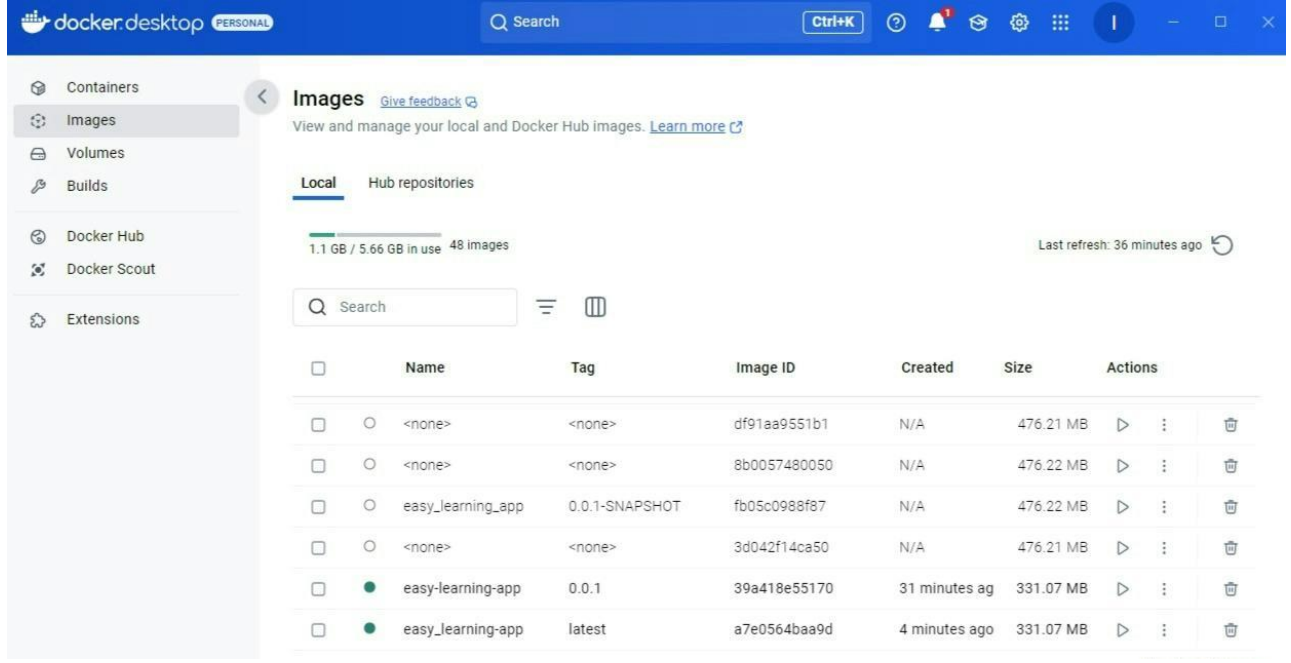

Рисунок 28 – Docker

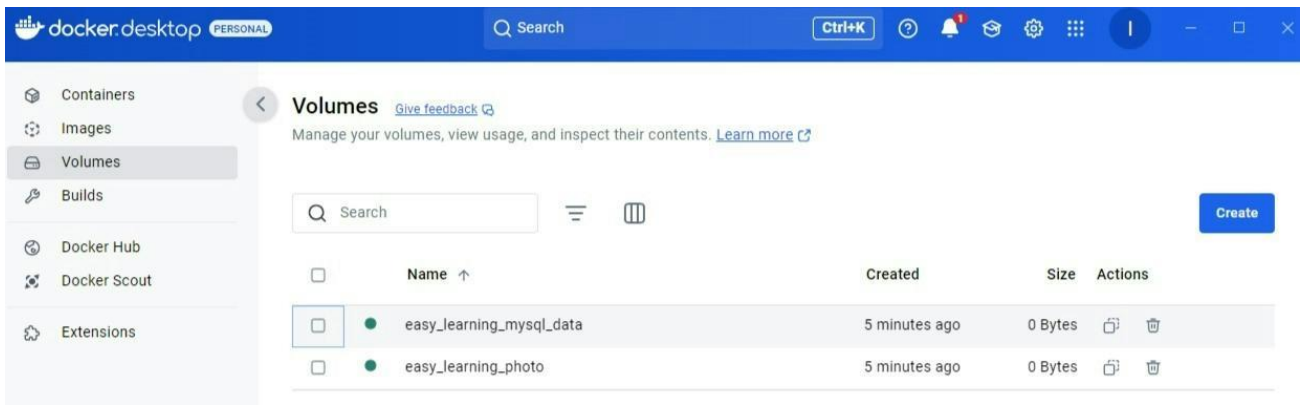

Рисунок 29 – Docker

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

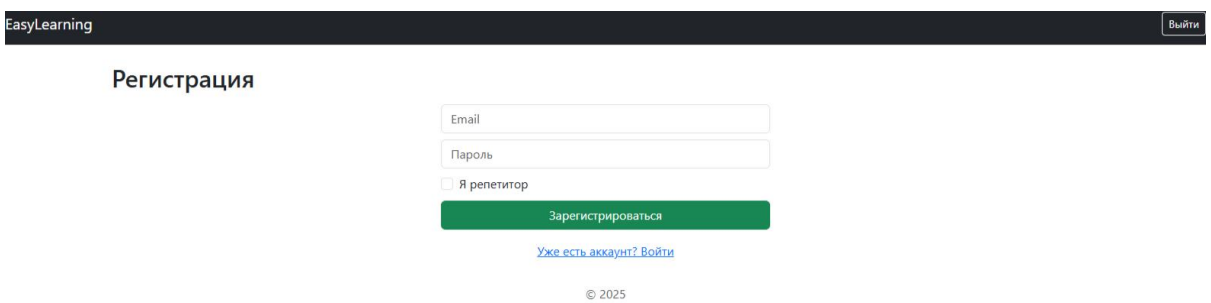

Рисунок 30 – Регистрация

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

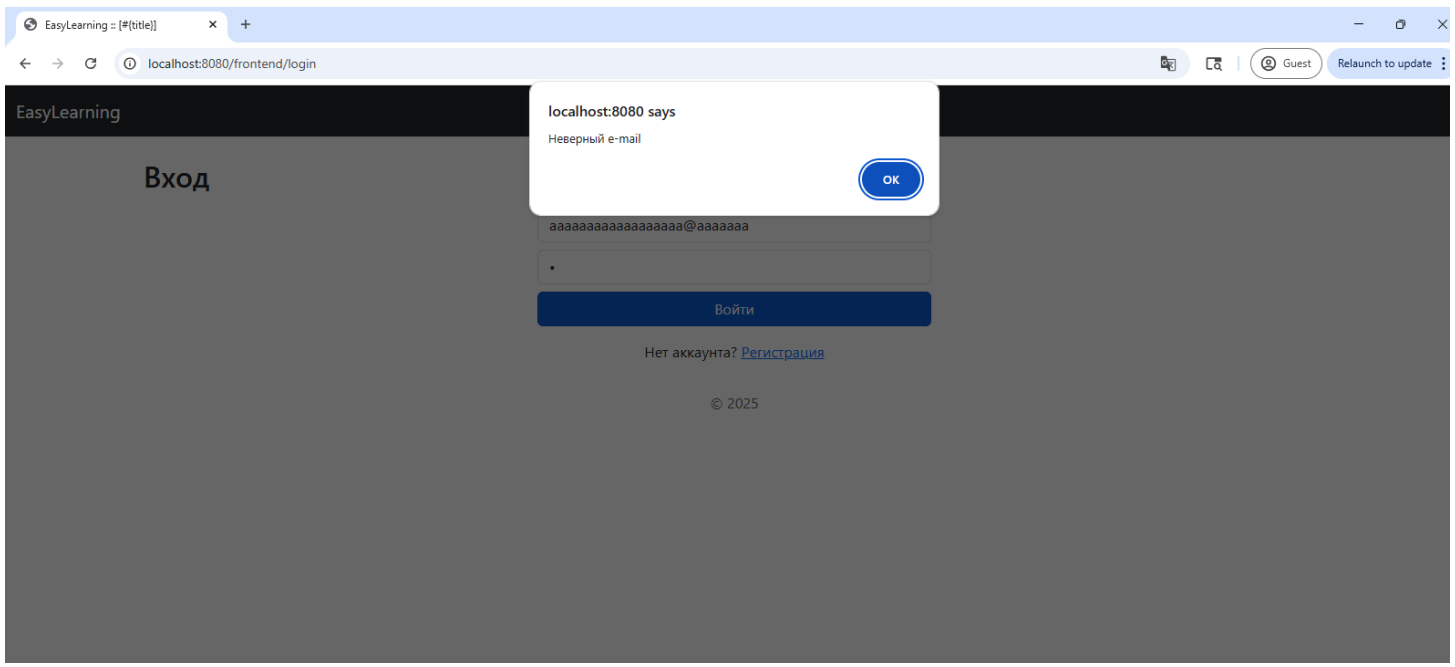

Рисунок 31 – Вход

3. Проверка. На некорректный email

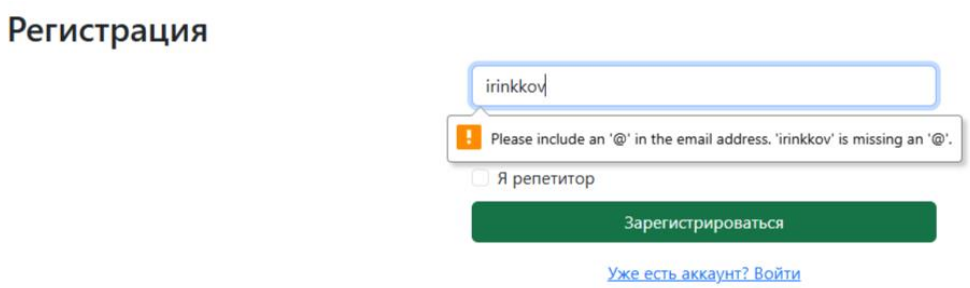

Рисунок 32 – Ошибка email при регистрации

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

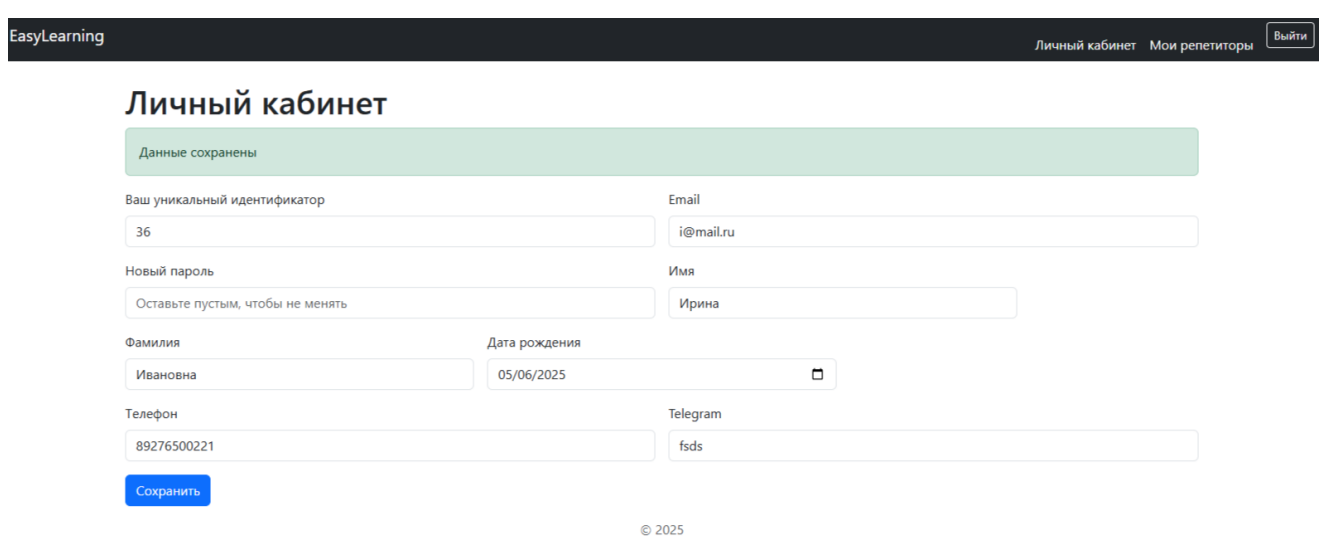

Рисунок 33 – Личный кабинет

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

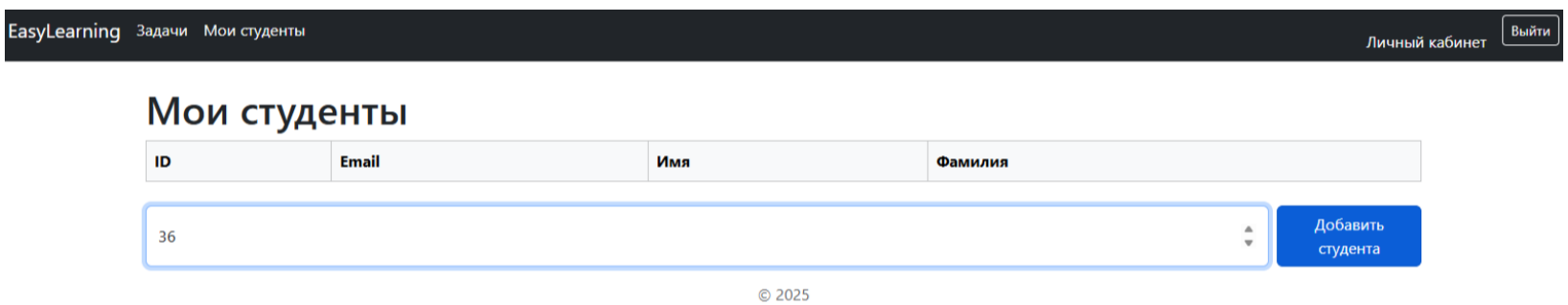

Рисунок 34 – Мои студенты

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

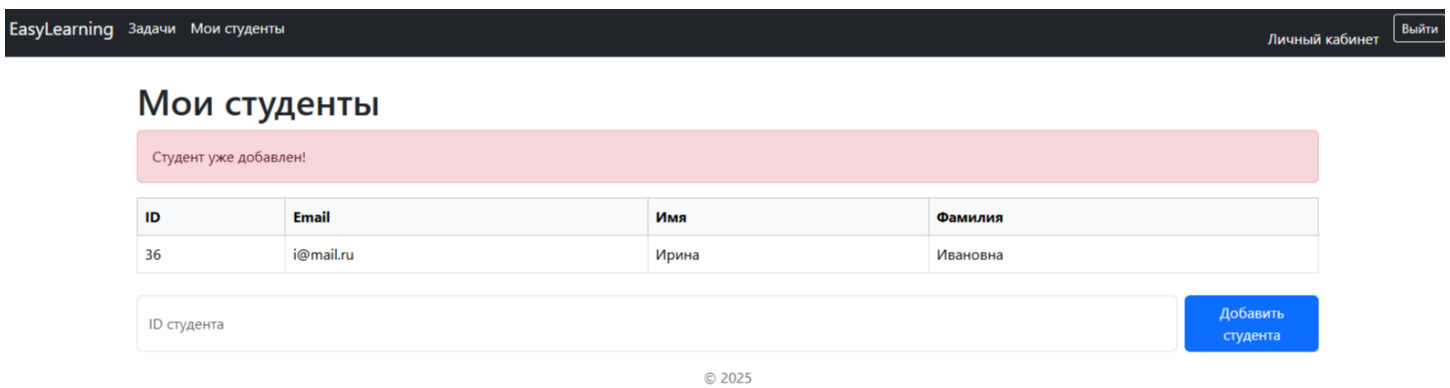

Рисунок 35 – Мои студенты (ошибки)

7. У студента на странице появляется его репетитор.

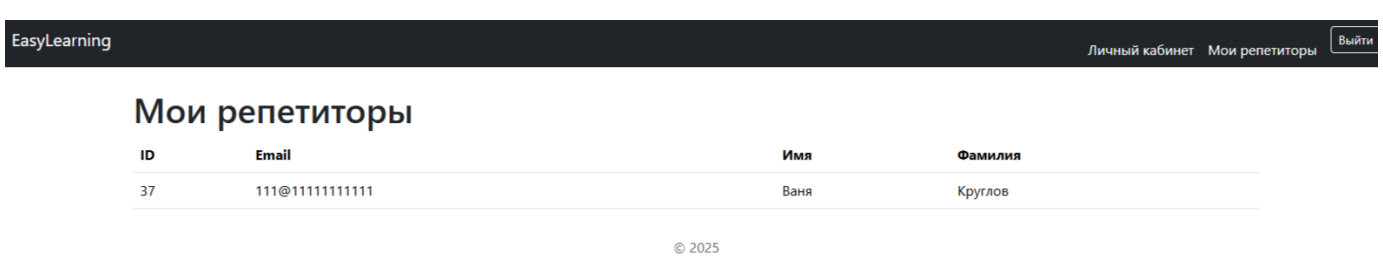

Рисунок 36 – Мои репетиторы

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

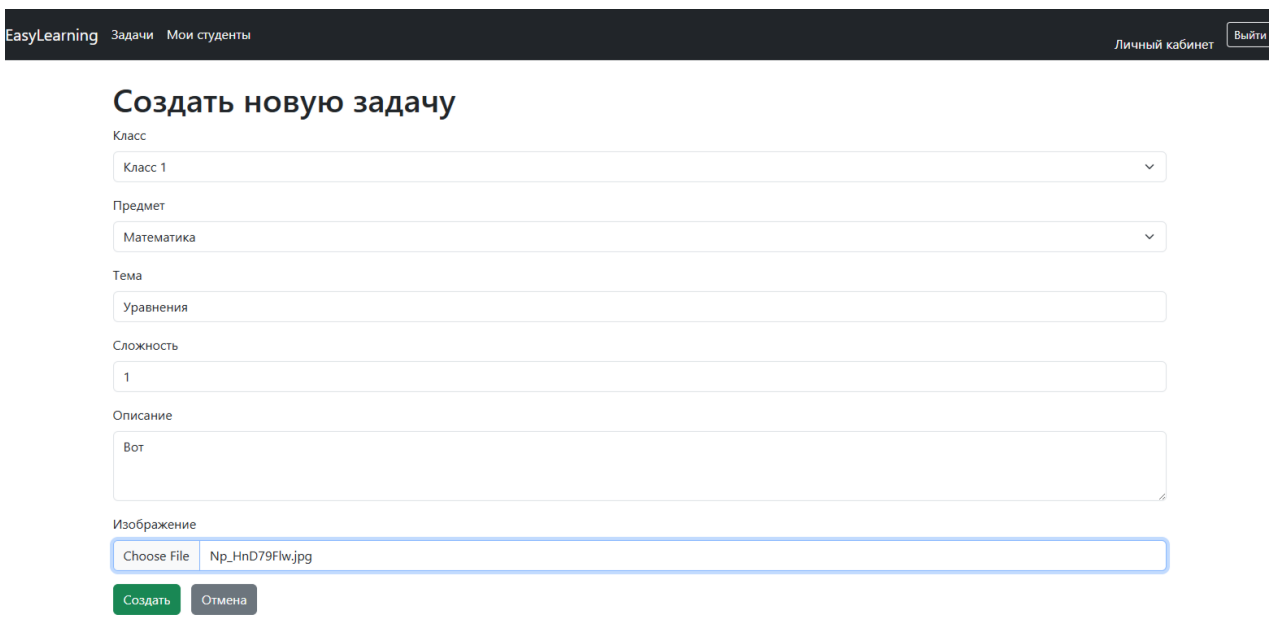

Рисунок 37 – Новая задача

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

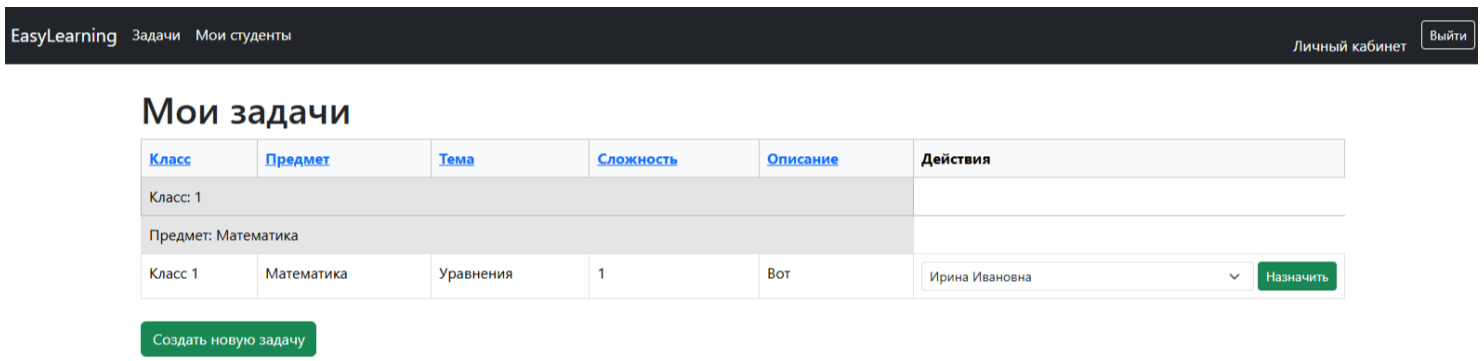

Рисунок 38 – Мои задачи

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

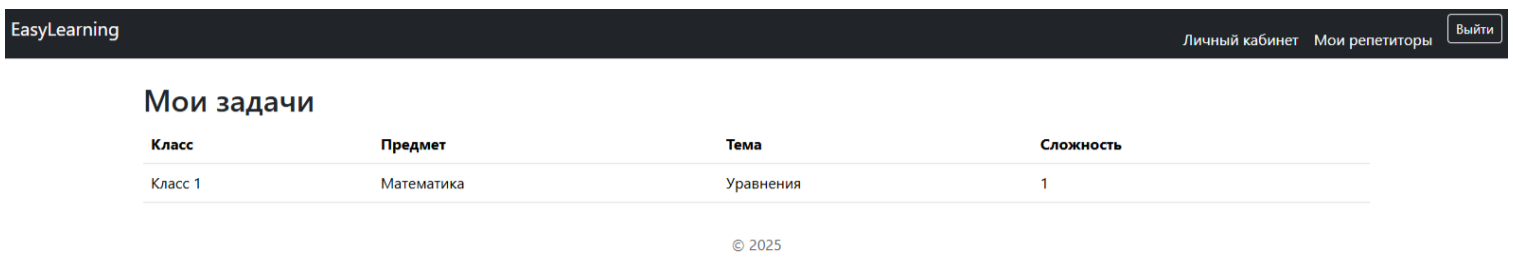

Рисунок 39 – Мои задачи

11. Кнопки выход работают. Перебрасывают на станицу входа.