Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

Работа выполнена в СПБ «DeCode»

СОГЛАСОВАНО

Начальник отдела ОНиПКРС

_____ Е.М. Димитриади
(подпись)

«____ » _____ 20___ г.

УТВЕРЖДАЮ

И. о. проректора
по научной работе

_____А.В. Космынин
(подпись)

«____ » _____ 20___ г.

Декан ФКТ

_____И.А. Трещев
(подпись)

«____ » _____ 20___ г.

«Электронная журнал посещаемости»

Комплект *конструкторской / проектной* документации

Руководитель СПБ _____ *Е.Б. Абарникова*
(подпись, дата)

Руководитель проекта _____ *Е.Б. Абарникова*
(подпись, дата)

Комсомольск-на-Амуре 2024

Карточка проекта

Название	*«Электронный журнал посещаемости»*
Тип проекта	Тип проекта: техническое творчество (инициативный)
Исполнители	Студент А.В. Зайцев – 4ВТм-1
Срок реализации	Сентябрь 2024 – октябрь 2024

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

ЗАДАНИЕ
на разработку

Название проекта: Разработка интерактивного сервиса «Электронный журнал посещаемости»

Назначение: проект направлен на повышение цифровизации учебного процесса университета путем внедрения интерактивного сервиса «Электронный журнал посещаемости». Данный сервис предоставляет набор возможностей: создание QR-кодов для автоматизированной отметки посещения пары студентом, ручная отметка посещения пары, выставление оценки и добавление заметки к конкретному студенту

Область использования: проект направлен на создание интерактивного сервиса, который будет внедрен в информационную среду университета. Данный интерактивный сервис будет содержать информацию о посещаемости студентов. Наличие такого интерактивного сервиса позволит уменьшить время, затраченного на отметку студентов во время пары, а также снизить затраты на покупку бумажных учебных журналов

Функциональное описание проекта: _____

- возможность автоматизации отметки посещения студентов
- возможность ведения журнала в реальном времени
- возможность синхронизации информации с удаленным хранилищем
- возможность обеспечения просмотра своевременной и полной статистики по посещению и успеваемости студентов

Техническое описание устройства: _____

_____

_____

Требования:

Функциональные требования:

- сервис должен автоматизировать отметку посещения студентов;
- сервис должен позволять ведение журнала в реальном времени;
- сервис должен синхронизировать информацию с удаленным хранилищем;
- сервис должен обеспечивать просмотр своевременной и полной статистики по посещению и успеваемости студентов;

Нефункциональные требования:

- сервис должен быть легким в использовании и понимании для потенциальных пользователей;
- сервис должен быть доступен на разных устройствах (ПК, планшеты, мобильные устройства);
- сервис должен соответствовать брендбуку университета

План работ:

Наименование работ	Срок
Исследование требований	02.09-03.09
Создание прототипа	04.09-04.10
Разработка клиентской части	05.10-10.10
Разработка серверной части	11.10-20.10
Тестирование и отладка	21.10-25.10
Публикация	30.10

Комментарии:

Разработка проекта велась с использованием agile-методологий, у каждого члена команды была своя задача, благодаря этому выполнение некоторых этапов велось параллельно

Перечень графического материала:

1. Принципиальная схема;
2. Чертежи изделия (или трехмерные модели изделия);
3. Внешний вид изделия;
4. Блок-схема алгоритмов (при наличии управляющих программ);

_____

_____

_____

_____

Руководитель проекта _____ *Е.Б. Абарникова*

(подпись, дата)

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

ПАСПОРТ

Разработка интерактивного сервиса «Электронный журнал посещаемости»

Руководитель проекта _____ *Е.Б. Абарникова*
(подпись, дата)

Комсомольск-на-Амуре 2024

Содержание

1 Общие положения

Настоящий паспорт является документом, предназначенным для ознакомления с основными техническими характеристиками, устройством, правилами установки и эксплуатации проекта «Электронный журнал посещаемости» (далее «сервис»).

1.1 Наименование изделия

Полное наименование изделия – «Электронный журнал посещаемости».

1.2 Наименования документов, на основании которых ведется проектирование изделия

Проектирвание «Электронного журнала посещаемости» осуществляется на основании требований и положений следующих документов:

- задание на разработку.

1.3 Перечень организаций, участвующих в разработке изделия

Заказчиком проекта «Электронный журнал посещаемости» является Федеральное государственное бюджетное образовательное учреждение высшего образования «Комсомольский-на-Амуре государственный университет» (далее заказчик), находящийся по адресу: 681013, Хабаровский край, г. Комсомольск-на-Амуре, Ленина пр-кт., д. 17.

Исполнителем проекта «Электронный журнал посещаемости» являются Конструкторы студенческого конструкторского студенческого проектного бюро «DeCode» (далее СПБ «DeCode»), студент группы 4ВТм-1 А.В. Зайцев.

1.4 Сведения об использованных при проектировании нормативно-технических документах

При проектировании использованы следующие нормативно-технические документы:

ГОСТ 2.001-2013. Единая система конструкторской документации. Общие положения.

ГОСТ 2.102-2013. Единая система конструкторской документации. Виды и комплектность конструкторских документов.

ГОСТ 2.105-95. Единая система конструкторской документации. Общие требования к текстовым документам.

ГОСТ 2.610-2006. Единая система конструкторской документации. Правила выполнения эксплуатационных документов.

ГОСТ 2.004-88. Единая система конструкторской документации. Общие требования к выполнению конструкторских технологических документов на печатающих и графических устройствах вывода ЭВМ.

ГОСТ 2.051-2006. Единая система конструкторской документации. Электронные документы. Общие положения.

ГОСТ 2.052-2006. Единая система конструкторской документации. Электронная модель изделия. Общие положения.

ГОСТ 2.601-2013. Единая система конструкторской документации. Эксплуатационные документы.

2 Техническое задание

2.1 Наименование проекта

Разработка интерактивного сервиса «Электронный журнал посещаемости».

2.2 Цель проекта

Целью проекта является создание интерактивного сервиса «Электронный журнал посещаемости», который в дальнейшем поможет уменьшить время, затраченное на отметку студентов во время пары на 75 %, за счет автоматизации процесса с помощью QR-кода.

2.3 Требования к функциональности

Сервис должен предоставлять следующие возможности:

- получение входных данных из корпоративной информационной систе-мы организации;
- просмотр учебного расписания преподавателем;
- просмотр, добавление и изменение информации о посещении, оценок и заметок о студентах выбранной учебной группы;
- просмотр, создание, удаление и скачивание QR-кодов для отметки по-сещения студентов;
- синхронизация информации с удаленным хранилищем;
- информирование пользователя о возникающих ошибках и сбоях.

2.4 Требования к дизайну

Дизайн сервиса должен быть современным и привлекательным для це-левой аудитории. Для этого необходимо использовать сочетание цветовых решений, текстовых блоков и графических элементов.

Также цветовая гамма должна соответствовать фирменному стилю Университета, а использование шрифтов должно соответствовать корпоративному стилю.

2.5 Технические требования

Сервис должен быть разработан с использованием современного стека технологий MERN. Должна быть поддержка с другими системами. Для удобства пользователей сервис должен поддерживать отображение на мобильных устройствах и иметь адаптивный дизайн.

2.6 Срок выполнения проекта

Проект должен быть завершен 11.05.2025.

2.7 Требования к аппаратному обеспечению

Для просмотра сервиса необходимо использовать персональный компьютер или мобильное устройство с доступом в Интернет и следующими характеристиками:

- процессор Intel Core i3 или аналогичный;
- оперативная память не менее 4 Гб;
- разрешение экрана не менее 1280 x 720 пикселей.

2.8 Требования к программному обеспечению

Для просмотра сервиса необходимо использовать веб-браузер, поддерживающий HTML3 и CSS3. Рекомендуется использовать один из следующих браузеров:

- Google Chrome версии 80 или выше;
- Mozilla Firefox версии 75 или выше;
- Apple Safari версии 13 или выше;
- Microsoft Edge версии 80 или выше.

3 Руководство разработчика

В качестве инструмента разработки использовался стек MERN, включающий в себя MySQL в качестве базы данных, Express.js в качестве сервера для приложения, React.js в качестве библиотеки для клиентской части приложения, Node.js в качестве базы самого приложения.

Основные схемы хранящихся данных показана в листингах 3.1 – 3.10

Листинг 3.1 – Схема, описывающая пользователя

```
const { Sequelize, DataTypes } = require('sequelize');

module.exports = (sequelize) => {
  const Roles = require('./role-model')(sequelize);

    return sequelize.define('users', {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
          allowNull: false
        },
        login: {
          type: DataTypes.STRING,
          allowNull: false,
          unique: true
        },
        password: {
            type: DataTypes.STRING,
            allowNull: false,
        },
        id_role: {
          type: DataTypes.INTEGER,
          allowNull: false,
          references: {
              model: Roles,
              key: 'id',
          },
          defaultValue: 1
      },
      },
    );
}
```

Листинг 3.2 – Схема, описывающая типы QR-кода

```
const { Sequelize, DataTypes } = require('sequelize');

module.exports = (sequelize) => {
    return sequelize.define('type_of_qr_code', {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
          allowNull: false
        },
        value: {
          type: DataTypes.STRING,
          allowNull: false,
          unique: true
        }
      }
    );
}
```

Листинг 3.3 – Схема, описывающая преподавателя

```
const { Sequelize, DataTypes } = require('sequelize');

module.exports = (sequelize) => {
  const Persons = require('./person-model')(sequelize);

    return sequelize.define('teachers', {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
          allowNull: false
        },
        id_person: {
          type: DataTypes.INTEGER,
          allowNull: true,
          references: {
              model: Persons,
              key: 'id',
          }
      },
      },
    );
}
```

Листинг 3.4 – Схема, описывающая дисциплины

```
const { Sequelize, DataTypes } = require('sequelize');
module.exports = (sequelize) => {
    return sequelize.define('subjects', {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
          allowNull: false
        },
        name: {
          type: DataTypes.STRING,
          allowNull: false,
          unique: true
        }
      }
    );
}
```

Листинг 3.5 – Схема, описывающая студента

```
const { Sequelize, DataTypes } = require('sequelize');
module.exports = (sequelize) => {
  const Persons = require('./person-model')(sequelize);
  const Groups = require('./group-model')(sequelize);
    return sequelize.define('students', {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
          allowNull: false
        },
        id_person: {
          type: DataTypes.INTEGER,
          allowNull: true,
          references: {
              model: Persons,
              key: 'id',
          }
      },
      id_group: {
        type: DataTypes.INTEGER,
        allowNull: true,
        references: {
            model: Groups,
            key: 'id',
        }
    },
      },
    );
}
```

Листинг 3.6 – Схема, описывающая расписание

```
const { Sequelize, DataTypes } = require('sequelize');

module.exports = (sequelize) => {
  const Teachers = require('./teacher-model')(sequelize);
  const Groups = require('./group-model')(sequelize);
  const Subjects = require('./subject-model')(sequelize);
  const     NumberOfCouple     =     require('./number_of_couple-
model')(sequelize);

    return sequelize.define('schedules', {
        id: {
            type: DataTypes.INTEGER,
            autoIncrement: true,
            primaryKey: true,
            allowNull: false
        },
        date: {
            type: DataTypes.DATEONLY,
            allowNull: false
        },
        room: {
            type: DataTypes.STRING,
            allowNull: false
        },
        type: {
            type: DataTypes.STRING,
            allowNull: false
        },
        id_number_of_couple:  {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: NumberOfCouple,
                key: 'id',
            }
        },
        id_teacher: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Teachers,
                key: 'id',
            }
        },
        id_group: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Groups,
                key: 'id',
```

```
            }
        },
        id_subject: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Subjects,
                key: 'id',
            }
        },
      },
    );
}
```

Листинг 3.7 – Схема, описывающая роли

```
const { Sequelize, DataTypes } = require('sequelize');

module.exports = (sequelize) => {
    return sequelize.define('roles', {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
          allowNull: false
        },
        value: {
          type: DataTypes.STRING,
          allowNull: false,
          unique: true
        }
      }
    );
}
```

Листинг 3.8 – Схема, описывающая QR-коды

```
const { Sequelize, DataTypes } = require('sequelize');
module.exports = (sequelize) => {
  const Teachers = require('./teacher-model')(sequelize);
  const Subjects = require('./subject-model')(sequelize);
  const Schedules = require('./schedule-model')(sequelize);
  const      TypeOfQRCode      =      require('./type_of_qr_code-
model')(sequelize);

    return sequelize.define('qr_code', {
        id: {
            type: DataTypes.INTEGER,
            autoIncrement: true,
            primaryKey: true,
            allowNull: false
```

```
        },
        start: {
            type: DataTypes.DATEONLY,
            allowNull: true
        },
        end: {
            type: DataTypes.DATEONLY,
            allowNull: true
        },
        room: {
            type: DataTypes.STRING,
            allowNull: true
        },
        id_type_of_qr_code: {
            type: DataTypes.INTEGER,
            allowNull: false,
            references: {
                model: TypeOfQRCode,
                key: 'id',
            }
        },
        id_teacher: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Teachers,
                key: 'id',
            }
        },
        id_subject: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Subjects,
                key: 'id',
            }
        },
        id_schedule: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Schedules,
                key: 'id',
            }
        },
        hashcode: {
            type: DataTypes.STRING,
            allowNull: false,
        },
      },
    );
}
```

Листинг 3.9 – Схема, описывающая журнал

```
const { Sequelize, DataTypes } = require('sequelize');

module.exports = (sequelize) => {
  const Schedules = require('./schedule-model')(sequelize);
  const Students = require('./student-model')(sequelize);
  const Groups = require('./group-model')(sequelize);

    return sequelize.define('journal', {
        id: {
            type: DataTypes.INTEGER,
            autoIncrement: true,
            primaryKey: true,
            allowNull: false
        },
        id_schedule: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Schedules,
                key: 'id',
            }
        },
        id_student: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Students,
                key: 'id',
            }
        },
        id_group: {
            type: DataTypes.INTEGER,
            allowNull: true,
            references: {
                model: Groups,
                key: 'id',
            }
        },
        id_attendance: {
            type: DataTypes.INTEGER,
            allowNull: false,
            defaultValue: 1
        },
        grade: {
            type: DataTypes.INTEGER,
            allowNull: false,
            defaultValue: 1
        },
        note: {
            type: DataTypes.STRING(2048),
```

```
            allowNull: false,
            defaultValue: ""
          },
      },
      {
        indexes: [
            {
                unique: true,
                fields: ['id_schedule', 'id_student']
            }
        ]
      }
    );
}
```

Листинг 3.10 – Схема, описывающая группы

```
const { Sequelize, DataTypes } = require('sequelize');

module.exports = (sequelize) => {
    return sequelize.define('groups', {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
          allowNull: false
        },
        name: {
          type: DataTypes.STRING,
          allowNull: false,
        }
      }
    );
}
```

Для каждой схемы данных созданы методы поддерживающие CRUD (Создание, чтение, обновление и удаление), данные операции показаны для схемы журнала, показаны в листинге 3.11.

Листинг 3.11 – CRUD методы для схемы журнала

```
const db = require('../models');
const { QueryTypes } = require('sequelize');

class JournalController {
    async getJournal(req, res) {
        try {
            const sel_id_schedule = req.body.id_schedule;
```

```
            const journals = await db.sequelize.query("SELECT
P.lastname, P.name, P.patronymic, ST.id AS id_student, J.id, CO-
ALESCE(J.id_schedule,    :sel_id_schedule)    AS    id_schedule,
J.id_attendance, J.grade, J.note " +
            "FROM university.persons AS P INNER JOIN universi-
ty.students AS ST " +
            "ON P.id = ST.id_person " +
            "LEFT JOIN university.schedules AS SCH " +
            "ON ST.id_group = SCH.id_group " +
            "LEFT JOIN university.journals AS J " +
            "ON J.id_schedule = SCH.id AND ST.id = J.id_student
" +
            "WHERE SCH.id = :sel_id_schedule " +
            "ORDER BY P.lastname, P.name, P.patronymic", {
                replacements:    {    sel_id_schedule:    Num-
ber(sel_id_schedule)},
                type: QueryTypes.SELECT
              });

            res.json(journals);
        } catch (e) {
            console.log(e);
            res.status(400).json({message: 'Getting journals er-
ror'});
        }
    }

    async getStatistic(req, res) {
        try {
            const id_student = req.body.id_student;
            const start = req.body.start ?? null;
            const end = req.body.end ?? null;

            const statistic = await db.sequelize.query("SELECT
SCH.id, ST.id, J.id_attendance, J.grade, SCH.date " +
            "FROM university.schedules AS SCH " +
            "INNER JOIN university.students AS ST ON ST.id_group
= SCH.id_group " +
            "LEFT JOIN university.journals AS J ON J.id_student
= ST.id AND J.id_schedule = SCH.id " +
            "WHERE (SCH.date >= :start OR :start IS NULL) AND
(SCH.date <= :end OR :end IS NULL) AND ST.id = :id_student", {
                replacements: {
                    id_student: Number(id_student),
                    start,
                    end
                },
                type: QueryTypes.SELECT
              });

            res.json(statistic);
```

```
        } catch (e) {
            console.log(e);
            res.status(400).json({message:   'Getting   statistic
error'});
        }
    }

    async setJournal(req, res) {
        try {
            const sel_journal = req.body.journal;

            if(!sel_journal.id) {
                await  db.sequelize.query("INSERT  INTO  universi-
ty.journals " +
                "(id_schedule, id_student, id_attendance, grade,
note) " +
                "VALUES " +
                "(:sel_id_schedule, " +
                ":sel_id_student, " +
                ":sel_id_attendance, " +
                ":sel_grade, " +
                ":sel_note);", {
                replacements: {
                    sel_id_schedule: sel_journal.id_schedule,
                    sel_id_student: sel_journal.id_student,
                    sel_id_attendance:  sel_journal.id_attendance
?? 1,
                    sel_grade: sel_journal.grade ?? 1,
                    sel_note: sel_journal.note ?? ''
                },
                type: QueryTypes.INSERT
              });
            } else {
                await    db.sequelize.query("UPDATE    universi-
ty.journals " +
                    " SET " +
                    " id_schedule = :sel_id_schedule, " +
                    " id_student = :sel_id_student, " +
                    " id_attendance = :sel_id_attendance, " +
                    " grade = :sel_grade, " +
                    " note = :sel_note " +
                    " WHERE id = :sel_id;", {
                replacements: {
                    sel_id: sel_journal.id,
                    sel_id_schedule: sel_journal.id_schedule,
                    sel_id_student: sel_journal.id_student,
                    sel_id_attendance:  sel_journal.id_attendance
?? 1,
                    sel_grade: sel_journal.grade ?? 1,
                    sel_note: sel_journal.note ?? ''
                },
                type: QueryTypes.UPDATE
```

```
            });
        }

        const new_journal = await db.sequelize.query("SELECT
P.lastname, P.name, P.patronymic, ST.id AS id_student, J.id,
J.id_schedule, J.id_attendance, J.grade, J.note " +
        "FROM university.persons AS P INNER JOIN universi-
ty.students AS ST " +
        "ON P.id = ST.id_person " +
        "INNER JOIN university.schedules AS SCH " +
        "ON ST.id_group = SCH.id_group " +
        "INNER JOIN university.journals AS J " +
        "ON J.id_schedule = SCH.id AND ST.id = J.id_student
" +
        "WHERE J.id_schedule = :sel_id_schedule AND
J.id_student = :sel_id_student" , {
            replacements: {sel_id_schedule:
sel_journal.id_schedule, sel_id_student:
sel_journal.id_student},
            type: QueryTypes.SELECT
          });

        res.status(200).json(new_journal[0]);
    } catch (e) {
        console.log(e);
        res.status(400).json({message: 'Setting journal er-
ror'});
    }
  }
}

module.exports = new JournalController();
```

Одним из важных элементов сервиса является регистрации и авторизация, код реализующий эти возможности показан в листинге 3.12.

Листинг 3.12 – Авторизация и регистрация

```
const db = require('../models');
const { QueryTypes } = require('sequelize');
const bcrypt = require('bcryptjs');
const { validationResult } = require('express-validator');
const tokenService = require('../service/token-service.js');

class UserController {
  async registration(req, res) {
    try {
      const errors = validationResult(req);
```

```
            if(!errors.isEmpty()){
                return      res.status(400).json('Ошибка      при
регистрации', errors);
            }

            const {login, password} = req.body;
            const  candidate  =  await  db.users.findOne({  where:
{login: login}})

            if(candidate) {
                return res.status(400).json({message: 'Пользова-
тель с таким именем уже существует'});
            }
            const hashPassword = bcrypt.hashSync(password, 7);
            const  user  =  await  db.users.create({login:  login,
password: hashPassword})

            return res.json({message: 'Пользователь успешно за-
регистрирован'});
        } catch (e) {
            console.log(e);
            res.status(400).json({message:   'Registration   er-
ror'});
        }
    }

    async login(req, res) {
        try {
            const {login, password} = req.body;

            const user = await db.users.findOne({where: {login:
login}});
            if(!user){
                return             res.status(400).json({message:
'Пользователь с таким именем не найден'});
            }

            const  validPassword  =  bcrypt.compareSync(password,
user.password);
            if(!validPassword){
                return   res.status(400).json({message:   'Введен
неверный пароль'});
            }

            const          extendedUser          =          await
db.sequelize.query('SELECT  U.id,  U.login,  T.id  AS  id_teacher,
ST.id  AS  id_student,  R.value  AS  role,  P.name,  P.lastname,
P.patronymic ' +
            'FROM university.users AS U ' +
            'LEFT  JOIN  university.persons  AS  P  ON  U.id  =
P.id_user ' +
            'LEFT JOIN university.roles AS R ON U.id_role = R.id
```

```
' +
            'LEFT JOIN university.teachers AS T ON P.id =
T.id_person ' +
            'LEFT JOIN university.students AS ST ON P.id =
ST.id_person ' +
            'WHERE U.id = :sel_id_user' , {
                replacements: { sel_id_user: user.id},
                type: QueryTypes.SELECT
            });

            const token = token-
Service.generateAccessToken(extendedUser[0]);

            return res.json({token, extendedUser: ex-
tendedUser[0]});
        } catch (e) {
            console.log(e);
            res.status(400).json({message: 'Login error'});
        }
    }

    async checkAuth(req, res) {
        try {
            const token = req.headers.authorization.split('
')[1];
            if(!token) {
                return res.status(400).json({message:
'Пользователь не авторизован'});
            }

            const decodedData = token-
Service.validateAccessToken(token);
            if(decodedData === null) {
                return res.status(400).json({message:
'Пользователь не авторизован'});
            } else {
                return res.status(200).json({extendedUser: de-
codedData, message: 'Пользователь авторизован'});
            }
        } catch (e) {
            console.log(e);
            return res.status(400).json({message: 'Пользователь
не авторизован'});
        }
    }

    async setRole(req, res) {
        try {
            const {idUser, newRole} = req.body;

            const role = await db.roles.findOne({
                where: {
```

```
                    value: newRole
                }
            });

            if(!role){
                json.status(400).json(`Не найдена роль -
${newRole}`);
                return;
            }

            await db.users.update({ id_role: role.id }, {
                where: {
                  id: idUser
                }
              });

              res.json({message: 'Роль пользователя
обновлена'});
        } catch (e) {
            console.log(e);
            res.status(400).json({message: 'Setting new role for
user error'});
        }
    }

    async getUsers(req, res) {
        try {
            const users = await db.users.findAll({raw: true});

            res.json({users});
        } catch (e) {
            console.log(e);
            res.status(400).json({message: 'Getting users er-
ror'});
        }
    }
}

module.exports = new UserController();
```

Журнал посещения являются самой большой и важной частью проекта, код реализующий вывод журнала, модальных окон показан в листингах 3.13 – 3.15.

Листинг 3.13 – Код, выводящий страницу журнала

```
import React, {useEffect, useContext, useState} from 'react';
import {observer} from 'mobx-react-lite';
```

```
import {Context} from '../index';
import { useNavigate } from "react-router-dom";
import { useParams } from 'react-router-dom';
import TableControlForJournal from './TableControlForJournal';
import                  TableForStudentsList                  from
'./UI/table/TableForStudentsList';
import '../styles/BodyJournal.css'

const BodyJournal = () => {
    const {id} = useParams();
    const {store} = useContext(Context);
    const navigate = useNavigate();
    const [couple, setCouple] = useState();

    useEffect(() => {
        async function fetchData() {
            await store.getCouple(id)
            .then((data) => {
                console.log(data)
                setCouple(data);
            })
            .catch(() => {
                console.log("getCouple")
                store.setIsMessage({isActiveBad:   true,   text:
"Ошибка загрузки журнала", callback: () => navigate('/schedule',
{replace: true})});
            })

          }
          fetchData();
    }, [id])

    return (
        <div className='body_journal'>
            <TableControlForJournal couple={couple}/>
            <TableForStudentsList id={id}/>
        </div>
    );
};

export default observer(BodyJournal);
```

Листинг 3.14 – Код, выводящий компонент управления таблицей

```
import React from "react";
import { useNavigate } from "react-router-dom";
import JButton from './UI/buttons/JButton';
import '../styles/TableControl.css';

const TableControlForJournal = ({couple, ...props}) => {
    const navigate = useNavigate();
```

```
     const prevPage = () => {
        navigate('/schedule', {replace: true});
    }

    return (
        <div className='table_control'>
            <div className='table_control_journal'>
                <JButton     style={{width:     '64px'}}     on-
Click={prevPage}>
                    <svg width="20" height="20" viewBox="0 0 20
20" fill="none" xmlns="http://www.w3.org/2000/svg">
                        <path     d="M7.97501     15.6833C7.81668
15.6833 7.65835 15.625 7.53335 15.5L2.47501 10.4416C2.23335 10.2
2.23335 9.79998 2.47501 9.55831L7.53335 4.49998C7.77501 4.25831
8.17501 4.25831 8.41668 4.49998C8.65835 4.74164 8.65835 5.14164
8.41668 5.38331L3.80001 9.99998L8.41668 14.6166C8.65835 14.8583
8.65835 15.2583 8.41668 15.5C8.30001 15.625 8.13335 15.6833
7.97501 15.6833Z" fill="#292D32"/>
                        <path d="M17.0834 10.625H3.05835C2.71668
10.625 2.43335 10.3417 2.43335 10C2.43335 9.65833 2.71668 9.375
3.05835 9.375H17.0834C17.425 9.375 17.7084 9.65833 17.7084
10C17.7084    10.3417    17.425    10.625    17.0834    10.625Z"
fill="#292D32"/>
                    </svg>
                </JButton>
                <div>
                    <p   style={{fontWeight:   '400',   fontStyle:
'normal', fontSize: '24px'}}>{couple?.name_subject}</p>
                    <p   style={{fontWeight:   '400',   fontStyle:
'normal',           fontSize:           '14px',           opacity:
'0.5'}}>{couple?.name_group}</p>
                </div>
            </div>
        </div>
    );
};

export default TableControlForJournal;
```

Листинг 3.15 – Код, выводящий компонент со списком студентов

```
import React, {useState, useContext, useEffect} from 'react';
import {observer} from 'mobx-react-lite';
import { Table, ConfigProvider, Select } from 'antd';
import SelectAttendance from './select/SelectAttendance';
import SelectGrade from './select/SelectGrade';
import JModal from '../../modals/JModal';
import CreateNote from '../../CreateNote';
import {Context} from '../../../index';
import { OverlayScrollbarsComponent } from 'overlayscrollbars-
react';
```

```
import 'overlayscrollbars/overlayscrollbars.css';
import './TableForStudentsList.css';

const { Column } = Table;

const attendanceItems = [
    {
      value: 1,
      label: (
        <SelectAttendance value={1}/>
      ),
    },
    {
        value: 2,
      label: (
        <SelectAttendance value={2}/>
      ),
    },
    {
        value: 3,
      label: (
        <SelectAttendance value={3}/>
      ),
    },
    {
        value: 4,
      label: (
        <SelectAttendance value={4}/>
      ),
    },
  ];

  const gradeItems = [
    {
      value: 1,
      label: (
        <SelectGrade value={'1'}/>
      ),
    },
    {
        value: 2,
      label: (
        <SelectGrade value={'2'}/>
      ),
    },
    {
        value: 3,
      label: (
        <SelectGrade value={'3'}/>
      ),
    },
    {
```

```
        value: 4,
      label: (
        <SelectGrade value={'4'}/>
      ),
    },
    {
      value: 5,
    label: (
      <SelectGrade value={'5'}/>
    ),
  },
  ];

const TableForStudentsList = ({id, props}) => {
    const {store} = useContext(Context);
    const  [modalCreateNoteActive,  setModalCreateNoteActive]  =
useState(false);
    const [journals, setJournals] = useState();
    const [student, setStudent] = useState();

    useEffect(() => {
      async function fetchData() {
          setJournals(await store.getJournals(id));
        }
        fetchData();
      }, [student])

    const saveStudent = async (change_student) => {
      store.setJournal(change_student)
      .then((res) => {
          setStudent(res.data);
      })
      .catch(() => {
        store.setIsMessage({isActiveBad:  true,  text:  "Ошибка
сохранения"});
      });
  }

    return (
        <div className='table_for_students_list_for_scrollbar'>
            <OverlayScrollbarsComponent
            options={{ scrollbars: { theme: 'os-theme-dark'} }}
            style={{ height: '100%'}}
            defer >
              <ConfigProvider
                theme={{
                  components: {
                    Table: {
                      borderColor: '#0FA958',
                    },
                  },
                }} >
```

```
                <Table bordered dataSource={journals}
style={{width: '100%', height: 'auto'}} pagination={false}
sticky={{ offsetHeader: 0}} scroll={{x: '100%'}} row-
Key={(record) => record.id_student}>
                  <Column align='center' title="№" dataIn-
dex="number" key="number" width={64} render={
                    (_, student, index) => {
                      return <p>{index + 1}</p>;
                  }
                  }/>
                  <Column title="ФИО" dataIndex="fio" key="fio"
width={400} render={
                    (_, student) => {
                        return (
                            <p style={{fontWeight: "550", font-
Size: "14px"}}>{`${student.lastname} ${student.name} ${stu-
dent.patronymic}`}</p>
                        );
                    }
                  }
                  />
                  <Column title="Присутствие" dataIndex="status"
key="status" width={192} render={
                    (_, student) => {
                        return (
                            <Select onSelect={async (value) => {
                                const change_student =
JSON.parse(JSON.stringify(student));
                                change_student.id_attendance =
value;
                                await saveStu-
dent(change_student);
                            }} default-
Value={student.id_attendance ?? 1} suffixIcon={null} style={{
width: '100%', height: '100%' }} variant="borderless" op-
tions={attendanceItems} />
                        );
                    }
                  }
                  />
                  <Column title="Оценка" dataIndex="grade"
key="grade" width={144} render={
                    (_, student) => {
                        return (
                            <Select onSelect={async (value) => {
                                const change_student =
JSON.parse(JSON.stringify(student));
                                change_student.grade = value;
                                await saveStu-
dent(change_student);
                            }} defaultValue={student.grade ??
1} suffixIcon={null} listHeight={500} style={{ width: '100%',
```

```
height: '100%' }} variant="borderless" options={gradeItems} />
                          );
                        }
                      }
                      />
                      <Column    title="Заметка"    dataIndex="note"
key="note" minWidth={128} ellipsis={true} onCell={
                        (student, rowIndex) => {
                          return {
                            onClick: (ev) => {
                              setModalCreateNoteActive(true);
                              setStudent(student);
                            },
                          };
                        }
                      }/>
                    </Table>
                  </ConfigProvider>
                </OverlayScrollbarsComponent>
                {modalCreateNoteActive      ?      <JModal      isAc-
tive={modalCreateNoteActive}                                  setIsAc-
tive={setModalCreateNoteActive}><CreateNote     student={student}
setStudent={setStudent}                                        setAc-
tive={setModalCreateNoteActive}/></JModal> : null}
        </div>
    );
};

export default observer(TableForStudentsList);
```

4 Руководство пользователя

Людей, взаимодействующих с сервисом можно разделить на 3 роли:

- сотрдуник деканата;
- преподаватель;
- студент.

При открытии сервиса ЭП появляется страница авторизации, на которой пользователь должен ввести свои учетные данные от сервиса в поля для ввода логина и пароля для продолжения использования сервиса (рисунок 4.1).

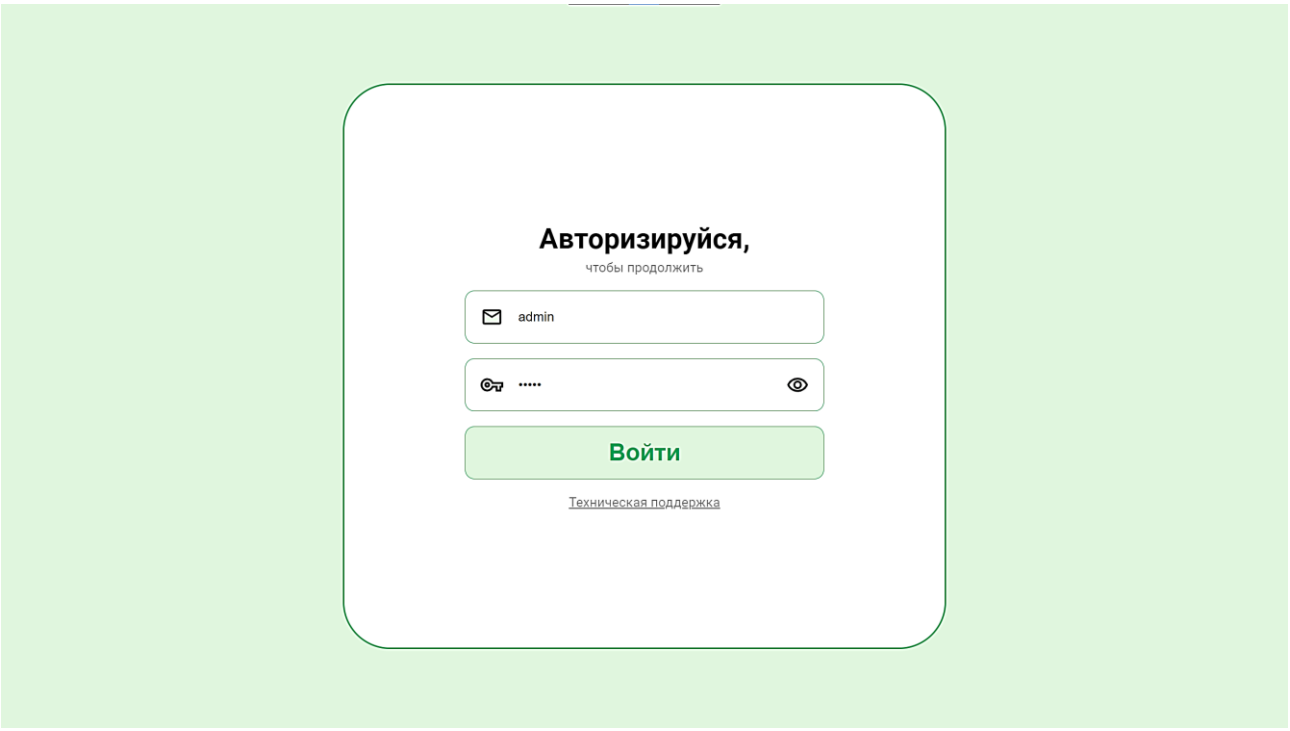

Рисунок 4.1 – Авторизация пользователя

При осуществлении перехода на главную страницу, путем нажатия кнопки «Войти» и успешной проверки введенных данных на сервере, появиться страница, на которой будет отображены учебное расписание, а также сопутствующие элементы интерфейса: ссылка для перехода на страницу с учебным журналом выбранной группы для осуществления процесса введения

учебного журнала, кнопка «Просмотр созданных QR-кодов» для просмотра, скачивания и удаления ранее созданных QR-кодов и кнопка «Создание QR-кода» для создания QR-кода (рисунок 4.2).

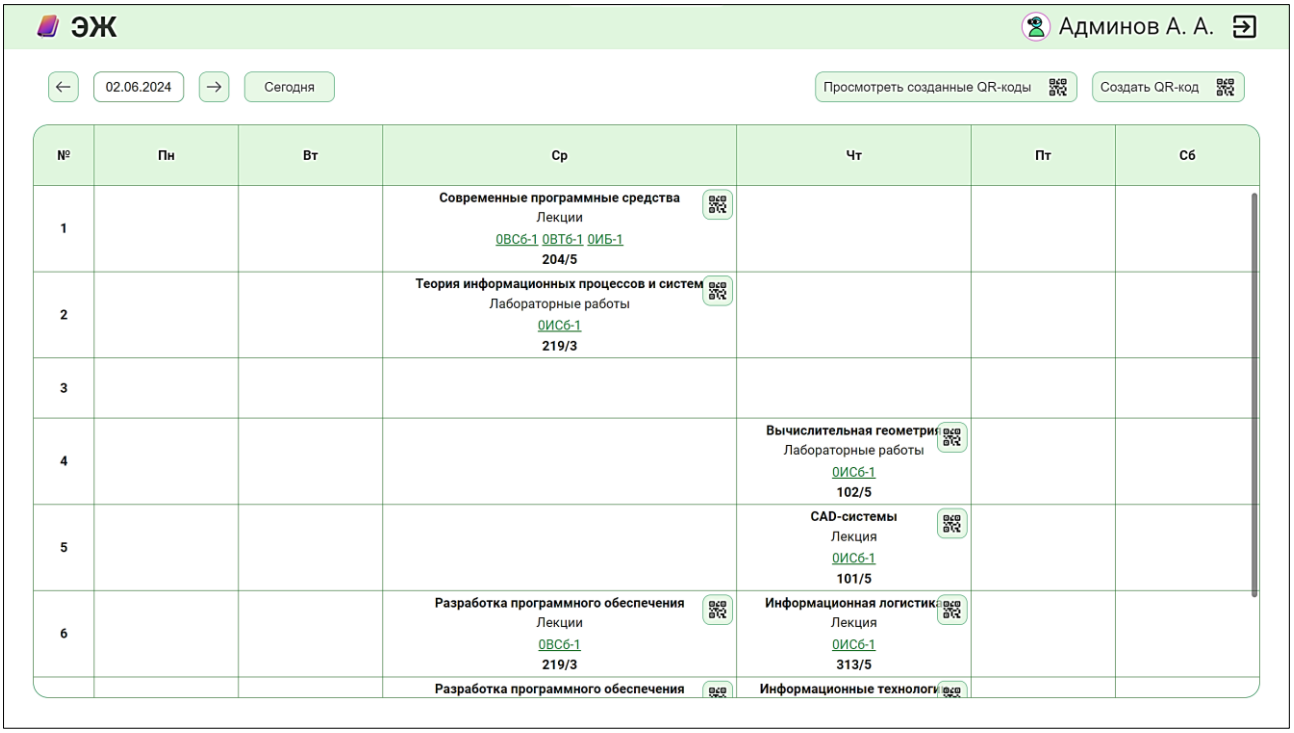

Рисунок 4.2 – Выбор необходимых действий

Для создания QR-кода необходимо нажать на кнопку «Создать QR-код». После чего откроется модальное окно «Создание QR-кода», в котором можно будет выбрать с помощью вкладок «Создать QR-код на аудиторию», «Создать QR-код на предмет», «Создать QR-код на преподавателя» тип QR-кода, а также установить срок действия кода с помощью полей «С» и «До» или сделать срок действия QR-кода бессрочным с помощью переключателя «Бессрочный» (рисунки 4.3, 4.4, 4.5).

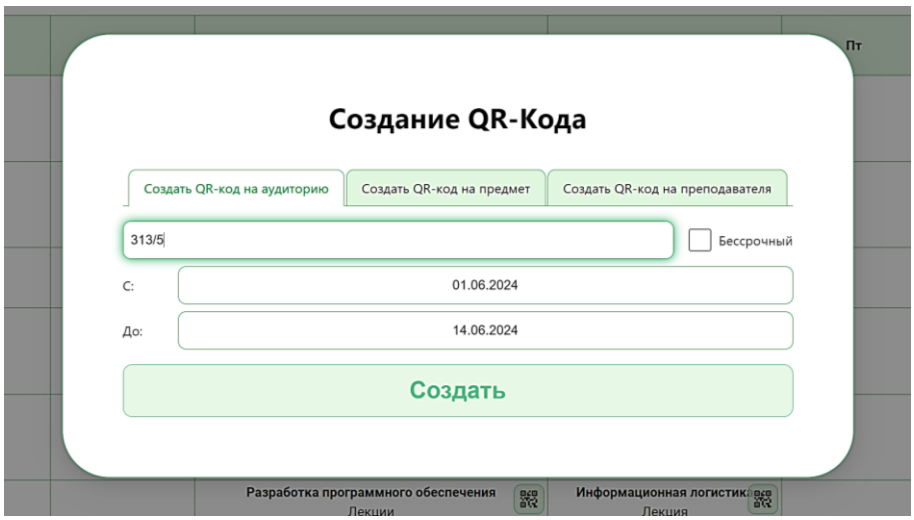

Рисунок 4.3 – Создание QR-кода с типом «На аудиторию»

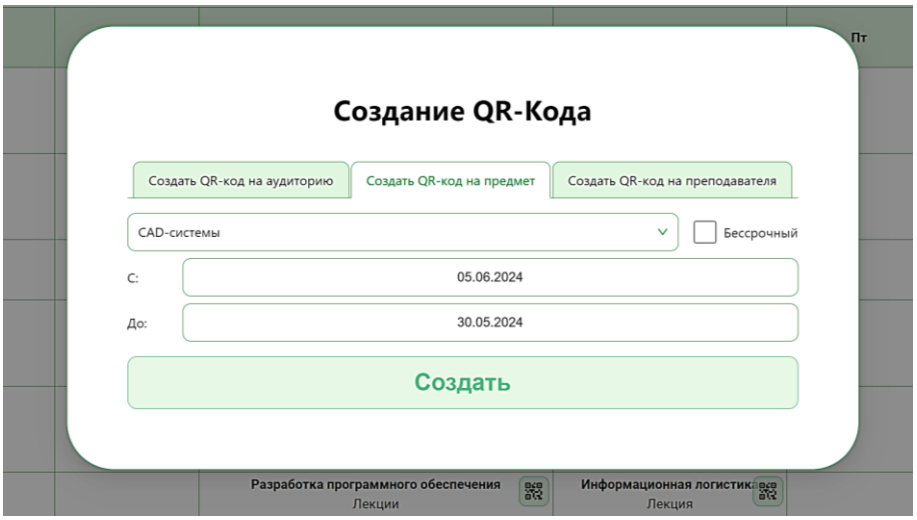

Рисунок 4.4 – Создание QR-кода с типом «На аудиторию»

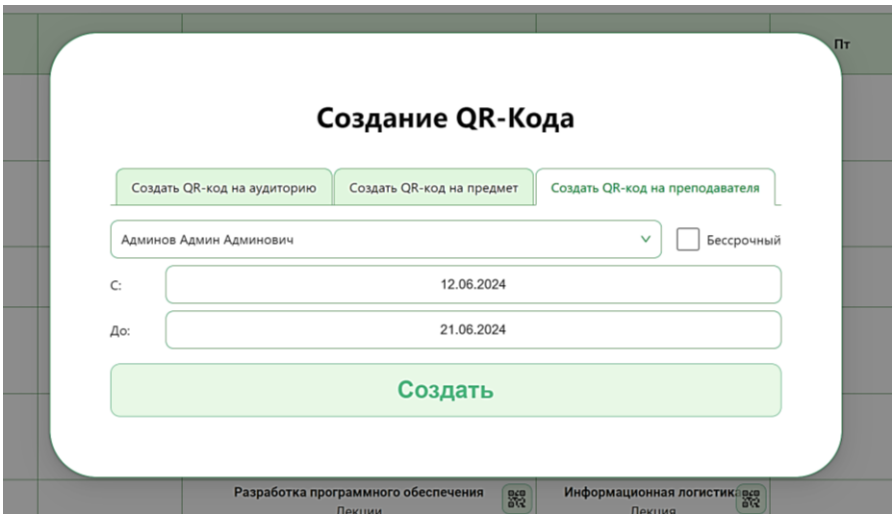

Рисунок 4.5 – Создание QR-кода с типом «На аудиторию»

Для просмотра ранее созданных QR-кодов необходимо нажать на кнопку «Просмотреть созданные QR-коды». После чего откроется модальное окно «QR-коды», в котором можно выбрать из списка необходимый QR-код с помощью нажатия зеленной кнопки со знаком QR-кода для открытия модального окна «QR-код», в котором доступен его просмотр, скачивание с помощью нажатия кнопки «Скачать» и удаление с помощью нажатия кнопки «Удалить» (рисунки 4.6, 4.7).

Рисунок 4.6 – Просмотр списка созданных QR-кодов

Рисунок 4.7 – Просмотр и ведение QR-кода

При осуществлении перехода на страницу с учебным журналом, путем нажатия на ссылку с необходимой группой, появиться страница, на которой будет отображен список со студентами группы, а также сопутствующие элементы интерфейса: выпадающий список для выбора статуса посещения, выпадающий список для выбора оценки и кнопка для создания заметки (рисунок 4.8).

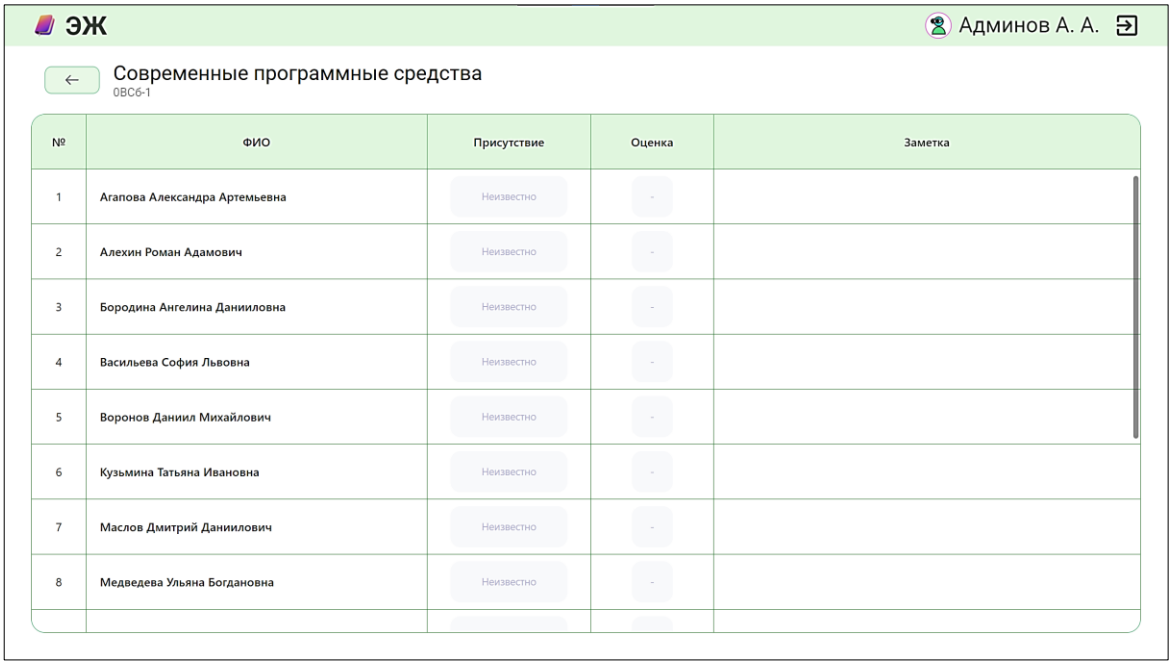

Рисунок 4.8 – Просмотр учебного журнала

Для выбора статуса посещения необходимо нажать на выпадающий список в колонке «Присутствие» и выбрать необходимый статус (рисунок 4.9).

Рисунок 4.9 – Выбор статуса посещения

Для выбора оценки необходимо нажать на выпадающий список в колонке «Оценка» и выбрать необходимую оценку (рисунок 4.10).

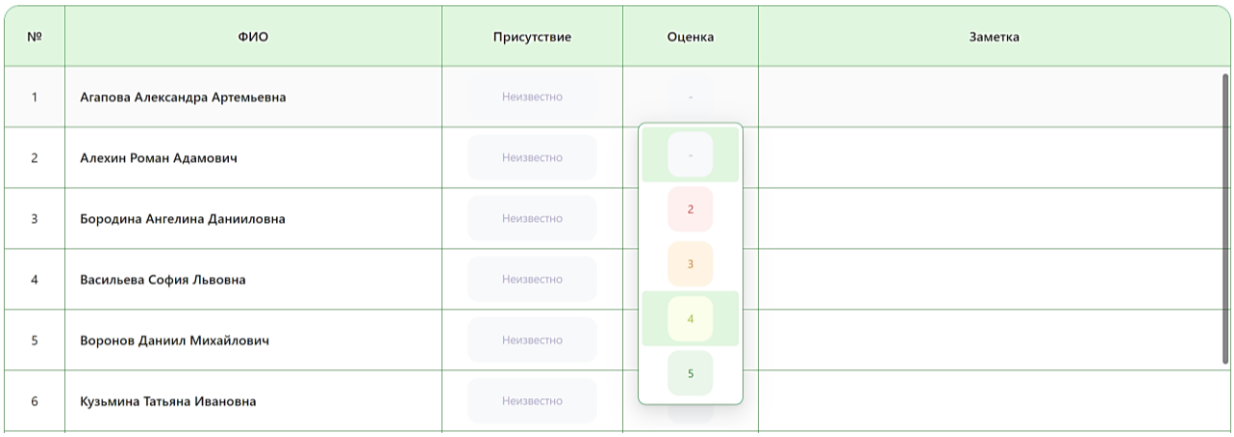

Рисунок 4.10 – Выбор оценки

Для создания заметки необходимо нажать на текстовое поле в колонке «Заметка». После чего откроется модальное окно «Заметка», в котором можно ввести заметку в поле ввода и сохранить ее с помощью кнопки «Отправить» (рисунок 4.11).

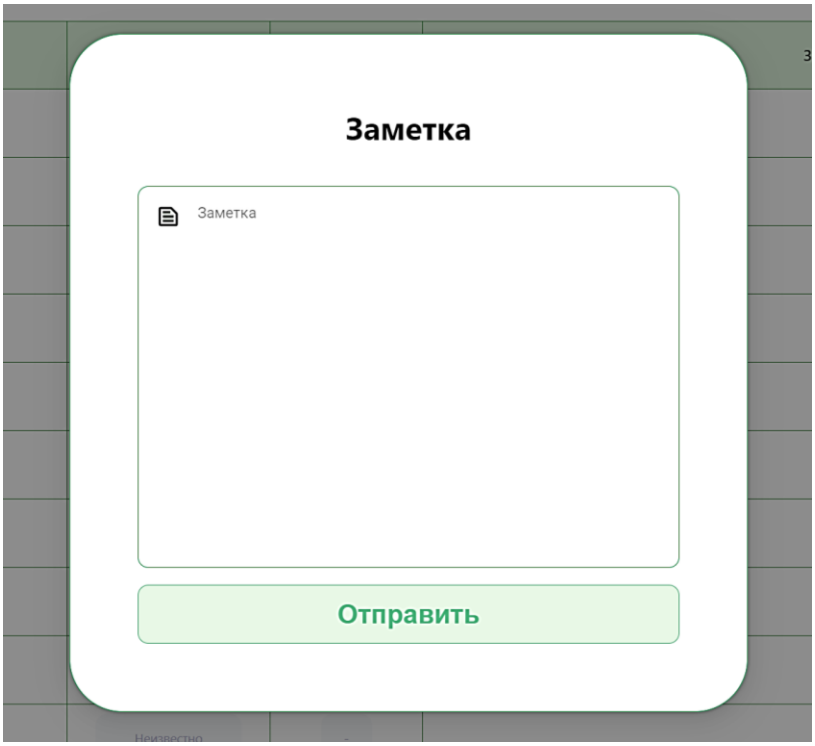

Рисунок 4.11 – Создание заметки

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

СОГЛАСОВАНО

Начальник отдела ОНиПКРС

_____ Е.М. Димитриади
(подпись)

«____ » _____ 20___ г.

Декан ФКТ

_____ *И.А. Трещев*
(подпись)

УТВЕРЖДАЮ

И.о. проректора
по научной работе

_____ А.В. Космынин
(подпись)

«____ » _____ 20___ г.

АКТ
о приеме проекта
Разработка интерактивного сервиса «Электронный журнал посещаемости»

г. Комсомольск-на-Амуре « » _____ 20__ г.

Комиссия в составе представителей:

со стороны заказчика

- Е.Б. Абарникова – руководителя СПБ «DeCode»,
- И.А. Трещев – декана _ФКТ__

со стороны исполнителя

- Е.Б. Абарникова – руководителя проекта,
- А.В. Зайцев – 4ВТм-1,

составила акт о нижеследующем:

«Исполнитель» передает проект *Разработка интерактивного сервиса «Электронный журнал посещаемости»*, в составе:

1 Руководство пользователя

2 Руководство программиста

Руководитель проекта _____ Е.Б. Абарникова
(подпись, дата)

Исполнитель проекта _____ А.В. Зайцев
(подпись, дата)