Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий  
Кафедра «Информатика и вычислительная техника»

Направление подготовки/ специальность: Системная и программная инженерия

ОТЧЕТ

по проектной практике

Студент: Кузнецов Никита Владимирович Группа: 241 – 326

Место прохождения практики: Московский Политех, кафедра «СМАРТ-технологии»

Отчет принят с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2025

**ОГЛАВЛЕНИЕ**

**ВВЕДЕНИЕ**

1. **Общая информация о проекте:**

* Название проекта
* Цели и задачи проекта

1. **Общая характеристика деятельности организации *(заказчика проекта)***

* Наименование заказчика
* Организационная структура
* Описание деятельности

1. **Описание задания по проектной практике**
2. **Описание достигнутых результатов по проектной практике**

**ЗАКЛЮЧЕНИЕ** *(выводы о проделанной работе и оценка ценности выполненных задач для заказчика)*

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

**ПРИЛОЖЕНИЯ** *(при необходимости)*

**ВВЕДЕНИЕ**

**1. Общая информация о проекте**

Название проекта:

Разработка платформы жестовой коммуникации ЦНИИ РЖЯ

Цели проекта:

Цель проекта заключается в создании комплексной электронной платформы, предназначенной для систематизации знаний о русском жестовом языке (РЖЯ), повышения доступности и эффективности его изучения, а также обеспечения коммуникации между носителями РЖЯ и широкой аудиторией. Основной элемент — электронный словарь жестов с интерфейсом как на русском, так и на жестовом языке, дополненный функциональными модулями.

Задачи проекта:

* Создание базы данных жестов и их лингвистических характеристик;
* Разработка и внедрение интерфейса словаря, доступного как на русском, так и на жестовом языках;
* Интеграция 3D-анимированного аватара для отображения жестов;
* Внедрение системы морфологической разметки для связи жестов с грамматическими формами;
* Модернизация технологического стека (переход на Go, React, PostgreSQL);

Отдельным подмодулем проекта является платформа аттестации переводчиков РЖЯ, реализующая подачу заявлений, организацию экзаменов и генерацию итоговой документации. Это необходимо для автоматизации и значительного ускорения непростого процесса аттестации переводчиков, которые впоследствии будут развивать электронный словарь.

**2. Общая характеристика деятельности организации (заказчика проекта)**

Наименование заказчика:

Центральный научно-исследовательский институт русского жестового языка (ЦНИИ РЖЯ)

Организационная структура: ЦНИИ РЖЯ представляет собой автономную некоммерческую научно-исследовательскую организацию. В структуру института входят лингвистические, технические и образовательные подразделения, включая лабораторию лексикографии жестового языка, отдел разработки цифровых инструментов и группу по обучению и сертификации переводчиков.

Описание деятельности: ЦНИИ РЖЯ осуществляет фундаментальные и прикладные исследования в области русского жестового языка, занимается разработкой программного обеспечения, обучением переводчиков и созданием цифровых инструментов — в частности, словарей, систем морфологической разметки и платформ для сертификации. Институт ведёт открытую публикацию лингвистических данных и активно сотрудничает с вузами, включая Московский Политех. Ключевым направлением является создание электронной справочно-аналитической системы "Толковый лексикографический словарь русского жестового языка" и связанных с ней модулей, включая образовательные и технические решения.

**3. Описание задания по проектной практике**

В рамках учебной практики мне было поручено реализовать серверную часть подпроекта — платформу аттестации переводчиков русского жестового языка (РЖЯ), входящую в состав более крупного проекта «Платформа жестовой коммуникации ЦНИИ РЖЯ». Этот модуль играет ключевую роль в автоматизации процесса сертификации специалистов, которые в дальнейшем будут работать над наполнением словаря РЖЯ.

Главной целью задания было — создание полнофункционального веб-сервиса, который позволяет:

* подавать заявления на участие в аттестации;
* загружать и проверять необходимые документы;
* формировать расписание и управлять экзаменационной сессией;
* проводить онлайн-оценивание через WebSocket в режиме реального времени;
* генерировать выходные документы (например, сертификаты) на основе шаблонов.

Задача была разбита на следующие ключевые этапы:

1. Проектирование архитектуры платформы
   * Разработка структуры базы данных PostgreSQL
   * Определение связей между таблицами (пользователи, документы, экзамены, роли и т.д.)
2. Реализация API на языке Go с использованием Fiber
   * Аутентификация и авторизация с использованием JWT (access/refresh)
   * Обработка multipart-запросов с сохранением данных и файлов
   * Реализация защищённых маршрутов для администратора, переводчика, председателя и экзаменаторов
3. Поддержка реального времени через WebSocket
   * Организация экзаменационных сессий
   * Передача статуса участников и баллов оценки между членами комиссии
4. Генерация ODT-документов
   * Использование шаблонов с плейсхолдерами и автозамена на данные из БД
   * Подстановка изображений (например, дипломов, паспортов) в отчётные документы
5. Интеграция с основным сайтом проекта
   * Совместимость с дизайном и стилем интерфейса
   * Подключение административной панели к интерфейсу модераторов и экспертов

**4. Описание достигнутых результатов по проектной практике**

В результате выполнения задания была спроектирована (см. Приложение 1), реализована и протестирована платформа аттестации переводчиков русского жестового языка (РЖЯ). Основной стек: Go (Golang) + Fiber, GORM, PostgreSQL. Система пока функционирует автономно, но в скором времени планируется сделать ее частью общей платформы жестовой коммуникации ЦНИИ РЖЯ.

Архитектура базы данных (см. Приложение 2)

Проектная база данных построена вокруг следующих ключевых сущностей:

1. Пользователи (User) — модель содержит информацию о пользователях, включая:
   * Личные данные в нескольких падежах (необходимость вызвана автозаполнением документов),
   * Контактные данные,
   * Роль в системе (examiner, student, admin),
   * Технические поля (JestID, StoragePath и др.).

*type User struct {*

*JestID string `gorm:"unique"`*

*Email string `gorm:"not null;unique"`*

*Role string `gorm:"not null"`*

*Password string `gorm:"not null"`*

*// ...*

*}*

1. Заявления (Application) — фиксируют информацию о подаче заявки на аттестацию:
   * Категория, организация, опыт, обучение и прочее,
   * Согласие на обработку персональных данных,
   * Статус обработки и причины отклонения (через связную модель ApplicationDecline).

*type Application struct {*

*UserID uint*

*ApplicationType string*

*BasisForAttestation string*

*Consent bool*

*Status string}*

1. Документы (UserDocument) — система поддерживает загрузку и классификацию документов по типам:
   * DocumentType (паспорт, диплом, СНИЛС, характеристика и др.),
   * FilePath — путь к файлу,
   * Автоматическое удаление при повторной загрузке.

*type UserDocument struct {*

*DocumentName string*

*DocumentType string*

*FilePath string*

*}*

1. Паспорт / диплом (Passport, EducationDocument) — отдельные модели для хранения личных документов с привязкой к UserID.
2. Экзамены (Exam, ExamStudent, ExamExaminer) — реализована полноценная модель:
   * Exam включает дату, статус (planned, scheduled, completed), коды комиссии (JestID),
   * Связанные таблицы exam\_students, exam\_examiners отражают участников.

*type Exam struct {*

*Status string // planned, scheduled, completed*

*Quorum int*

*JestID string*

*}*

1. Оценки (ExamGrade, ExamGradeCriterion) — поддерживается раздельная фиксация оценок по критериям:
   * Поддержка флага Abstained,
   * Система переоценки,
   * Возможность не выставлять балл (Score \*int).

*type ExamGradeCriterion struct {*

*CriterionID int*

*Score \*int*

*}*

7. JWT-Аутентификация

Реализована безопасная авторизация с поддержкой:

* access и refresh токенов,
* сохранение RefreshToken в базе,
* обновление access-токена через /refresh.

*type Token struct {*

*User string*

*RefreshToken string*

*}*

8. Загрузка и просмотр документов

Загрузка документов реализована через multipart/form-data. Особенности:

* Поддержка разных типов документов: "паспорт\_разворот", "паспорт\_прописка", "диплом", "снилс" и др.
* Файлы хранятся в папках пользователей, формируемых на основе ФИО и JestID.

*func SaveFile(file \*multipart.FileHeader, path string) error {*

*return c.SaveFile(file, path)*

*}*

9. Проведение экзаменов

Экзаменационная сессия реализована через WebSocket:

* Студенты подключаются к конкретному экзамену,
* Экзаменаторы выставляют оценки в реальном времени,
* Председатель может следить за ходом голосования.

Сервер на Go обрабатывает JSON-сообщения типа init\_user, grade\_update, discussion\_start и др.

10. Генерация документов

После завершения экзамена:

* Автоматически генерируется ODT-файл на основе шаблона,
* Подставляются текстовые поля и изображения из StoragePath,
* Поддержка вставки сканов в плейсхолдеры с alt="{{passport\_1}}" и др.

// Пример замены текста

*node.TextContent = strings.ReplaceAll(node.TextContent, "{{fullname}}", user.FullName)*

API-эндпоинты

* 1. Общее количество маршрутов

Сервер реализует 50+ HTTP-эндпоинтов, распределённых по 4 основным модулям:

* registration.go — регистрация и подтверждение почты
* user.go — взаимодействие студента (заявки, документы, экзамены)
* admin.go — маршруты администратора (работа с экзаменами, пользователями)
* pages.go — отрисовка HTML-страниц (универсальные маршруты)

registration.go:

*app.Get("/registration", controllers.RegisterPage)*

*app.Post("/registration", controllers.Register)*

*app.Get("/verify", controllers.Verify)*

*app.Post("/confirm", controllers.Confirm)*

*Назначение: регистрация по email, подтверждение почты, установка пароля.*

user.go:

Маршруты находятся в RegisterUserRoutes, группа /user и защищены UniversalAuthMiddleware.

Работа с профилем:

*/user/profile*

*/user/maindata*

*/user/change/photo*

Работа с документами:

*/user/document*

*/user/documents/send*

*/user/data/correct*

*/user/data/aprove*

*/user/documents/reason*

Работа с заявками:

*/user/application*

*/user/create-application // GET — отображение формы*

*/user/create-application // POST — отправка формы*

Работа с экзаменами:

*/user/exams*

*/user/exam/:id*

*/user/exam/check-chairman*

*/user/exam/waiting/:exam\_id*

*/user/exam/student/:exam\_id/:student\_id*

*/user/exam/start/:exam\_id*

*/user/exam/start-page/:exam\_id*

admin.go:

Админские маршруты, доступные только пользователям с ролью admin.

Студенты:

*/admin/applications*

*/admin/api/student*

*/admin/student/profile*

*/admin/student/documents*

*/admin/application/approve*

*/admin/application/decline*

Экзамены:

*/admin/exam/create*

*/admin/exam/show*

*/admin/exam/save*

*/admin/exam/delete*

*/admin/exam/finalize*

*/admin/api/exam/waiting*

WebSocket:

*/exam/ws*

Поддерживает соединение для реального времени в ходе экзамена:

* пересылка оценок
* контроль подключения экзаменаторов
* координация между председателем и участниками

Пример контроллера: подача заявления

*func SaveUserApplication(c \*fiber.Ctx) error {*

*var app models.Application*

*if err := c.BodyParser(&app); err != nil {*

*return c.Status(400).SendString("Invalid request")*

*}*

*app.CreatedAt = time.Now()*

*database.DB.Create(&app)*

*return c.JSON(app)*

*}*

Назначение: принимает JSON с данными заявления, записывает в таблицу applications.

Пример запроса с клиента (AJAX):

*fetch("/user/create-application", {*

*method: "POST",*

*headers: { "Content-Type": "application/json" },*

*body: JSON.stringify({*

*UserID: 42,*

*ApplicationType: "Первичная",*

*NativeLanguage: "Русский",*

*// ...*

*}),*

*});*

Примечание

Маршруты покрывают полный цикл: от регистрации до экзамена, включая:

* редактирование профиля;
* загрузку, проверку и отображение документов;
* участие в онлайн-экзамене через WebSocket;
* автоматическую генерацию документов;
* административную проверку и управление.

JWT-авторизация

Внедрена полная схема авторизации и обновления сессии:

* access токен — на 15 минут;
* refresh токен — на 7 дней, хранящийся в HTTP-only cookie;
* Поддержка middleware на Fiber:

*func AuthMiddleware(c \*fiber.Ctx) error {*

*token := c.Cookies("access\_token")*

*claims, err := VerifyToken(token)*

*if err != nil {*

*return c.Status(fiber.StatusUnauthorized).SendString("Unauthorized")*

*}*

*c.Locals("user\_id", claims.UserID)*

*return c.Next()*

*}*

Работа с файлами и изображениями

Платформа поддерживает:

* загрузку изображений (JPEG, PNG);
* предварительный просмотр;
* сортировку по типу и названию.

*func UploadDocuments(c \*fiber.Ctx) error {*

*form, err := c.MultipartForm()*

*files := form.File["passport"]*

*for \_, file := range files {*

*path := fmt.Sprintf("storage/%s", file.Filename)*

*c.SaveFile(file, path)*

*db.Create(&models.UserDocument{UserID: uid, Path: path, Type: "passport"})*

*}*

*return c.SendString("Uploaded")*

*}*

Онлайн-экзамены через WebSocket

Был реализован модуль реального времени:

* Подключение экзаменаторов и председателя;
* Передача информации о студентах;
* Ввод и пересылка оценок;
* Завершение экзамена по команде председателя.

*func HandleWebSocket(c \*websocket.Conn) {*

*for {*

*msg := Message{}*

*if err := c.ReadJSON(&msg); err != nil {*

*break*

*}*

*broadcast <- msg // пересылаем другим участникам*

*}*

*}*

Генерация документов ODT

Автоматизирована генерация заявлений через шаблоны .odt:

* Данные пользователя подставляются через zip + xml парсинг;
* Изображения вставляются по alt text плейсхолдерам ({{passport}}, {{snils}});

Пример вставки текста:

*func ReplacePlaceholder(content string, placeholder string, value string) string {*

*return strings.ReplaceAll(content, "{{"+placeholder+"}}", value)*

*}*

Развёртывание и публикация (см. Приложение 3)

* Платформа развернута на сервере с nginx;
* И доступна по адресу: <https://att.cnii-jest.ru>

**ЗАКЛЮЧЕНИЕ**

В ходе проектной практики была практически полностью реализована платформа аттестации переводчиков русского жестового языка (РЖЯ). Работа включала проектирование и реализацию архитектуры базы данных, построение REST API, интеграцию с системой WebSocket для проведения онлайн-экзаменов, а также генерацию документов на основе шаблонов. Результатом стало создание backend-модуля, совместимого в будущем с основной платформой «Платформа жестовой коммуникации ЦНИИ РЖЯ». Разработанная система решает актуальные задачи автоматизации аттестации переводчиков, включая:

* Подачу и верификацию заявлений;
* Загрузку и классификацию документов;
* Проведение онлайн-экзамена с синхронной оценкой;
* Генерацию отчётных документов.

Проект имеет важную практическую и социальную ценность: он обеспечивает цифровизацию процессов, ранее выполнявшихся вручную, и значительно повышает доступность и быстроту проведения процедуры аттестации. Для заказчика — ЦНИИ РЖЯ — данный инструмент облегчает администрирование, повышает прозрачность и снижает издержки.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Документация Go Fiber — <https://docs.gofiber.io>
2. GORM ORM для Golang — <https://gorm.io/docs>
3. WebSocket API — MDN: <https://developer.mozilla.org/ru/docs/Web/API/WebSocket>
4. OpenDocument Format (ODF) XML спецификация — <https://docs.oasis-open.org>
5. PostgreSQL Documentation — <https://postgrespro.ru/docs>
6. Внутренние материалы ЦНИИ РЖЯ (необходимая для корректной логики процедуры информация о порядке проведения аттестации) – <https://cnii-jest.ru>

**ПРИЛОЖЕНИЕ**

[Рисунок 1 Архитектура системы на Go + PostgreSQL 18](#_Toc198537006)

[Рисунок 2 Общая структура 19](#_Toc198537007)

[Рисунок 3 Первые 3 таблицы подробно 20](#_Toc198537008)

[Рисунок 4 Следующие 3 таблицы подробно 21](#_Toc198537009)

[Рисунок 5 Следующие 4 таблицы подробно 22](#_Toc198537010)

[Рисунок 6 Следующие 3 таблицы подробно 23](#_Toc198537011)

[Рисунок 7 Последние 2 таблицы подробно 24](#_Toc198537012)

[Рисунок 8 index 25](#_Toc198537013)

[Рисунок 9 registration 25](#_Toc198537014)

[Рисунок 10 user/profile 26](#_Toc198537015)

[Рисунок 11 user/document 27](#_Toc198537016)

[Рисунок 12 user/application 28](#_Toc198537017)

[Рисунок 13 user/exams 28](#_Toc198537018)

[Рисунок 14 user/exam/:id 29](#_Toc198537019)

[Рисунок 15 admin 29](#_Toc198537020)

[Рисунок 16 admin/user/list 30](#_Toc198537021)

[Рисунок 17 admin/student/profile 30](#_Toc198537022)

[Рисунок 18 admin/exam/planning 31](#_Toc198537023)

[Рисунок 19 admin/exam/create 31](#_Toc198537024)

[Рисунок 20 admin/exam/scheduled 32](#_Toc198537025)

[Рисунок 21 user/exam/start-page/:id 32](#_Toc198537026)

[Рисунок 22 user/exam/start-page/:id + modal 33](#_Toc198537027)

[Рисунок 23 user/exam/start/:id 33](#_Toc198537028)

[Рисунок 24 user/exam/student/:id/:id 34](#_Toc198537029)

Приложение 1. Архитектура серверной платформы

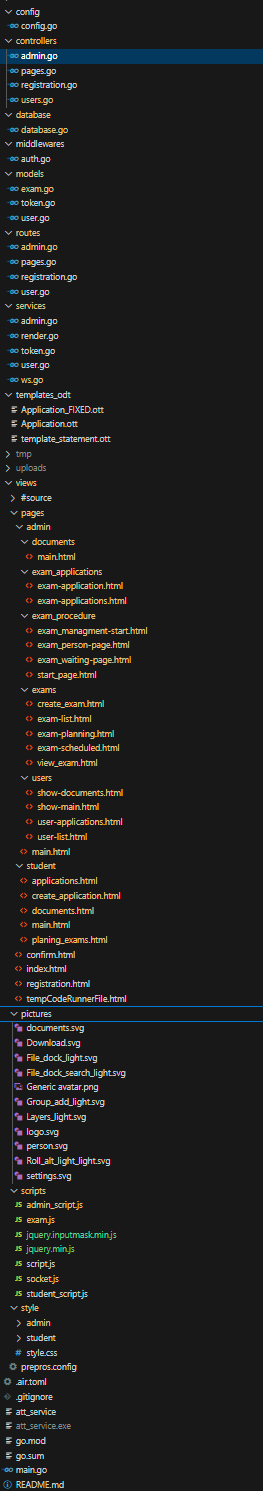


Рисунок 1 Архитектура системы на Go + PostgreSQL

Приложение 2. Структура базы данных

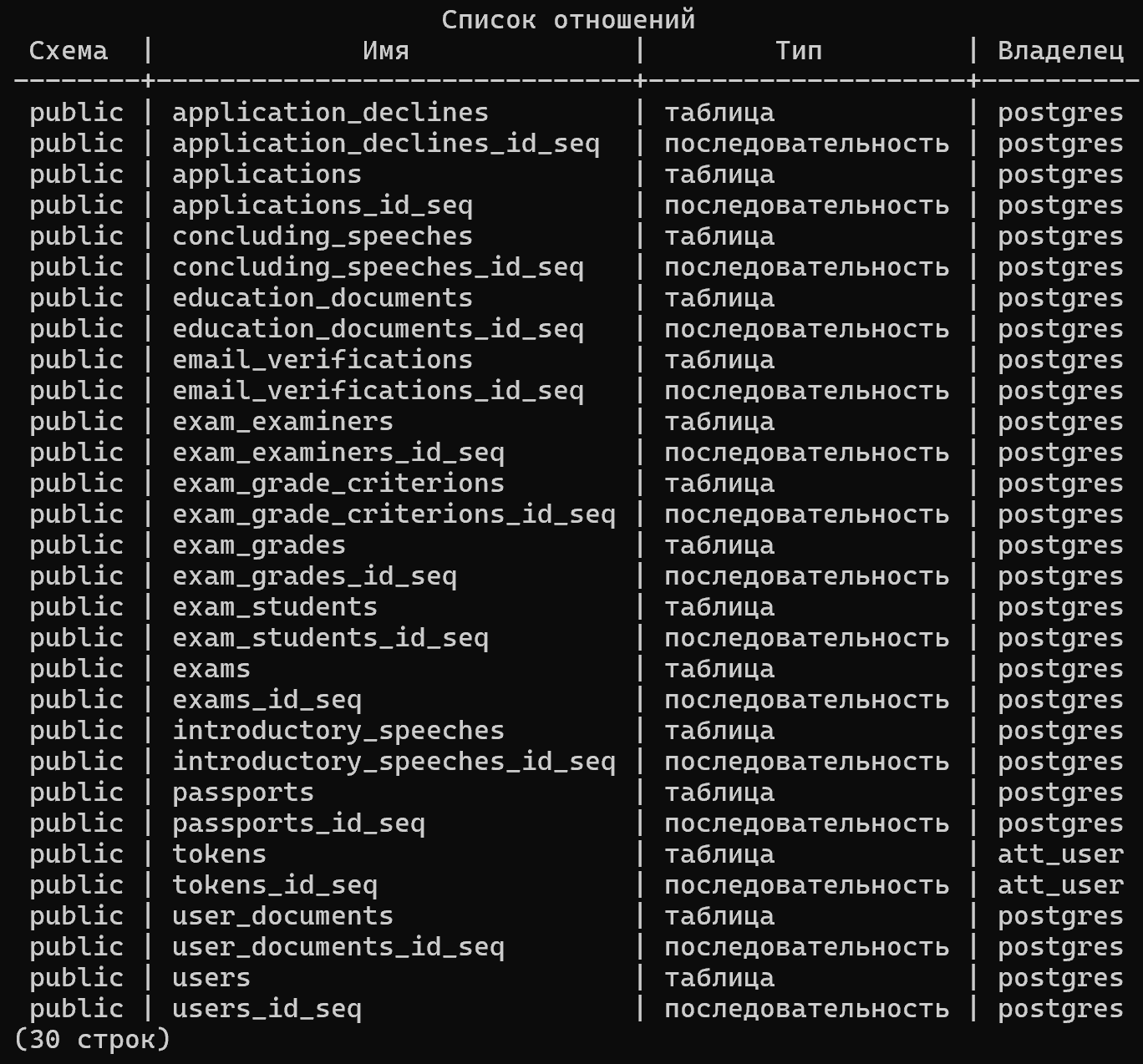


Рисунок 2 Общая структура

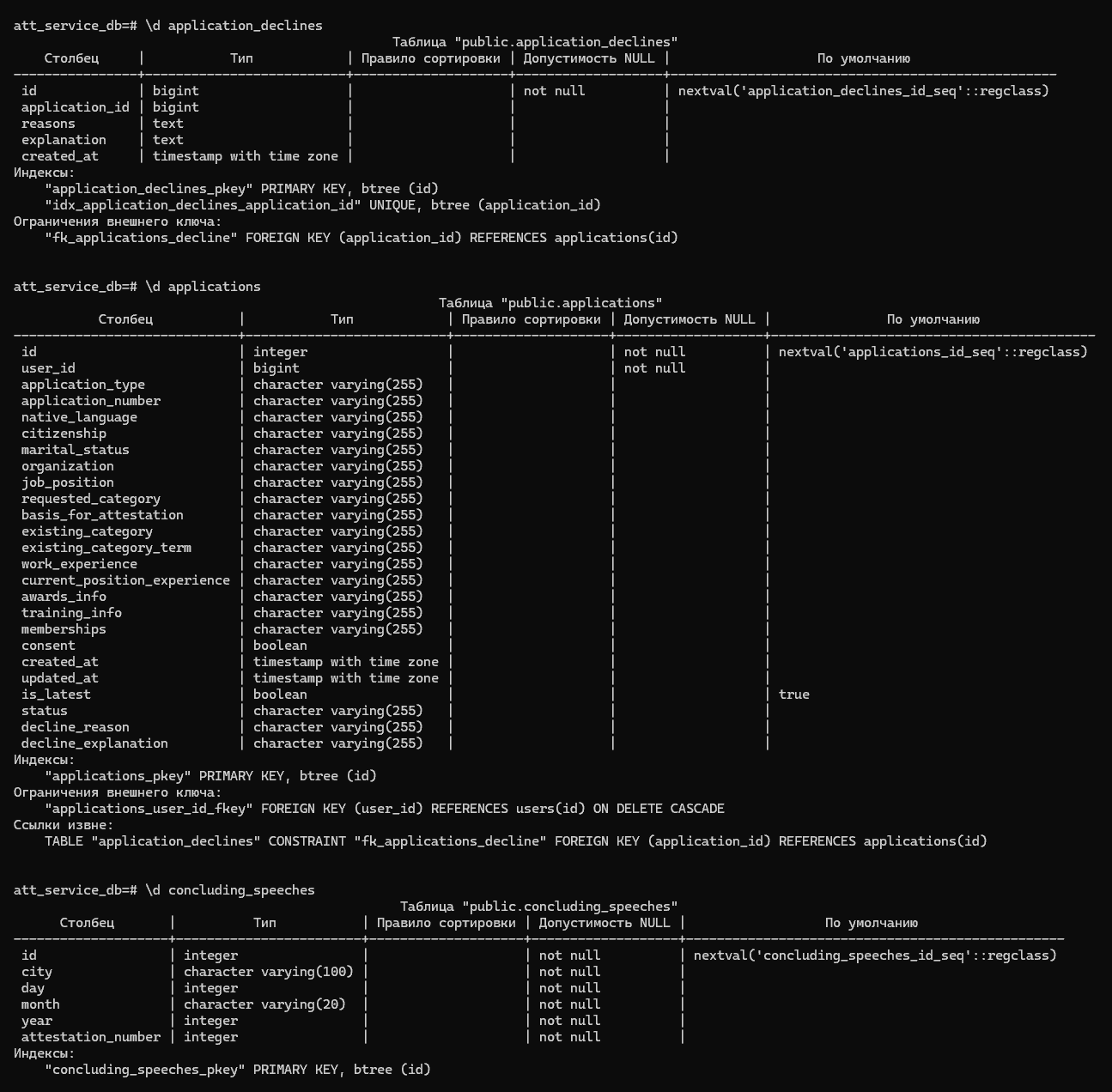


Рисунок 3 Первые 3 таблицы подробно

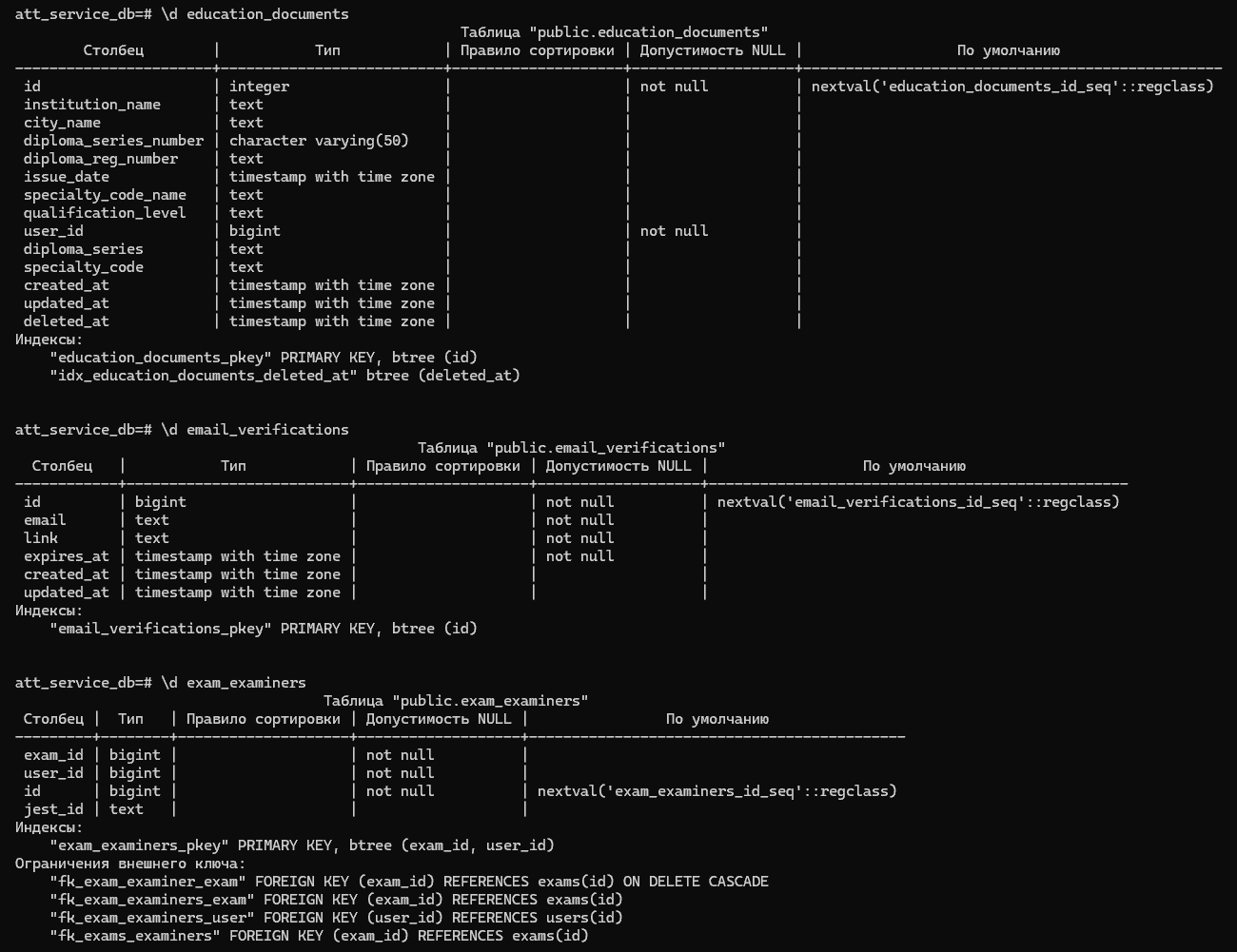


Рисунок 4 Следующие 3 таблицы подробно

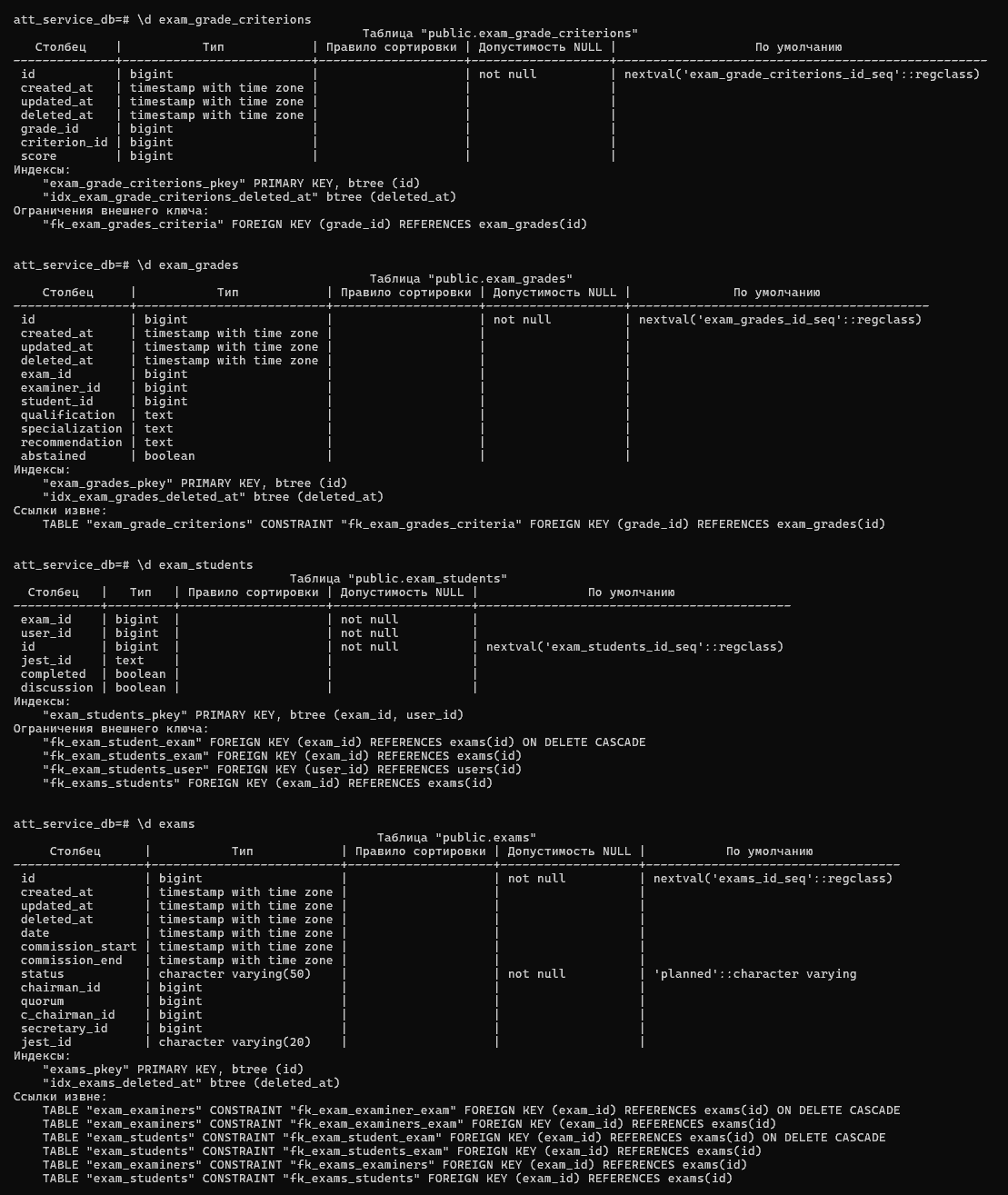


Рисунок 5 Следующие 4 таблицы подробно



Рисунок 6 Следующие 3 таблицы подробно

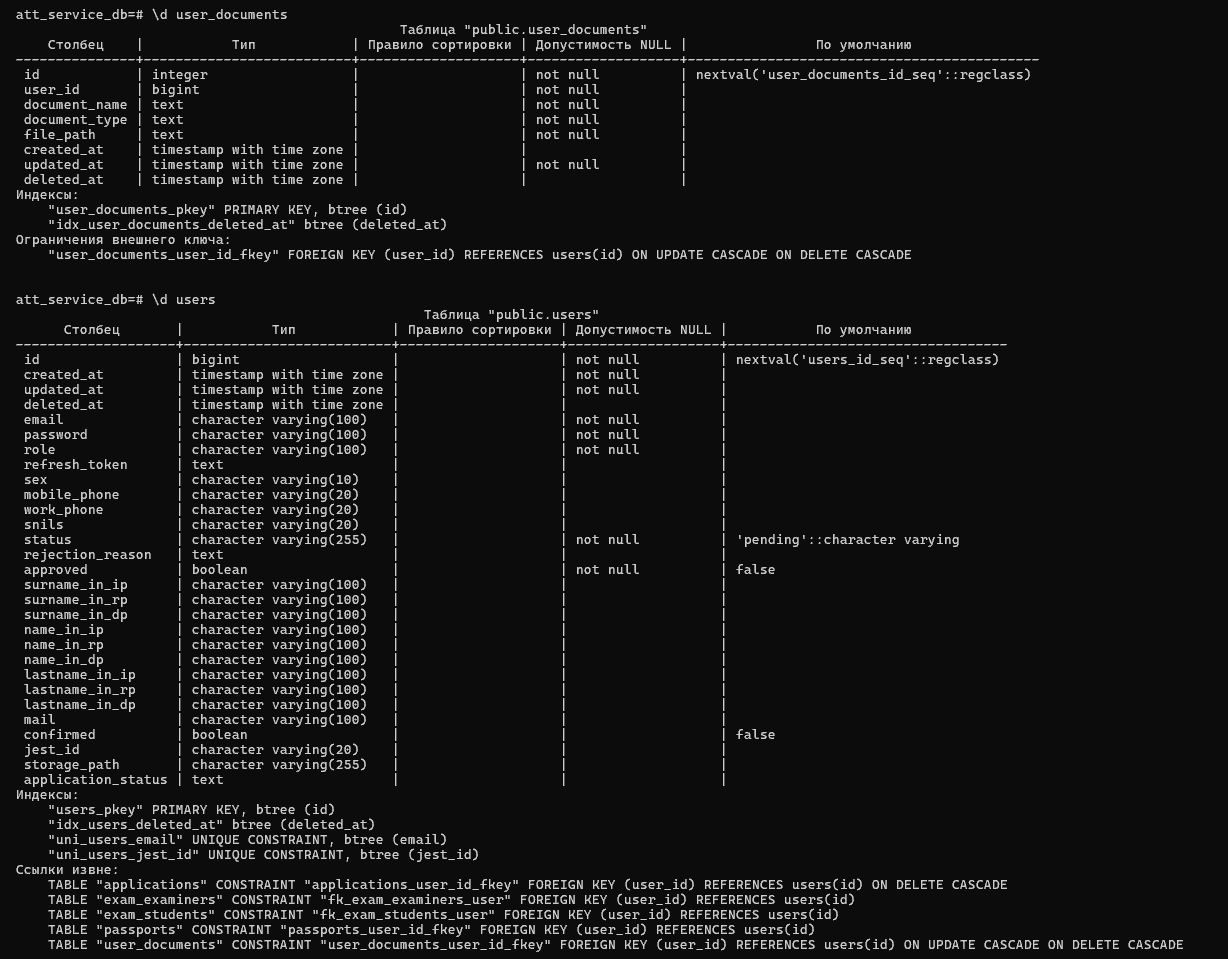


Рисунок 7 Последние 2 таблицы подробно

Приложение 3. Интерфейс платформы

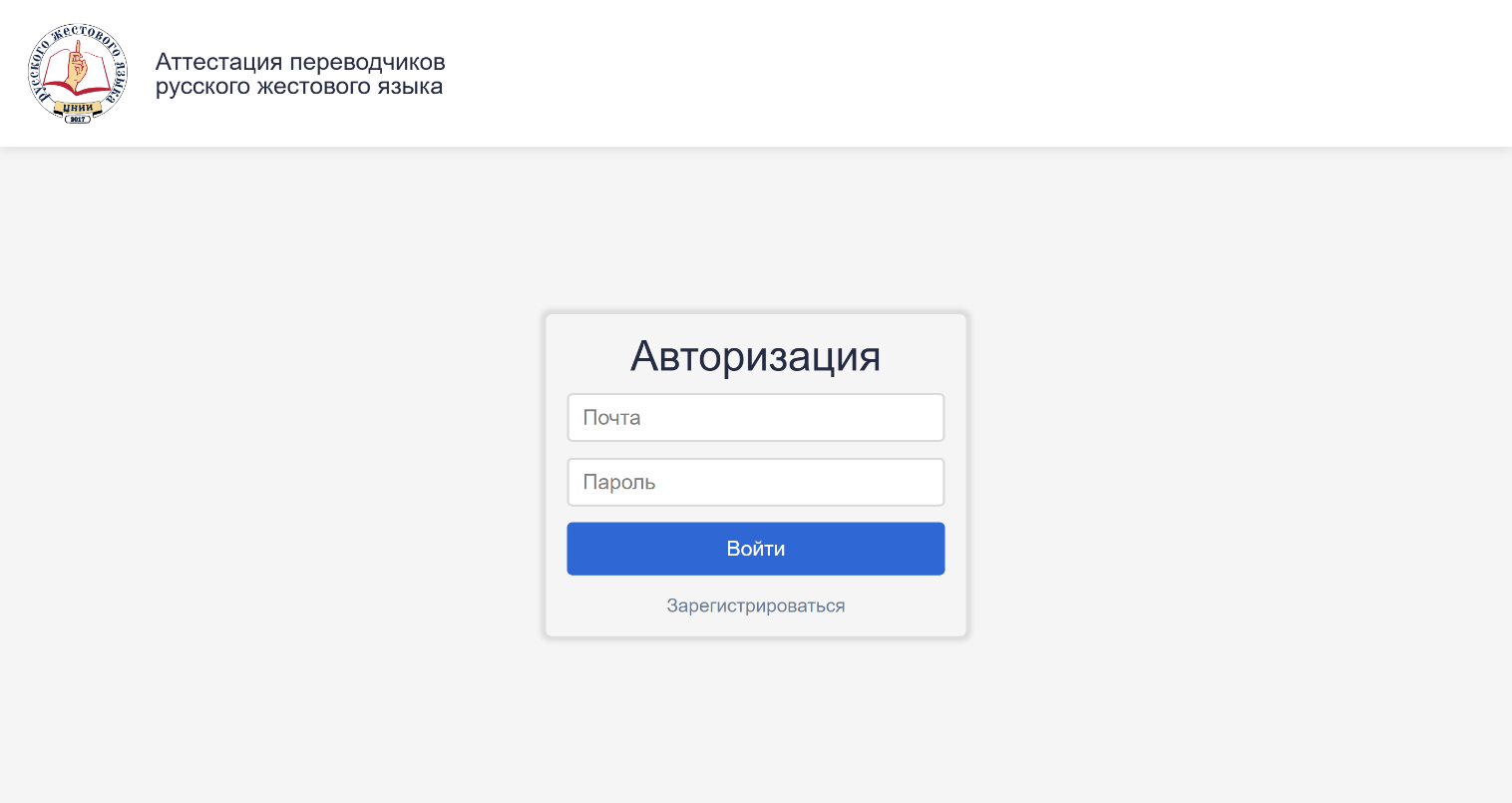


Рисунок 8 index

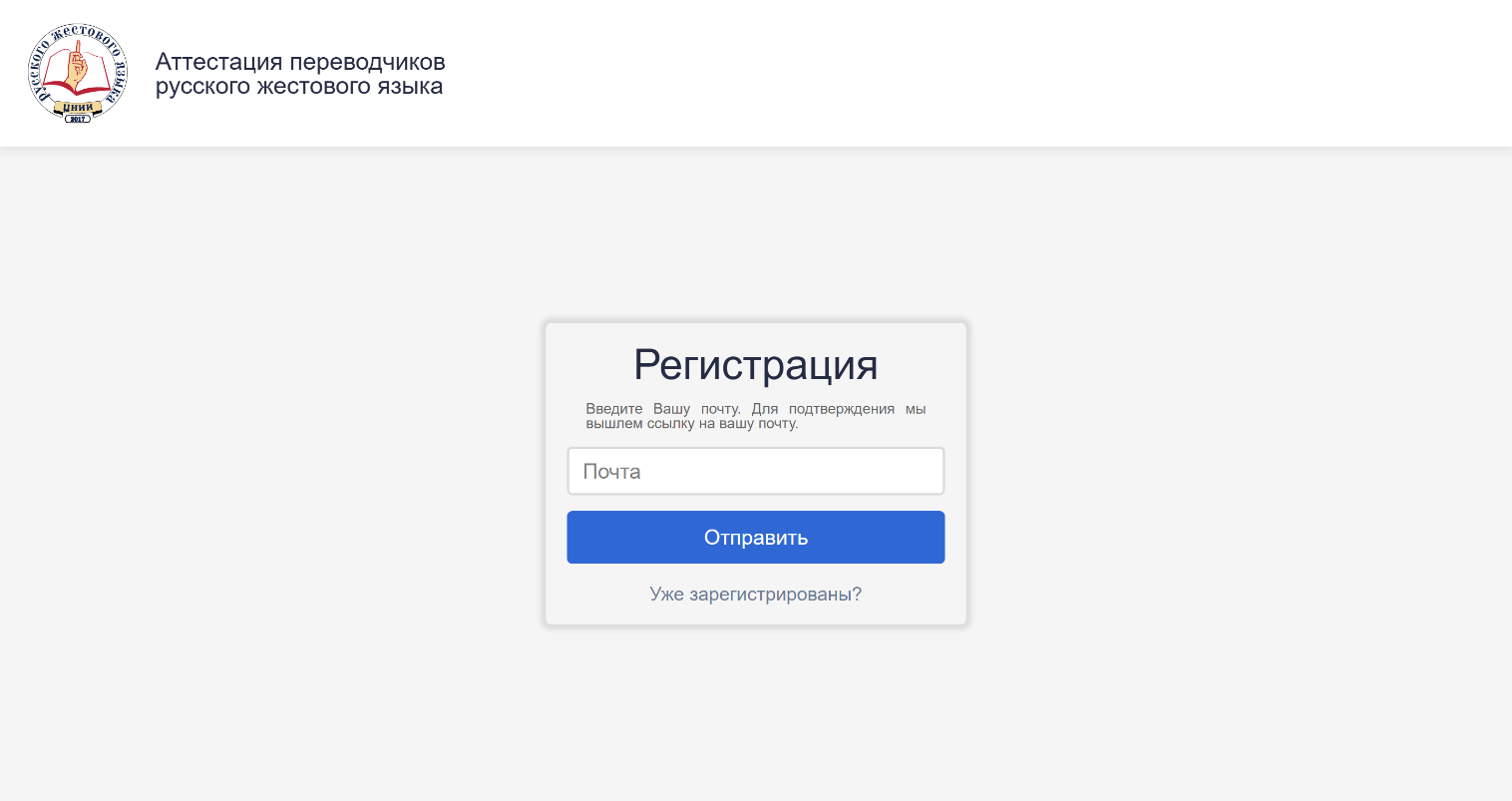


Рисунок 9 registration

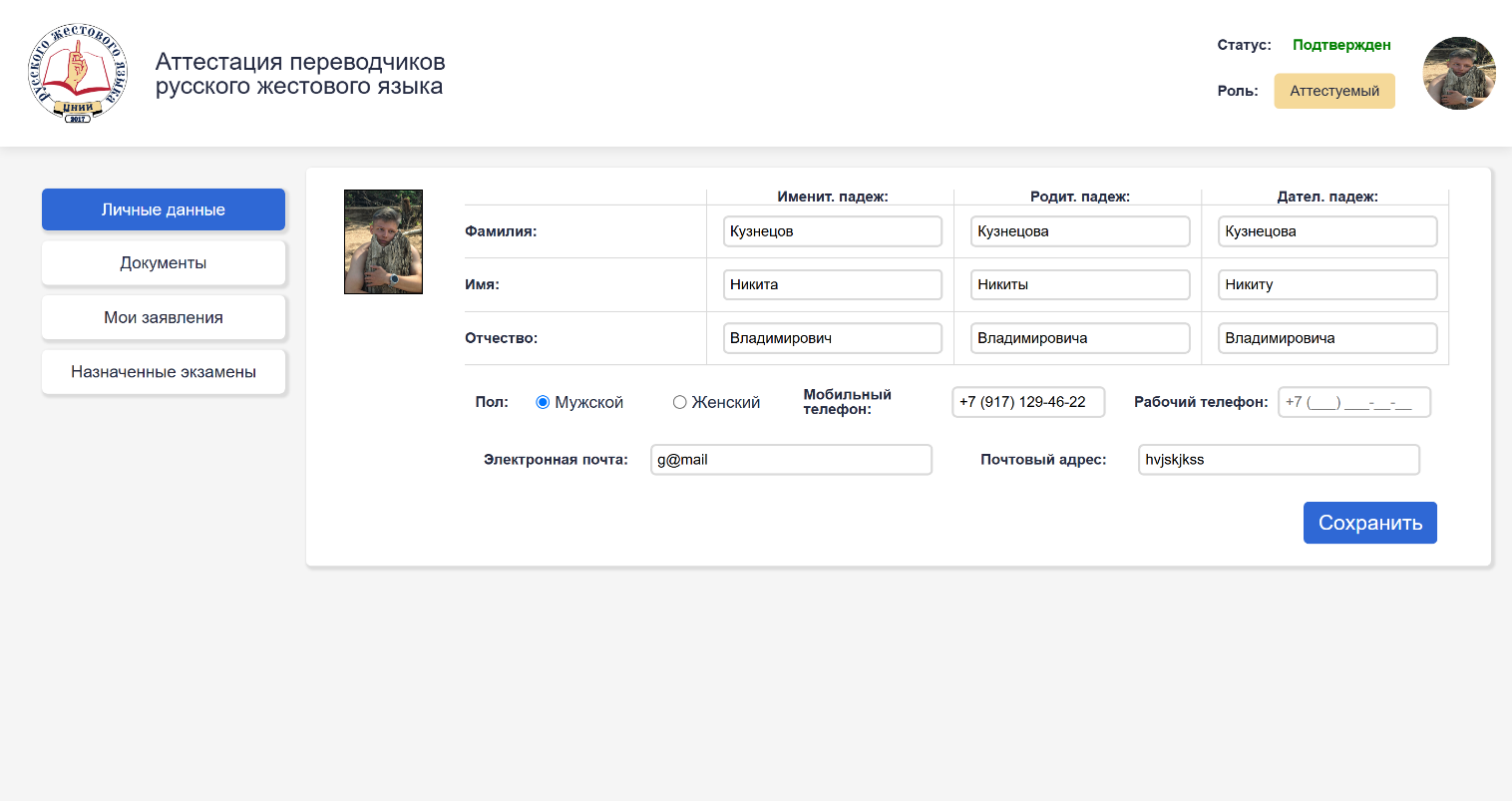


Рисунок 10 user/profile

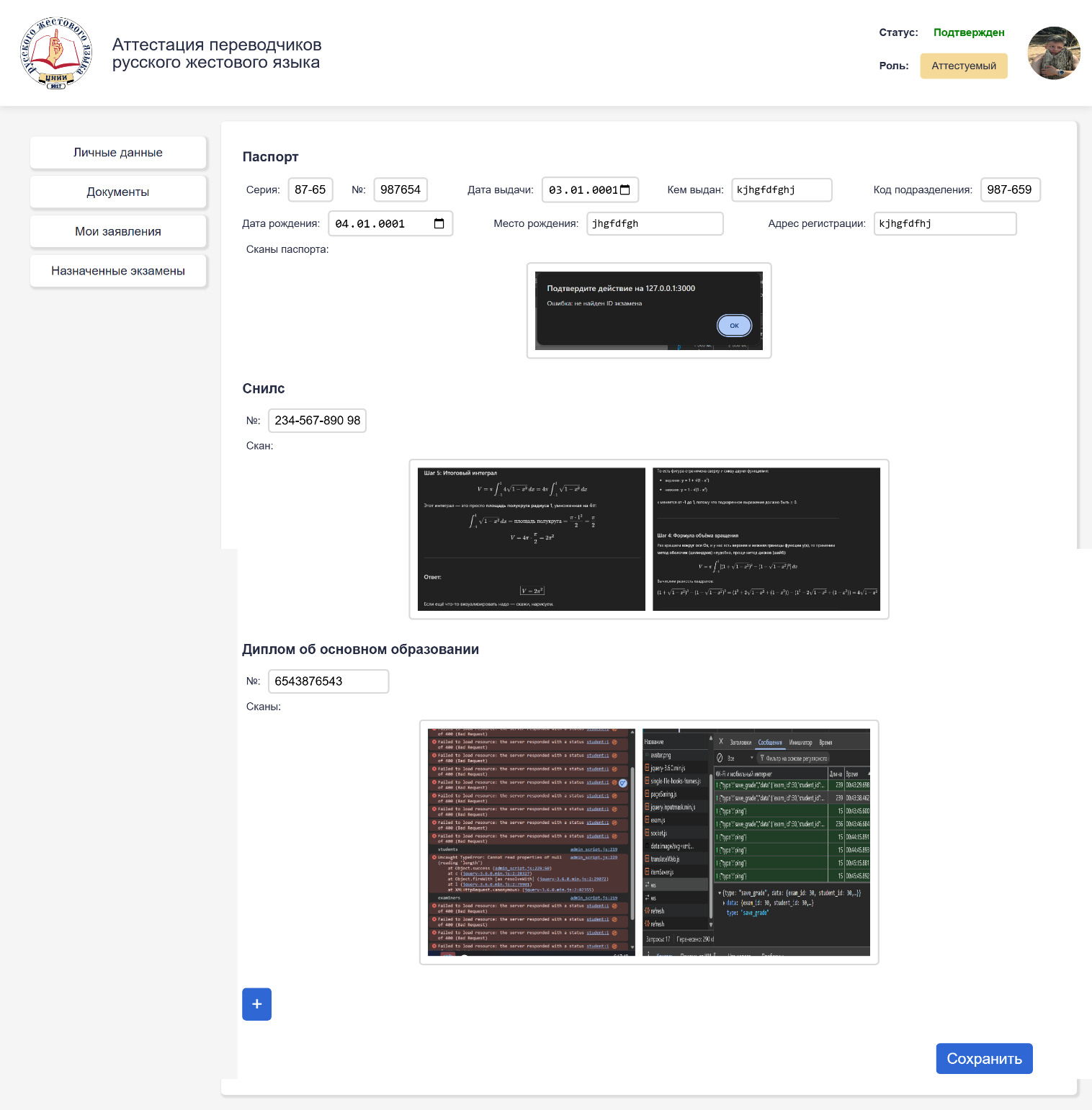


Рисунок 11 user/document

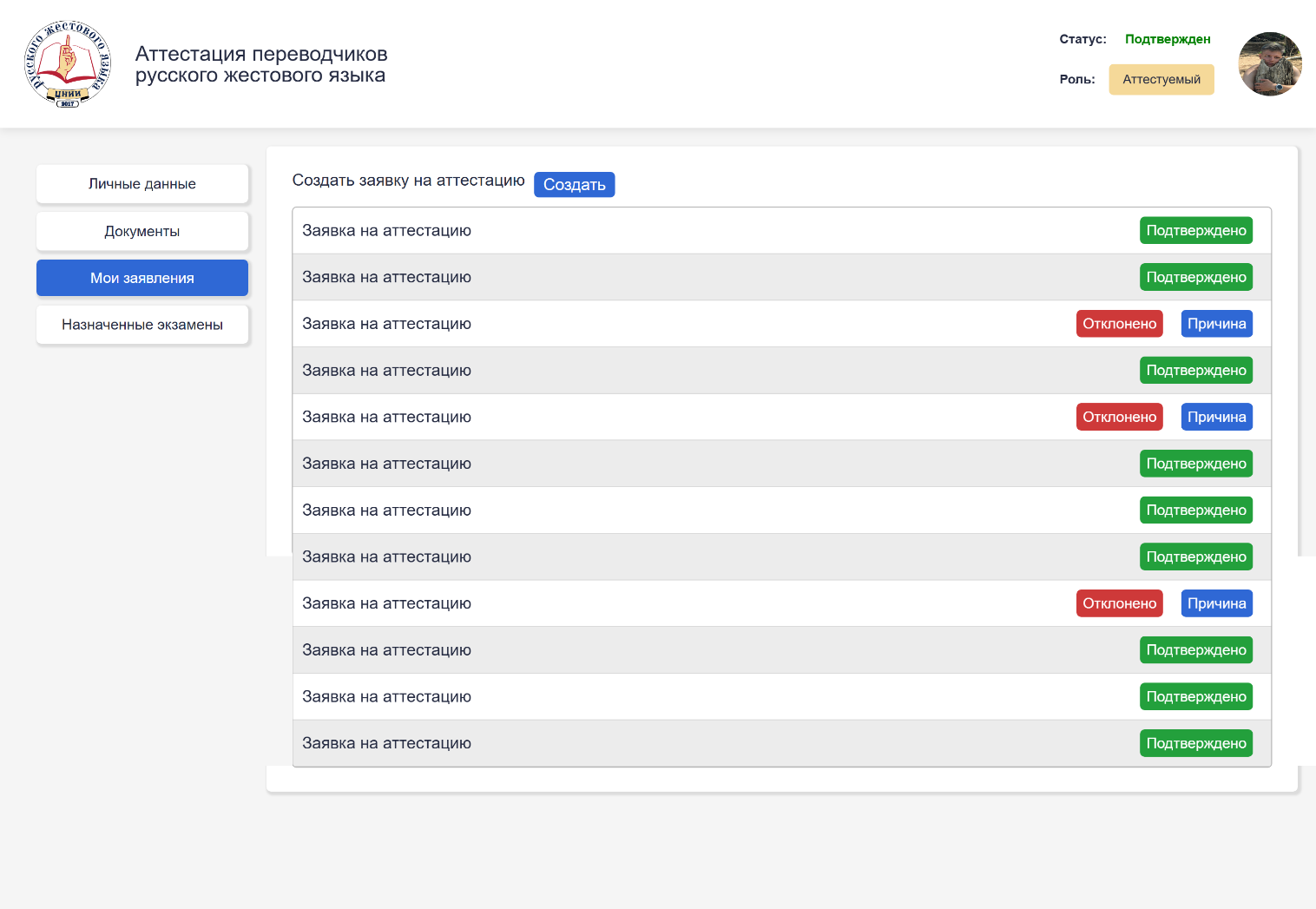


Рисунок 12 user/application

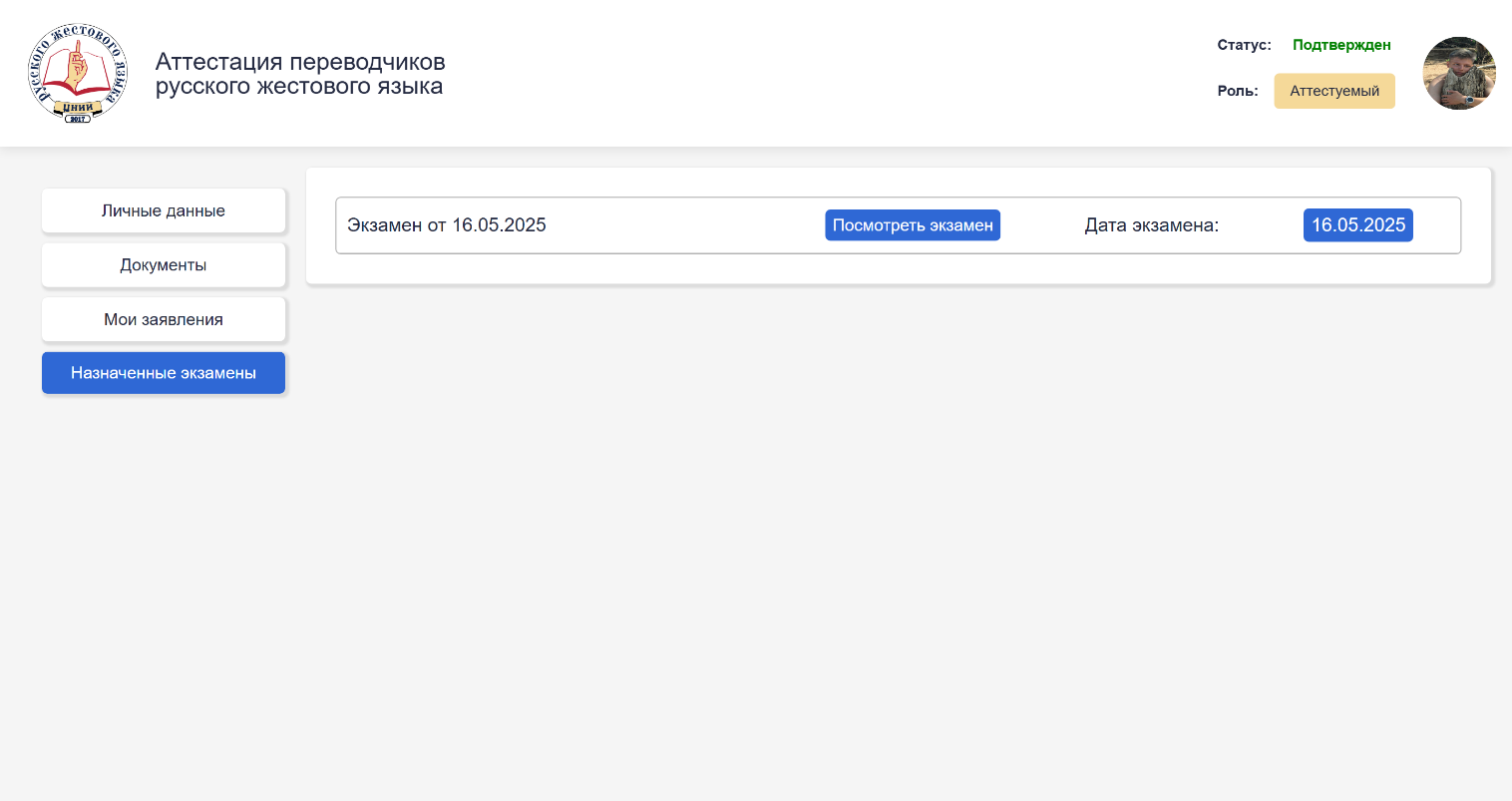


Рисунок 13 user/exams

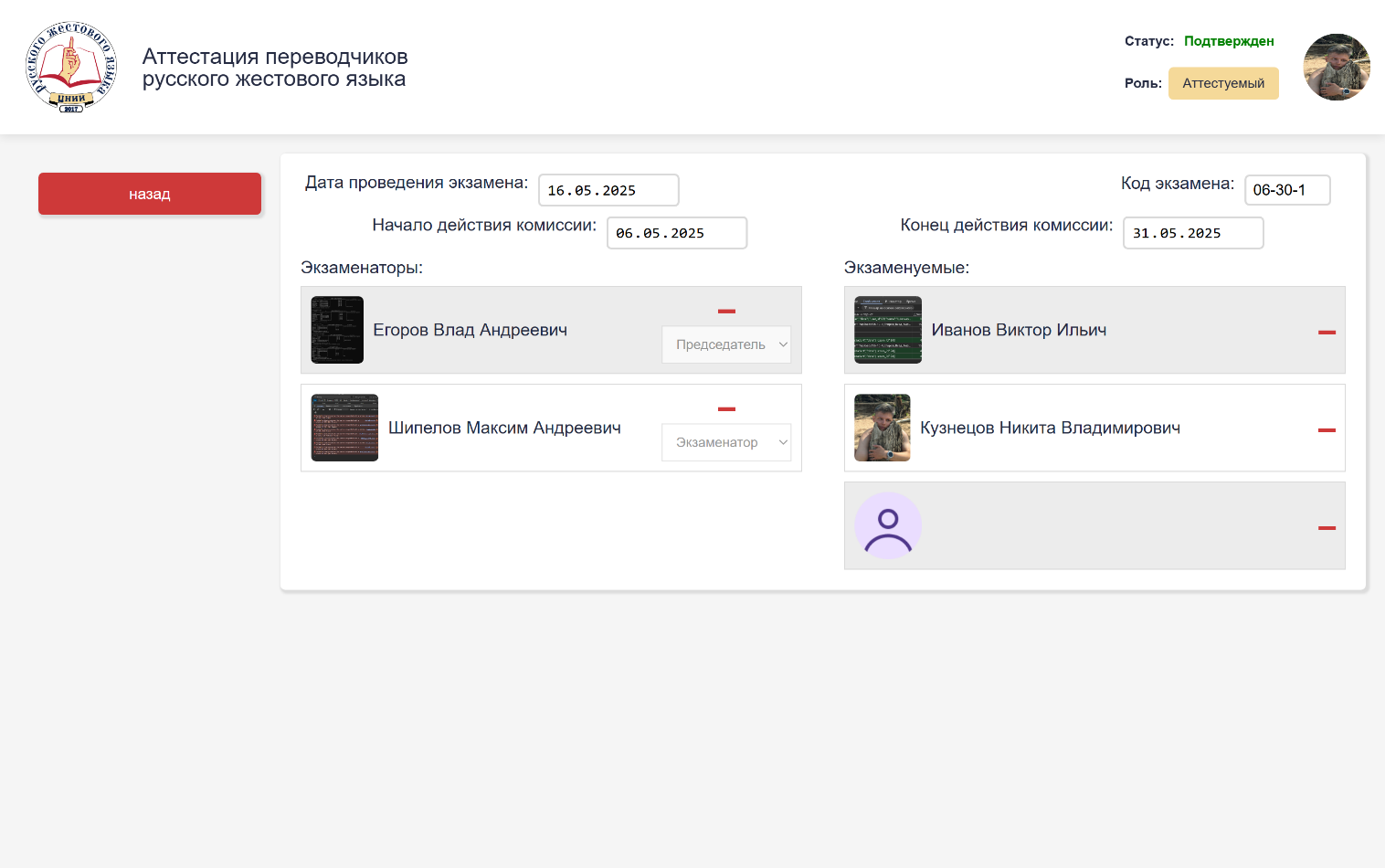


Рисунок 14 user/exam/:id

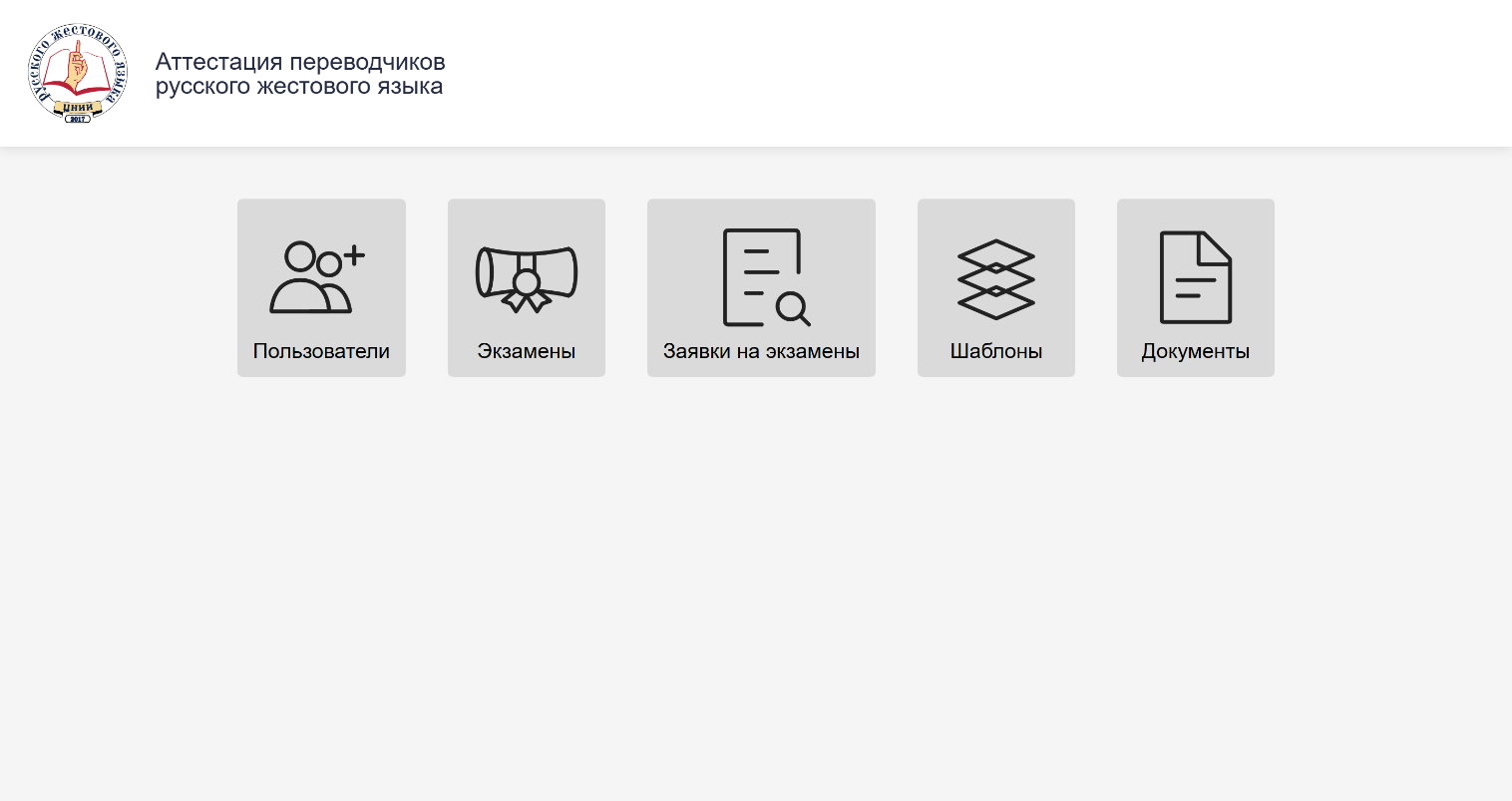


Рисунок 15 admin

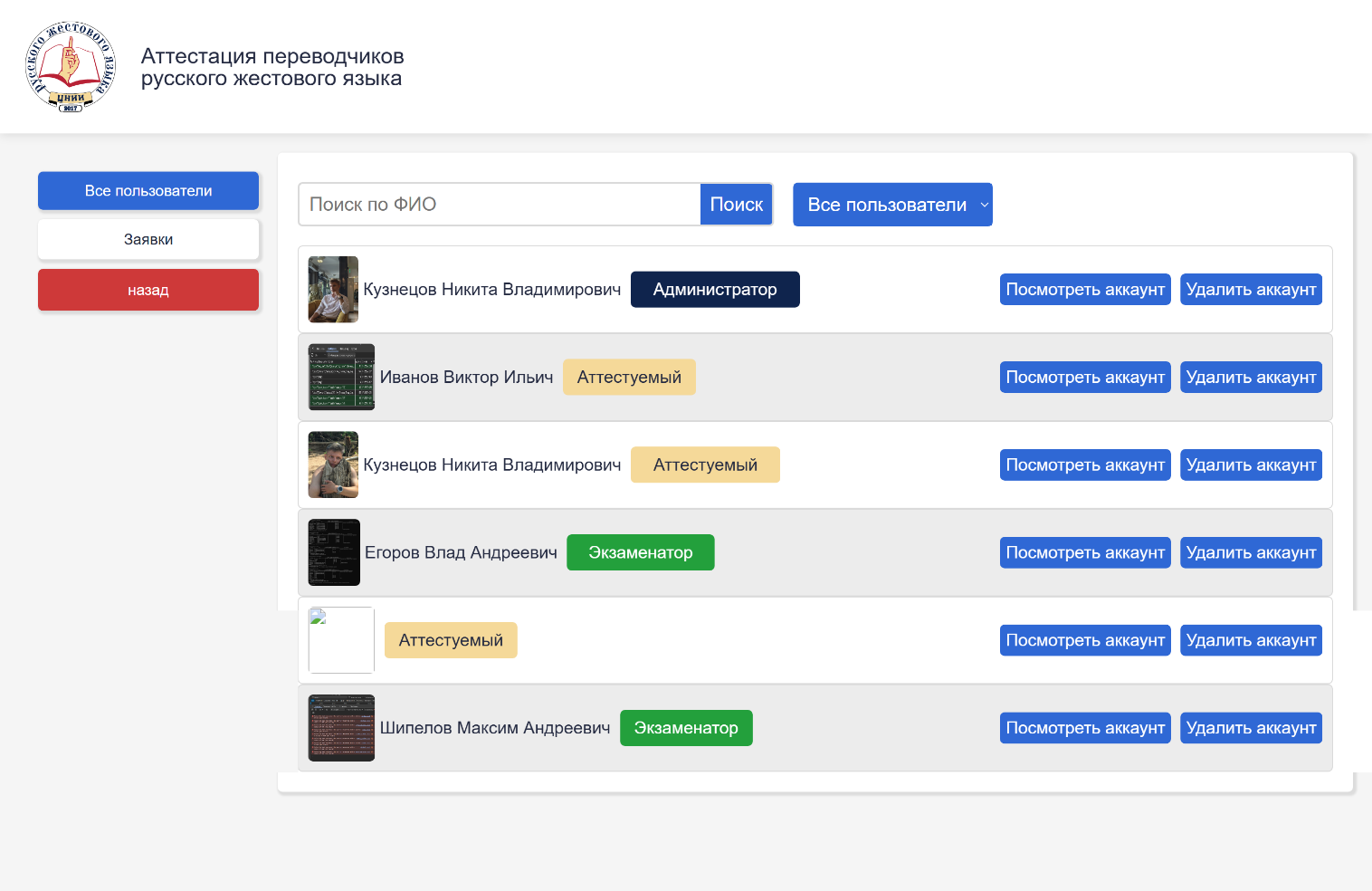


Рисунок 16 admin/user/list

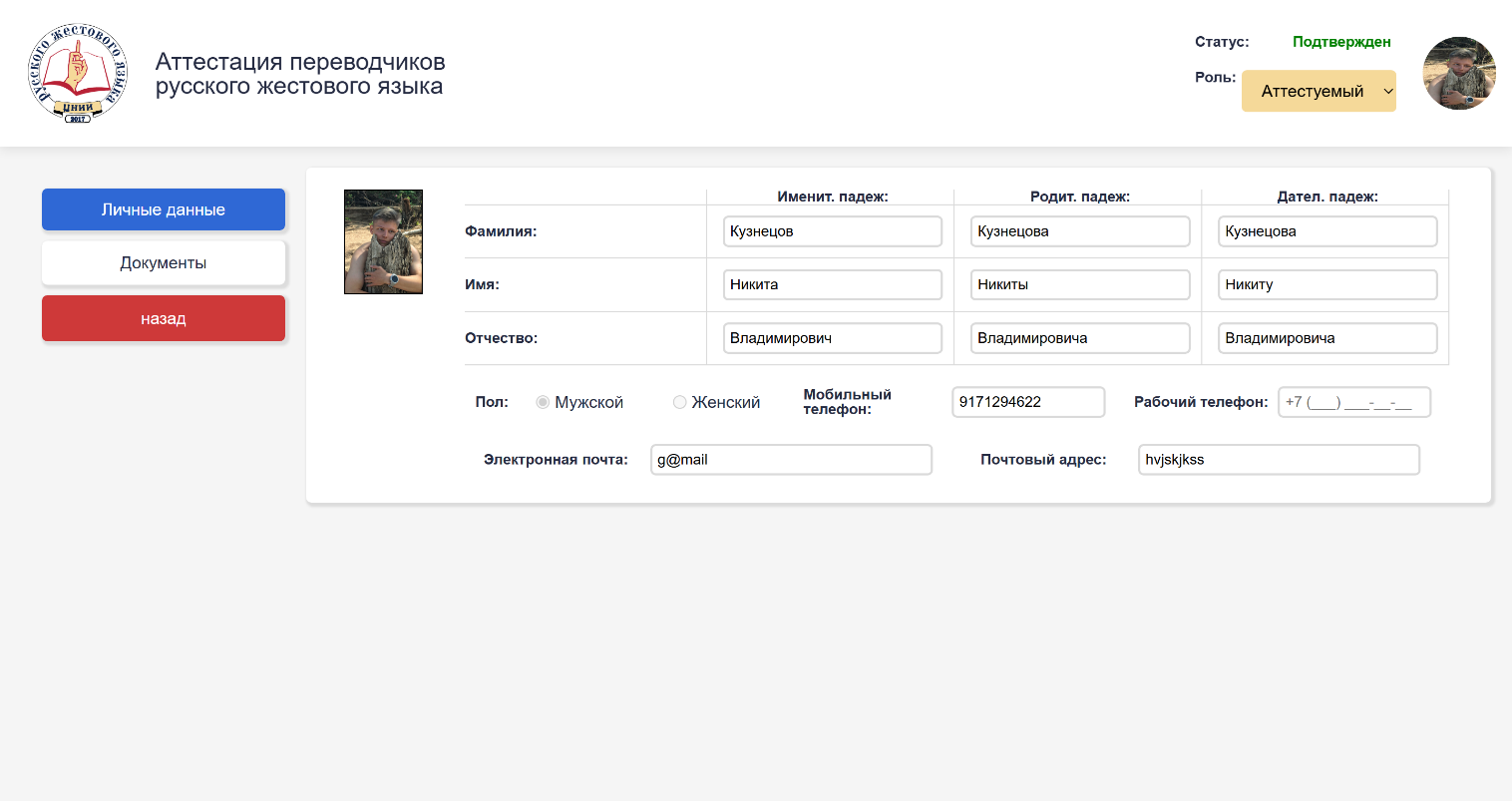


Рисунок 17 admin/student/profile

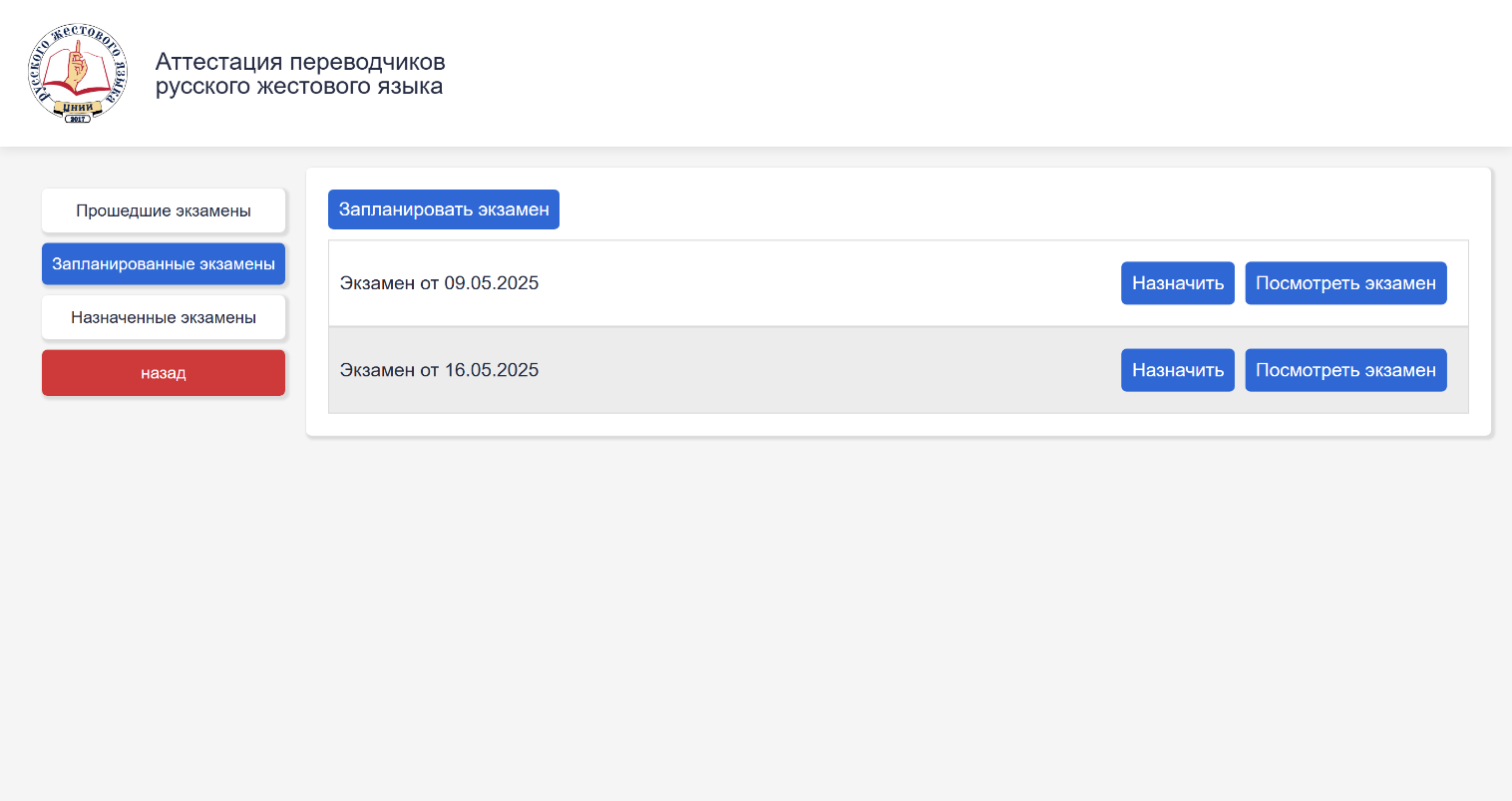


Рисунок 18 admin/exam/planning

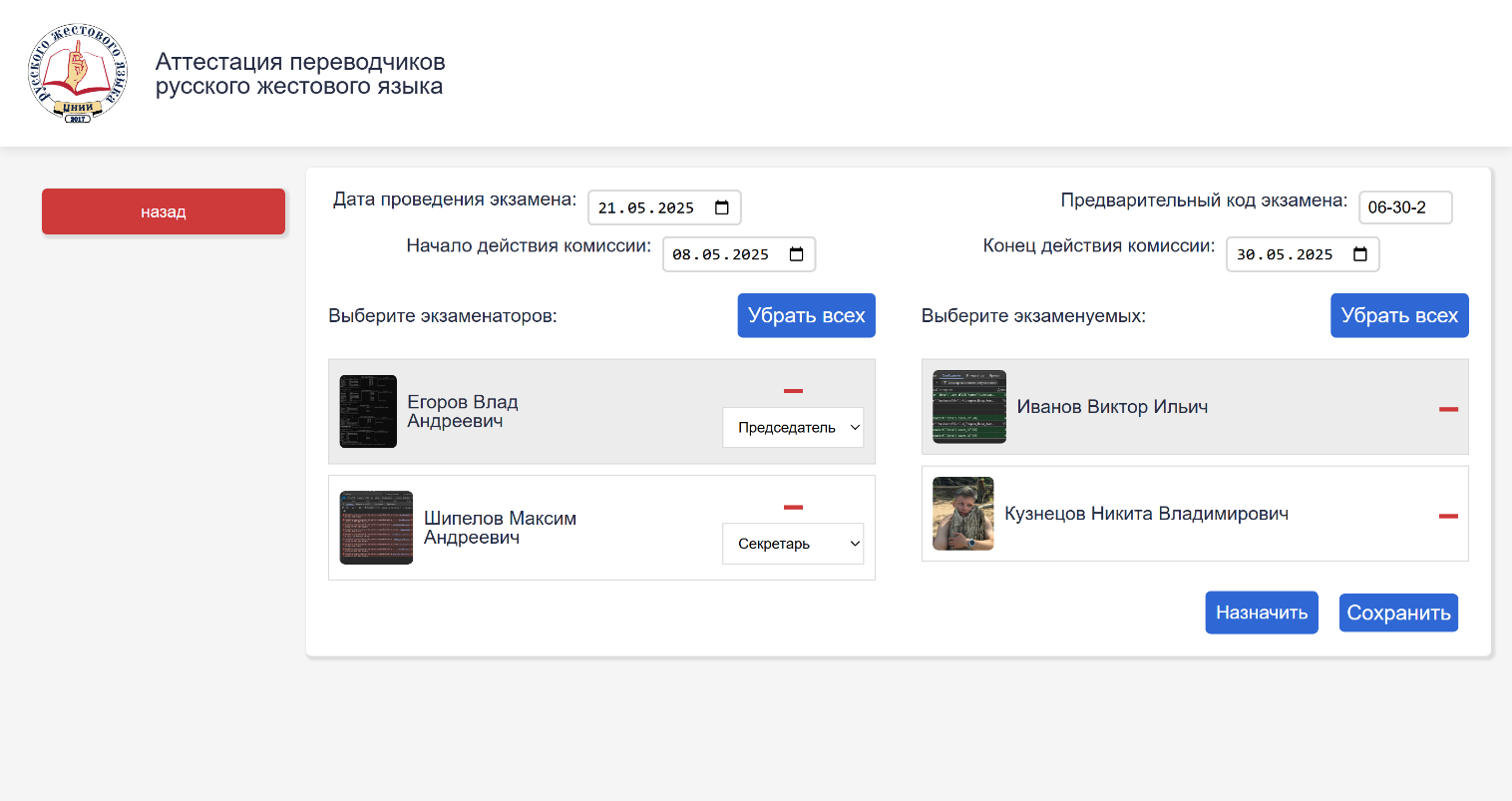


Рисунок 19 admin/exam/create

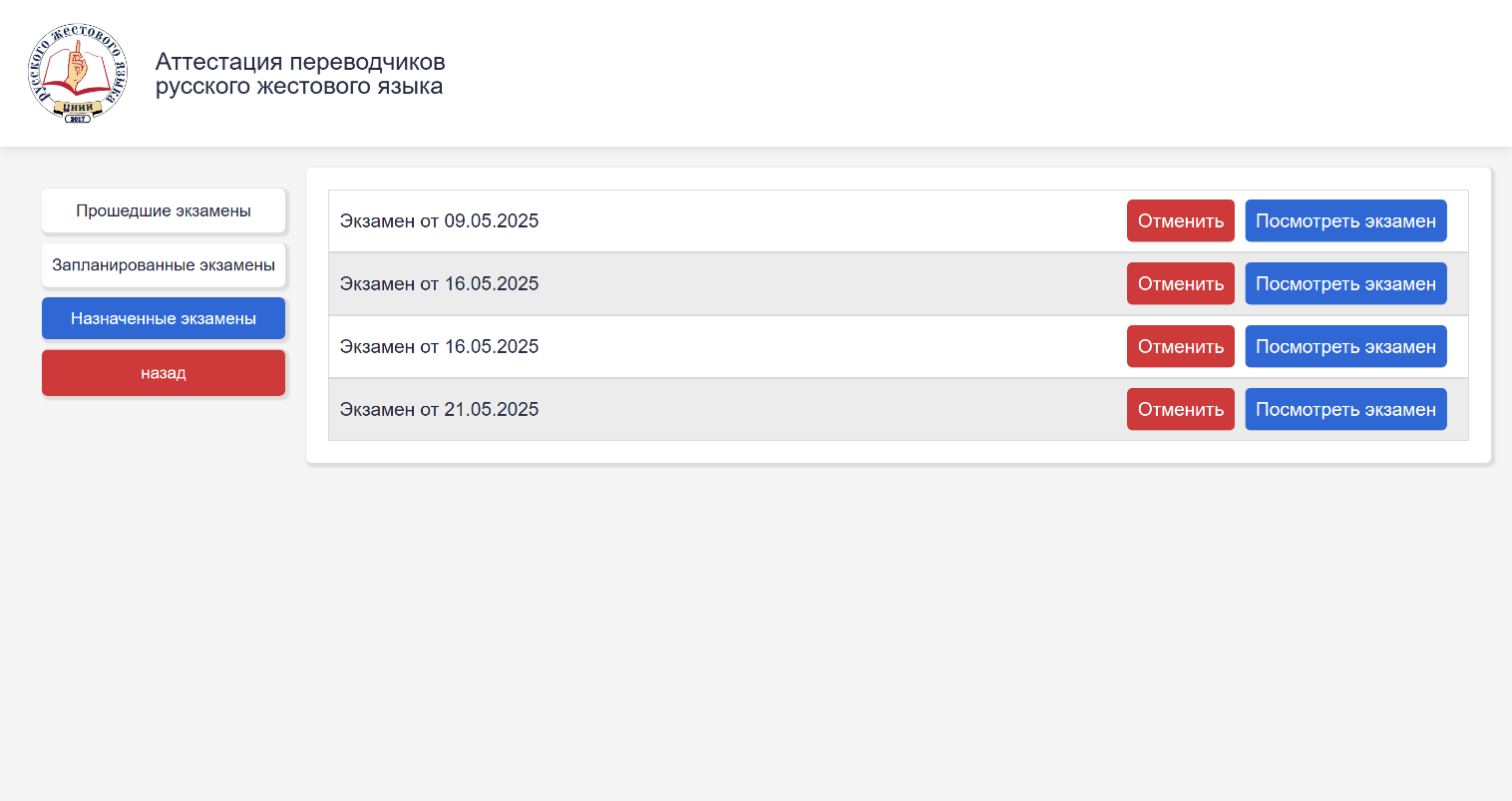


Рисунок 20 admin/exam/scheduled

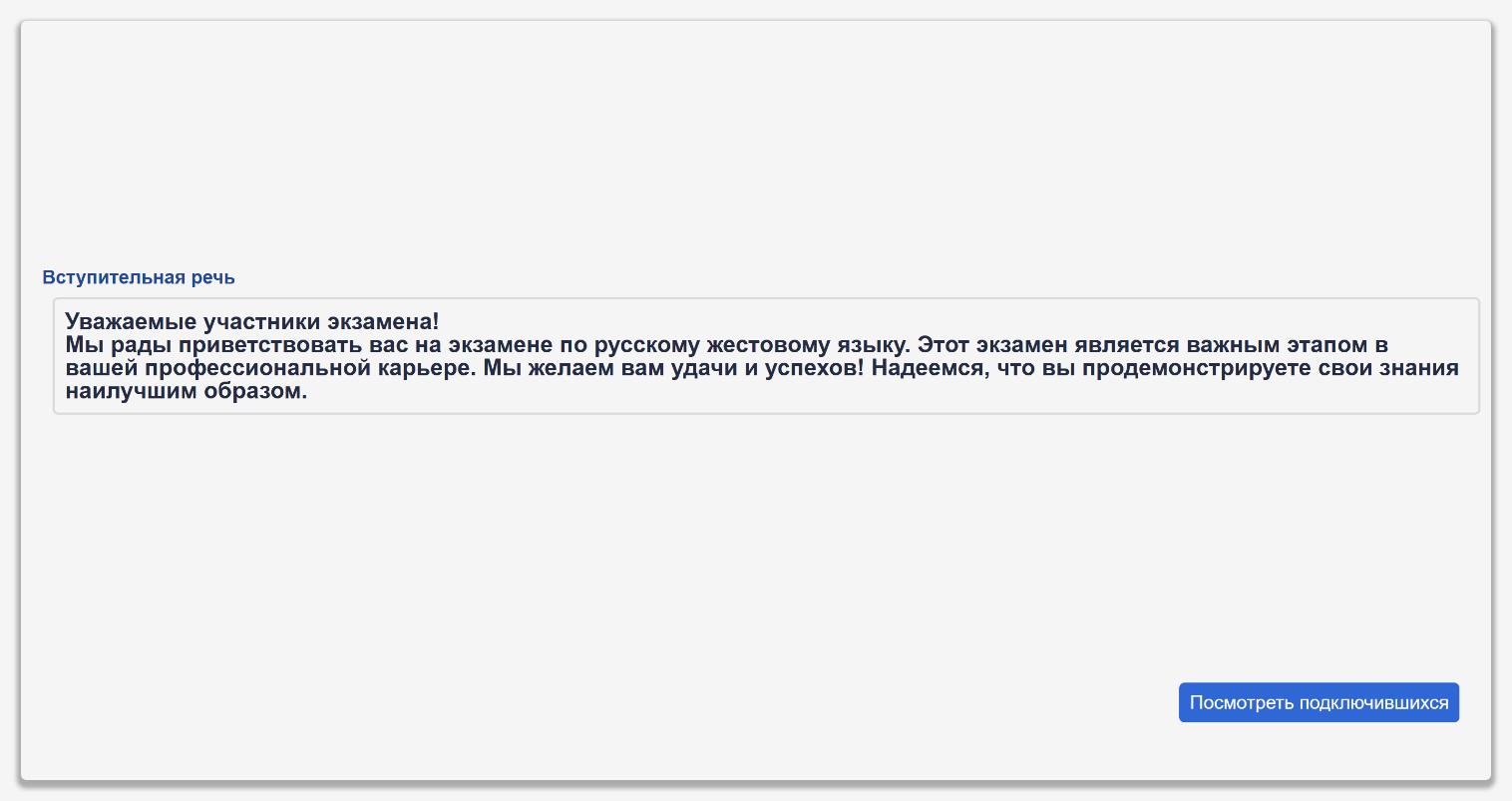


Рисунок 21 user/exam/start-page/:id

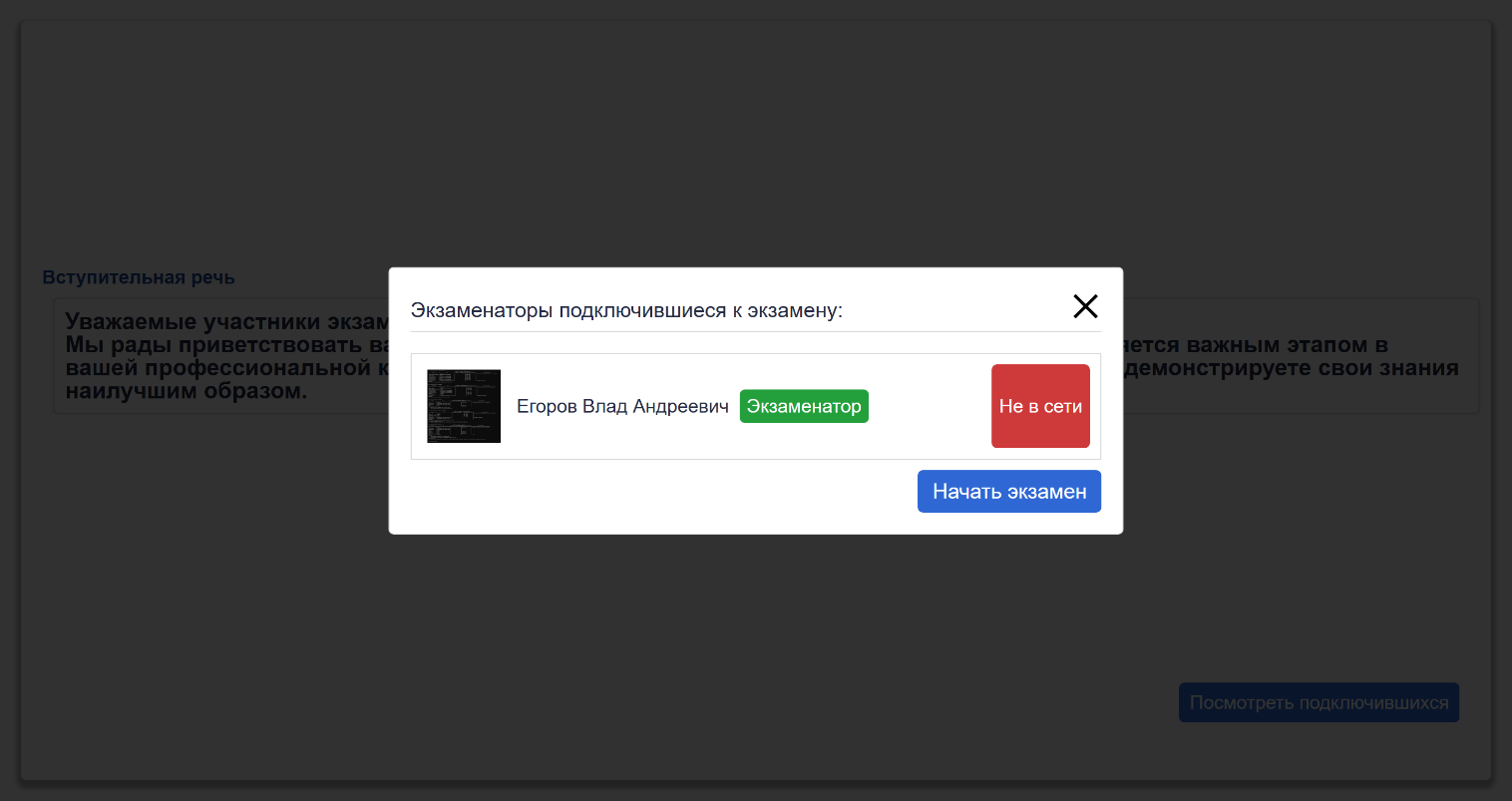


Рисунок 22 user/exam/start-page/:id + modal

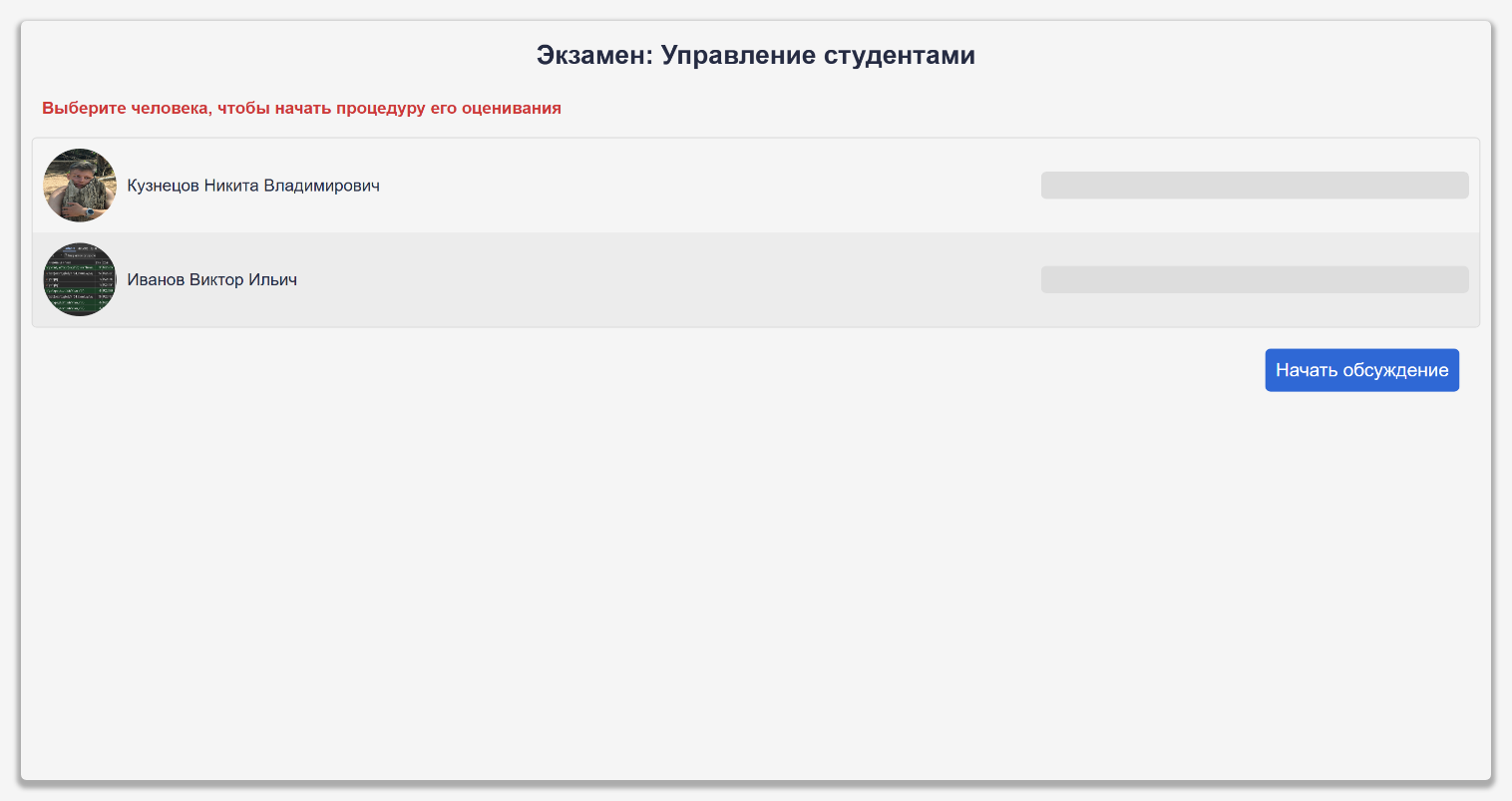


Рисунок 23 user/exam/start/:id

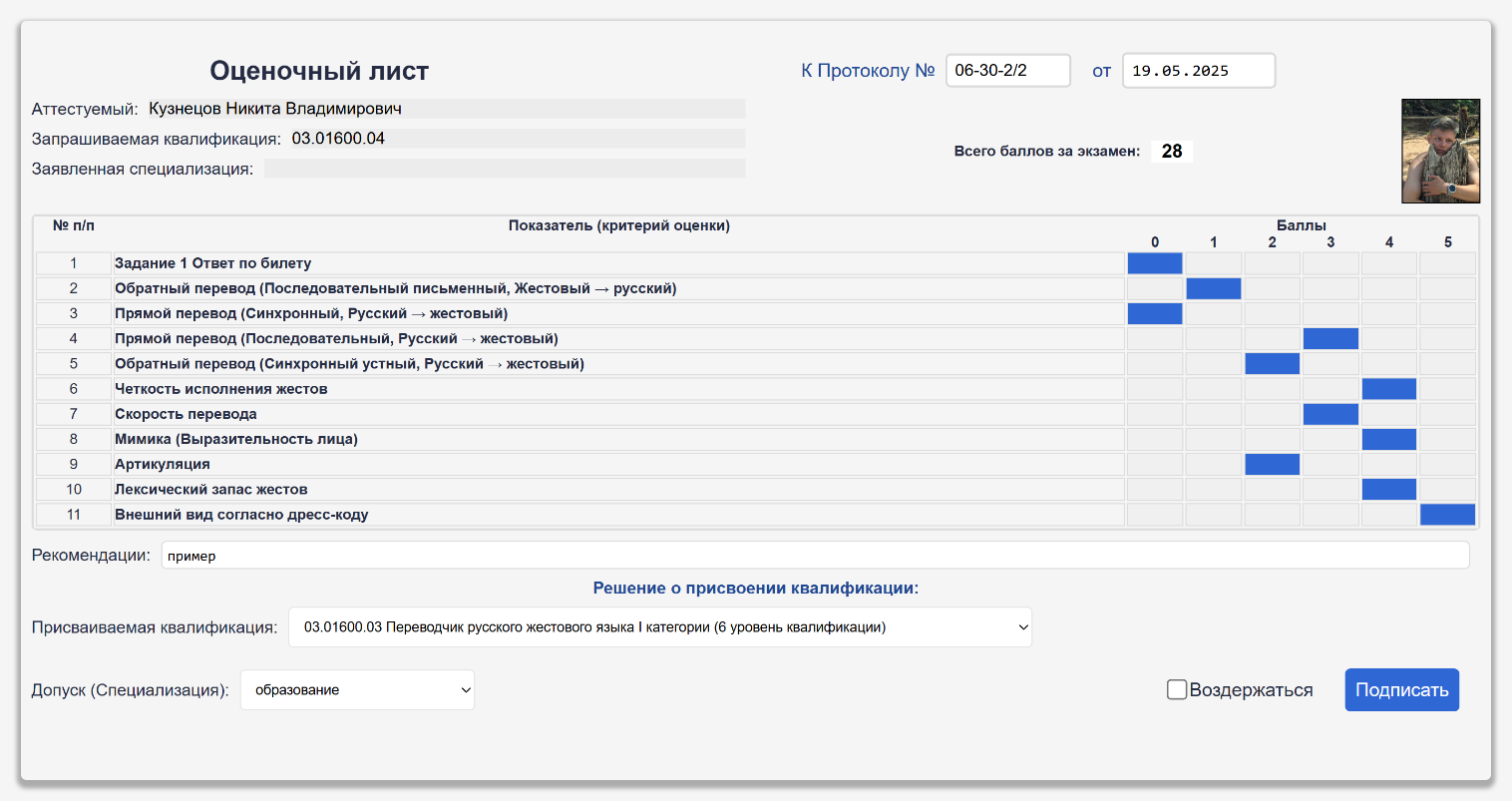


Рисунок 24 user/exam/student/:id/:id