



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К**  
**КУРСОВОЙ РАБОТЕ ПО КУРСУ БАЗЫ ДАННЫХ:**

Клиент-серверное приложение «Электронный  
дневник студента»

Студент

\_\_\_\_\_  
*подпись, дата*

\_\_\_\_\_  
*фамилия, и.о.*

Научный руководитель

\_\_\_\_\_  
*подпись, дата*

\_\_\_\_\_  
*фамилия, и.о.*

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
1. Обзор предметной области.....	4
1.1 Клиент-серверные приложения .....	4
2. Разработка приложения .....	6
2.1 Проектирование базы данных .....	6
2.2 Архитектура разрабатываемого приложения .....	11
2.3 Формирование прав доступа .....	12
2.4 Алгоритм формирования очередей .....	13
3. Программная реализация приложения.....	15
3.1. Особенности реализации базы данных .....	15
3.2. Особенности реализации модуля просмотра академической успеваемости и посещаемости.....	16
3.3. Особенности реализации модуля организации очередей .....	20
4. Тестирование приложения .....	24
ЗАКЛЮЧЕНИЕ .....	38
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ .....	39
Приложение 1. Функция-обработчик входных сообщений боту .....	40

## **ВВЕДЕНИЕ**

В современном мире с развитием информационных технологий образовательные учреждения сталкиваются с необходимостью эффективной организации учебного процесса и обработки огромного объема данных. Студенты стремятся найти действенные способы для организации учебной деятельности и отслеживания академической успеваемости. В этом контексте создание приложения "Электронный дневник студента" становится актуальной задачей, которая позволит студентам ускорить процессы ведения учета успеваемости, а также обеспечить удобный доступ к необходимой информации.

Целью данной курсовой работы является создание интуитивно понятного и простого в использовании инструмента, который позволит студентам управлять своим учебным процессом, получать доступ к оценкам и другой важной информации по дисциплинам, а также организовывать очередь среди студентов на сдачу лабораторных работ. Реализация перечисленных операций будет основана на взаимодействии с базой данных, что обеспечит эффективное управление информацией об учебной деятельности студентов.

## **1. Обзор предметной области**

Электронные дневники и журналы становятся все более популярными в наше время, упрощая процессы отслеживания и анализа академических данных и способствуя повышению успеваемости студентов. Такие системы также помогают учебным заведениям повысить эффективность учебного процесса, обеспечив прозрачность и доступность информации, а также улучшив взаимодействие между студентами и преподавателями.

### **1.1 Клиент-серверные приложения**

Клиент-серверные приложения являются одним из наиболее распространенных и востребованных типов программного обеспечения в современном мире. Они представляют собой архитектурную модель, где клиентские устройства, такие как компьютеры или мобильные устройства, взаимодействуют с серверами, которые предоставляют данные, ресурсы и услуги.

Благодаря своей гибкости и масштабируемости, применение клиент-серверных приложений охватывает широкий спектр областей. В сфере интернета и веб-разработки, клиент-серверные приложения играют ключевую роль. Они обеспечивают взаимодействие между веб-браузерами пользователей и веб-серверами, где хранится и обрабатывается веб-содержимое. Это позволяет создавать динамические и интерактивные веб-сайты, электронные магазины, социальные сети и другие онлайн-приложения.

Взаимодействие между клиентом, сервером и базой данных является неотъемлемой частью многих современных приложений. Обмен информацией между клиентом, сервером и базой данных происходит следующим образом: клиент отправляет запрос на сервер, сервер обрабатывает этот запрос, взаимодействуя с базой данных, и затем отправляет ответ обратно клиенту. Этот процесс может повторяться несколько раз, пока клиент не получит нужные данные. Схема процесса представлена на рисунке 1.



Рисунок 1. Схема взаимодействия частей клиент-серверного приложения

## **2. Разработка приложения**

### **2.1 Проектирование базы данных**

Электронные дневники и журналы становятся все более популярными в наше время, упрощая процессы отслеживания и анализа академических данных и способствуя повышению успеваемости студентов. Такие системы также помогают учебным заведениям повысить эффективность учебного процесса, обеспечив прозрачность и доступность информации, а также улучшив взаимодействие между студентами и преподавателями. Однако помимо доступа к информации о прогрессе в учебе, в разрабатываемое приложение было решено добавить возможности для формирования студентами организованной очереди в случае сдачи лабораторных работ. Данная функция позволила бы упорядочить процесс сдачи работ, предотвратить хаос и долгое ожидание, а также помогла бы студентам быть более ответственными.

В таком случае для эффективной работы такого приложения необходима хорошо спроектированная база данных, которая дает возможности для хранения и управления данными о студентах. Проектируемая база данных для описанного выше электронного дневника должна соответствовать следующим требованиям:

- эффективное хранение и поиск информации о пользователях, о дисциплинах и курсах, об образовательных занятиях и полученных студентом баллах;
- эффективное хранение и поиск информации о созданных очередях и о студентах в каждой очереди;
- возможность просмотра информации об академической успеваемости и посещаемости по дисциплинам через пользовательский интерфейс.

Данная база данных состоит из нескольких основных объектов, которые взаимодействуют между собой и обеспечивают связность данных. Модель «сущность–связь» базы данных представлена на рисунке 2.

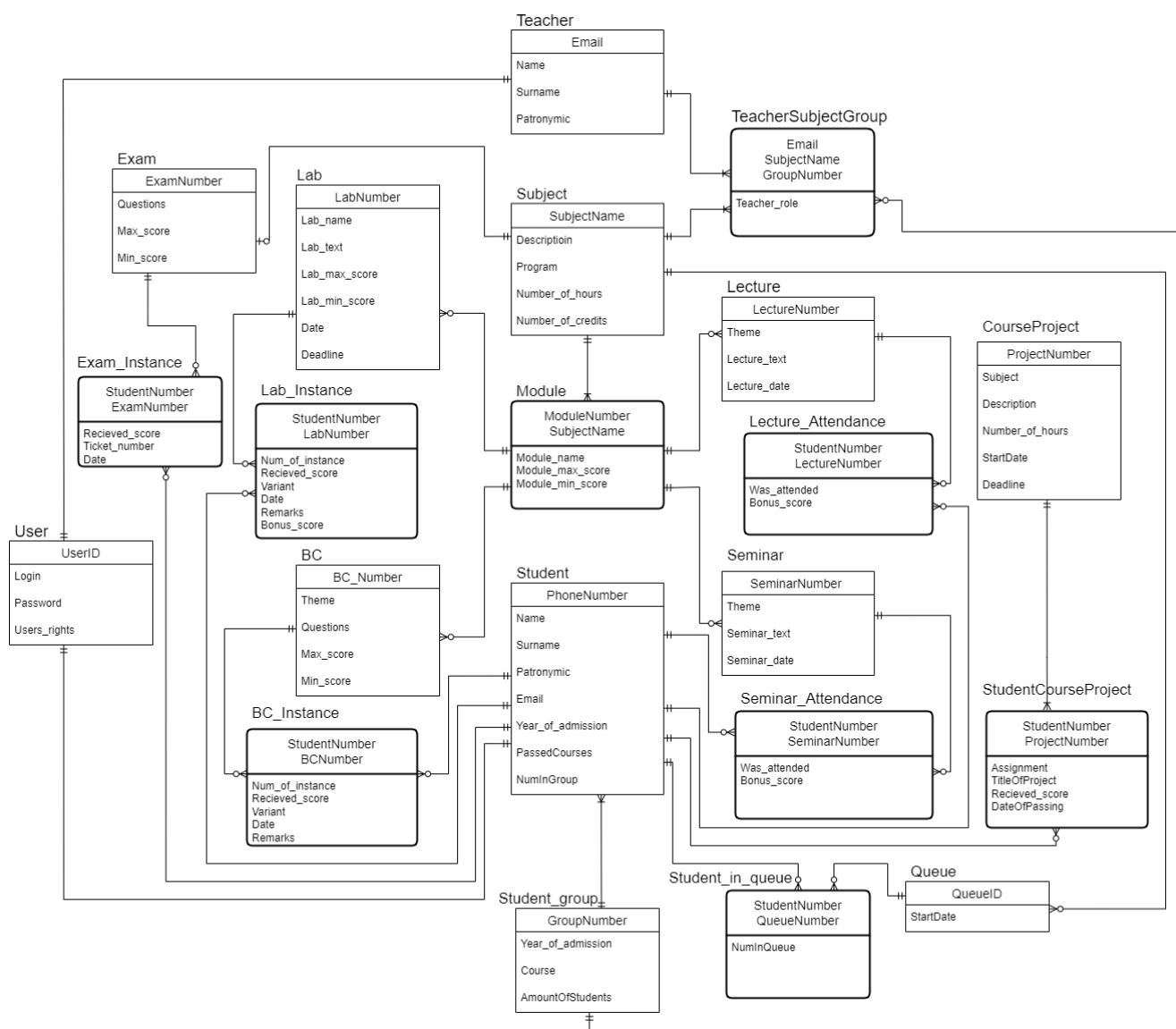


Рисунок 2. Модель «сущность–связь» базы данных

Данная модель состоит из следующих сущностей:

- User – сущность, хранящая данные о пользователе (имя пользователя, хеш пароля, права пользователя);
- Student – сущность, хранящая данные о студенте (его номер телефона, адрес электронной почты, имя, фамилию, отчество, год поступления, номер курса и порядковый номер в списке группы);
- Teacher – сущность, хранящая данные о преподавателе (его адрес электронной почты, имя, фамилию, отчество);

- Student\_group – сущность, хранящая данные об учебной группе студентов (год поступления, номер курса и число студентов в списке группы);
- Subject – сущность, хранящая данные о дисциплине (название курса, его описание, количество часов и количество зачетных единиц);
- TeacherSubjectGroup – сущность, хранящая данные о том, какой преподаватель ведет некоторую дисциплину у некоторой учебной группы студентов (что именно ведет преподаватель: лекции, семинары, лабораторные работы);
- Exam – сущность, хранящая данные об экзамене, как событии (список вопросов, возможные минимальный и максимальный баллы за экзамен);
- Exam\_Instance – сущность, хранящая данные об экзамене некоторого студента (дату сдачи, полученный за экзамен балл, номер билета);
- CourseProject – сущность, хранящая данные о курсовом проекте (описание работы, количество часов на работу, даты выдачи и сдачи работы, название дисциплины, по которой выдана работа);
- StudentCourseProject – сущность, хранящая данные о курсовом проекте некоторого студента (техническое задание, название проекта, дату сдачи, полученный балл);
- Module – сущность, хранящая данные о модуле некоторой дисциплины (номер модуля, название, возможные минимальный и максимальный балл за модуль);
- Lab – сущность, хранящая данные о лабораторной работе некоторого модуля дисциплины (название работы, задание, возможные минимальный и максимальный баллы за работу, дата выдачи, дедлайн работы);
- Lab\_Instance – сущность, хранящая данные о лабораторной работе некоторого студента (номер попытки сдачи, полученный балл, дату сдачи, номер варианта при наличии, бонусные баллы при наличии, комментарий преподавателя);
- VC – сущность, хранящая данные о рубежном контроле некоторого модуля дисциплины (номер рубежного контроля, тему, список вопросов, возможные минимальный и максимальный баллы);



- BC\_Instance – сущность, хранящая данные о рубежном контроле некоторого студента (номер попытки сдачи, полученный балл, дату сдачи, номер варианта при наличии, комментарий преподавателя);
- Lecture – сущность, хранящая данные о лекции некоторого модуля дисциплины (номер лекции, тему лекции, текст лекции и дату лекционного занятия);
- Lecture\_Attendance – сущность, хранящая данные о присутствии и активности студента на лекции (присутствие студента на занятии, бонусные баллы при наличии);
- Seminar – сущность, хранящая данные о семинаре некоторого модуля дисциплины (номер семинара, тему семинара, текст семинара и дату семинарского занятия);
- Seminar\_Attendance – сущность, хранящая данные о присутствии и активности студента на лекции (присутствие студента на занятии, бонусные баллы при наличии);
- Queue – сущность, хранящая данные об организованной очереди на определенную дату;
- Student\_in\_Queue – сущность, хранящая данные о студенте, записанном в некоторую очередь (номер студента в определенной очереди).

Полученная модель «сущность–связь» была преобразована к реляционной модели, которая представлена на рисунке 3.

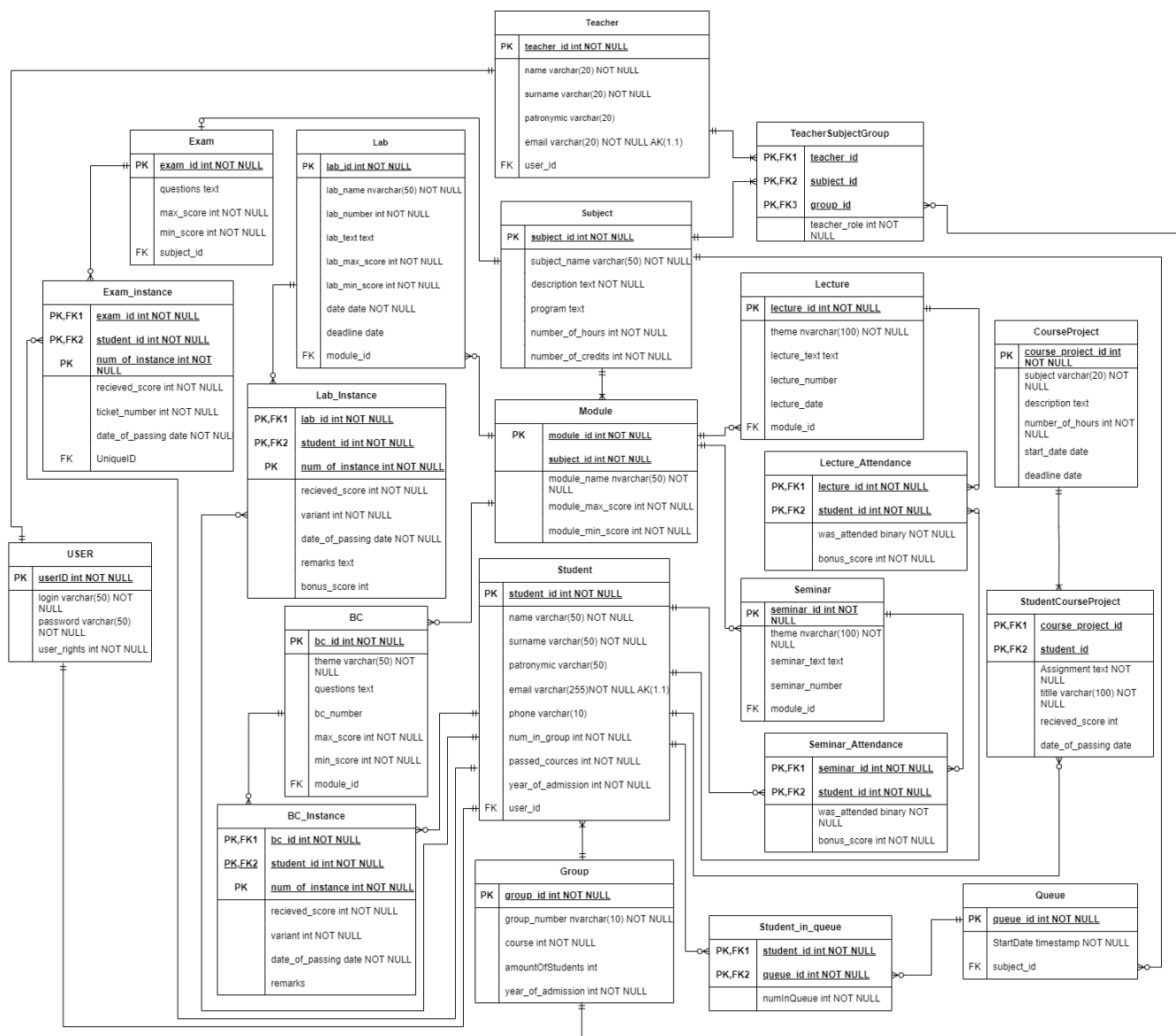


Рисунок 3. Реляционная модель базы данных

Полученная реляционная модель отражает связи между объектами в разрабатываемой базе данных.

## 2.2 Архитектура разрабатываемого приложения

Было решено разработать приложение, состоящее из двух модулей: модуль просмотра академической успеваемости и посещаемости, а также модуль организации очередей студентов.

Первый модуль представляет собой пользовательский интерфейс, через который происходит вход в электронный дневник, а также просмотр данных о дисциплинах, академических заданиях и посещаемости студента. Данный модуль реализует следующую логику: клиент-студент должен авторизоваться в приложении, в таком случае ему доступна домашняя страница электронного дневника с личной информацией и списком доступных курсов, дисциплин в соответствии с номером группы студента. В процессе интерактивного взаимодействия пользователя с интерфейсом, на сервер отправляются запросы, после чего сервер запрашивает необходимые данные из существующей базы, а в результате ответной передачи данных на клиентской стороне генерируются веб-страницы, отображающие полученную информацию в удобном для пользователя виде.

В соответствии с поставленными требованиями в интерфейсе должны быть предусмотрены:

- веб-страница для авторизации пользователей;
- главная страница веб-приложения с полями для просмотра основной информации о пользователе-студенте и его текущих курсах;
- веб-страницы для просмотра детальной информации о выбранной дисциплине: список лабораторных работ, лекций, семинарских занятий и рубежных контролей, а также возможные баллы за каждую работу, даты выдачи и дедлайны работ;
- поле для перехода ко второму модулю приложения – созданию очереди и записи в нее.

В данном модуле используется клиент-серверная архитектура. В таком случае при взаимодействии пользователя (клиента) с интерфейсом запросы будут отправлены на сервер, который в свою очередь делает запросы в базу данных и возвращает полученные данные на сервер, после чего их получает клиент. Описанная выше архитектура особенно эффективна при динамической генерации веб-страниц на стороне клиента, поскольку основным требованием к модулю является корректное отображение информации по выбранным пользователем-студентом предметам.

Второй модуль приложения отвечает за организацию процесса формирования очередей студентов для сдачи лабораторных работ. Его основная задача заключается в получении и обработке запроса от пользователя на запись в очередь по выбранному студентом предмету (дисциплине). Таким образом, рассматриваемый модуль должен реализовывать визуализацию списка актуальных по дате очередей из базы данных, создание новых очередей, а также добавление студента в выбранную им очередь и удаление его из заданной очереди.

Отметим, что важным аспектом разрабатываемого приложения является возможность независимой работы каждого модуля. Так пользователю будет доступен модуль для организации очереди отдельно от модуля просмотра академической успеваемости и посещаемости, и наоборот. Однако, в таком случае возникает необходимость реализации авторизации пользователей при работе как с первым, так и со вторым модулем.

### **2.3 Формирование прав доступа**

В разрабатываемой системе "Электронного дневника студента" определены две роли пользователей: преподаватель и студент. Каждая роль имеет набор прав доступа и привилегий в приложении. Для каждой роли определяются соответствующие права доступа к различным функциям и данным приложения. Так, студент может просматривать собственные оценки за определенные работы по дисциплинам и читать комментарии преподавателя к работам, иметь доступ к данным о посещаемости семинаров и лекций, а также узнавать сроки сдачи

лабораторных работ и даты экзаменов. В свою очередь, преподаватель может оценивать работы студентов, давать комментарии, выставять бонусные баллы и изменять данные студентов и образовательных курсов.

Установка уровня доступа основана на роли пользователя. По этой причине студент не имеет доступа ко всем функциям и данным, с которыми работает преподаватель. Пользователь-студент имеет ограниченный набор веб-страниц, которые он может просматривать, не имея возможности редактировать информацию на них. При этом все попытки пользователя перейти на недоступные ему веб-страницы должны быть пресечены. Например, студентам будут недоступны веб-страницы приложения, через которые преподаватель заполняет данные об успеваемости и посещаемости студентов.

Авторизация в приложении также связана с правами доступа. В электронный дневник могут войти только зарегистрированные пользователи. Преподаватель проводит регистрацию своих студентов, создавая новых пользователей в базе данных. При попытке неавторизованного пользователя войти в электронный дневник, должно появляться сообщение о неверном логине. Зарегистрированные студенты после ввода логина и пароля будут перенаправлены на главную страницу дневника студента, где отображаются текущие курсы учебной программы пользователя.

## **2.4 Алгоритм формирования очередей**

Алгоритм формирования очередей для записи на сдачу лабораторных работ в университете способствует эффективной организации процесса сдачи работ и обеспечивает равные возможности для всех студентов.

Алгоритм формирования очередей состоит из следующих этапов:

1. Выбор дисциплины.
2. Определение временных рамок.
3. Запись студентов в очередь.
4. Очищение очереди по истечении срок действия.

Таким образом, студентам сначала необходимо выбрать дисциплину, по которой предстоит сдача работы. Далее определяется время для очереди на сдачу лабораторных работ, после чего происходит запись студентов в выбранную ими очередь. Большую роль на данном этапе играет установка ограничений и правил: каждый студент может иметь право забронировать только одно место в очереди за раз, чтобы обеспечить равные возможности для всех. Также должны присутствовать возможности для отмены записи в очередь и очистки очереди.

### 3. Программная реализация приложения

#### 3.1. Особенности реализации базы данных

В качестве системы управления базами данных было решено использовать PostgreSQL [4]. Особо значимыми особенностями данной СУБД являются производительность и возможность масштабирования. PostgreSQL эффективно обрабатывает запросы даже при большом объеме данных и высоких нагрузках.

Для начала работы было необходимо реализовать код базы данных. Данный код вынесен в отдельный файл **init.sql** и представляет собой последовательность запросов на языке SQL для создания таблиц и представлений в базе данных. В листинге 1 и листинге 2 представлены примеры того, как определены сущности в базе данных.

##### Листинг 1. Таблица User

```
CREATE TABLE IF NOT EXISTS User (  
    UserID UUID PRIMARY KEY,  
    Login VARCHAR(50) NOT NULL,  
    Passw VARCHAR(50) NOT NULL,  
    UsersRights INT NOT NULL,  
    CONSTRAINT user_unique UNIQUE(Login)  
);
```

##### Листинг 2. Таблица Student

```
CREATE TABLE IF NOT EXISTS Student (  
    StudentID UUID PRIMARY KEY,  
    StudentName VARCHAR(50) NOT NULL,  
    Surname VARCHAR(50) NOT NULL,  
    Patronymic VARCHAR(50),  
    Email VARCHAR(255) UNIQUE NOT NULL,  
    Phone VARCHAR(11) NOT NULL,  
    YearOfAdmission INT NOT NULL,  
    PassedCourses INT NOT NULL,  
    NumInGroup INT NOT NULL,
```

```

    user_id UUID NOT NULL REFERENCES User(UserID),
    group_id UUID NOT NULL REFERENCES StudentGroup(GroupID),
    CONSTRAINT student_unique UNIQUE(user_id, group_id)
);

```

### 3.2. Особенности реализации модуля просмотра академической успеваемости и посещаемости

В соответствии с выбранной архитектурой, было необходимо реализовать клиентскую и серверную части приложения, а также протокол для обмена данными между этими сторонами. Выбор языка реализации обусловлен возможностями для сетевого программирования, а также наличием официального драйвера PostgreSQL. Идеальным решением оказался язык программирования Go [5]. Одной из ключевых возможностей выбранного языка является возможность работы с сетевыми сервисами, а основные инструменты представлены пакетом “net” [6] стандартной библиотеки языка Go, предоставляющий различные низкоуровневые сетевые примитивы, через которые идет взаимодействие по сети. Помимо этого, для выполнения поставленных задач были использованы следующие пакеты: “encoding/json” для обмена данными между клиентом и сервером в формате JSON [7], “html/template” для шаблонов веб-страниц при динамической генерации [8]. А для работы с PostgreSQL в Go были использованы пакет “database/sql” и официальный драйвер Pure Go Postgres driver [9].

Клиентская сторона проекта состоит из директории с шаблонами веб-страниц и основной программы, отвечающей за взаимодействие с сервером и генерацию веб-страниц пользовательского интерфейса. В листинге 3 представлен пример шаблона домашней страницы приложения.

Листинг 3. Шаблон главной страницы

```

{{define "student_main"}}
{{template "header"}}
<main role="main" class="inner cover">

```



```

    <p class="lead" style="margin-bottom: 30px">Добро пожаловать в электронный дневник
студента!</p>
    <div class="student-info" style="outline-style: double; margin-bottom: 50px">
        <p class="lead"> Студент: {{.StudentInfo.Surname}} {{.StudentInfo.Name}}
{{.StudentInfo.Patronymic}}</p>
        <p class="lead"> Группа: {{.StudentInfo.Group}}</p>
        <h6 class="title" style="text-align:center; padding: 20px;">
            <a style="color:#fff" href="/logout">Выйти</a>
        </h6>
    </div>
    <p class="lead" style="font-size: 200%; margin-top: 20px" >Текущие курсы:</p>
    <div class="lead">
        {{range $item := .CourseInfo}}
            <div>
                <a href="{{$.CourseLink}}" class="btn btn-lg btn-light fw-bold"
style="margin-bottom: 20px;">{{$.CourseName}}</a>
            </div>
        {{end}}
    </div>
    <a href="/coursework" class="btn btn-lg btn-light fw-bold" style="margin-
bottom: 20px;">КУРСОВАЯ РАБОТА</a>
    </div>
    </div>
    <p class="lead" style="font-size: 200%; margin-top: 30px">Очередь на сдачу
лабораторных работ</p>
    <div>
        <a href="http://t.me/stud_queue_bot" class="btn btn-lg btn-light fw-bold"
style="margin-top: 10px; margin-bottom: 20px;">ЗАПИСАТЬСЯ</a>
    </div>
</main>
</body>
</html>
{{end}}

```

Для клиента определены различные функции, которые отвечают за соединение с сервером, передачу и получение данных, а также генерацию веб-страниц приложения. В листинге 4 продемонстрирована функция, отвечающая за проверку наличия пользователя в базе данных при его попытке авторизоваться.

Листинг 4. Проверка наличия пользователя в базе данных

```

func check_student(w http.ResponseWriter, r *http.Request) {
    err = r.ParseForm()
    if err != nil {
        log.Println(err)
    }
}

```

```

if r.FormValue("username") != "" && r.FormValue("userpassword") != "" {
    data = UserDetails{
        Username: r.FormValue("username"),
        Password: r.FormValue("userpassword"),
        Succes:  false,
    }
}
// запрос на сервер
conn = connectToServer()
defer conn.Close()
encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
// данные для передачи на сервер
var info proto.LoginInfo
info.Username = data.Username
info.Password = data.Password
send_request(encoder, "check", &info)
// Получение ответа с сервера
var resp proto.Response
if err := decoder.Decode(&resp); err != nil {
    log.Printf("error: %v\n", err)
}
switch resp.Status {
case "failed":
    if resp.Data == nil {
        log.Printf("error: data field is absent in response\n")
    } else {
        var errorMsg string
        if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
            log.Printf("error: malformed data field in response\n")
        } else {
            log.Printf("failed: %s\n", errorMsg)
        }
    }
}
case "ok":
    if resp.Data == nil {
        log.Printf("error: data field is absent in response\n")
    } else {
        var info proto.LoginInfo
        if err := json.Unmarshal(*resp.Data, &info); err != nil {
            log.Printf("error: malformed data field in response\n")
        } else {
            log.Printf("result: %v\n", info.Exists)
            data.Succes = info.Exists
        }
    }
}
default:
    log.Printf("error: server reports unknown status %q\n", resp.Status)
}

```

```

    if data.Succes {
        log.Printf("Successful authentication")
        isAuth = true
        http.Redirect(w, r, "/student", 301)
    } else {
        isAuth = false
        log.Printf("Authentication error: %s is missing from the database",
data.Username)
        http.Redirect(w, r, "/badlogout", 301)
    }
}

```

Для отправки запроса на сервер используется функция **send\_request**, представленная в листинге 5.

#### Листинг 5. Функция отправки запроса на сервер

```

// send_request - вспомогательная функция для передачи запроса с указанной командой
// и данными. Данные могут быть пустыми (nil).
func send_request(encoder *json.Encoder, command string, data interface{}) {
    var raw json.RawMessage
    raw, _ = json.Marshal(data)
    encoder.Encode(&proto.Request{Command: command, Data: &raw})
}

```

В свою очередь серверная часть приложения запускает цикл приема входящих соединений и обслуживает клиентов в отдельной go-программе, делает запросы в базу данных, а потом формирует ответ клиенту и отправляет данные вместе со статусом операции. Для обслуживания клиентов используется специальный метод **serve**, представленный в листинге 6. В случае успешного декодирования сообщения клиента, будет вызван метод **handleRequest**, отвечающий за обработку запроса от клиента.

#### Листинг 6. Метод для обслуживания клиентов

```

// serve - метод, в котором реализован цикл взаимодействия с клиентом.
// метод serve будет вызываться в отдельной go-программе.
func (client *StudentClient) serve() {
    defer client.conn.Close()
    log.Println("new client serve")
}

```

```

decoder := json.NewDecoder(client.conn)
for {
    var req proto.Request
    if err := decoder.Decode(&req); err != nil {
        log.Println("client: cannot decode message", "reason ", err)
        break
    } else {
        log.Println("client received command", req.Command)
        client.handleRequest(&req)
    }
}
}

```

Для отправки ответа клиенту используется метод **respond**, представленный в листинге 7.

Листинг 7. Метод отправки ответа клиенту

```

// respond - вспомогательный метод для передачи ответа с указанным статусом
// и данными. Данные могут быть пустыми (data == nil).
func (client *StudentClient) respond(status string, data interface{}) {
    var raw json.RawMessage
    raw, _ = json.Marshal(data)
    client.enc.Encode(&proto.Response{Status: status, Data: &raw})
}

```

### 3.3. Особенности реализации модуля организации очередей

Данный модуль было решено реализовать на языке Python в формате уникального Телеграм-бота, предоставляющего решение задачи организации процесса формирования очередей студентов при сдаче лабораторных работ.

Код для развертывания Телеграм-бота представляет собой обработчик сообщений от пользователя. Для создания собственного бота было решено использовать библиотеку Telebot [10] – одну из наиболее популярных и простых библиотек для работы с HTTP интерфейсом Telegram Bot API [11]. Данная библиотека предоставляет множество функций, которые позволяют легко создавать ботов, включая поддержку клавиатур, обработку сообщений и многое другое.

Перед написанием кода для бота было необходимо авторизоваться в системе Телеграм и зарегистрировать свой будущий бот, получив уникальный токен для доступа к HTTP API, чтобы пользователи смогли работать с приложением.

В реализации бота было решено использовать поддержку клавиатур, таким образом предоставляя пользователю интуитивно понятный интерфейс. В таком случае основная логика Телеграм-бота заключается в следующем: функции-обработчики сообщений бота в зависимости от полученного сообщения отправляют пользователю сообщение и генерируют набор кнопок, одну из которых пользователь должен выбрать для продолжения работы с ботом. В случае, если необходимо пользовательское сообщение, не являющееся одной из сгенерированных кнопок, то бот отправит пользователю в каком формате требуется написать сообщение.

Новая сессия пользователя с ботом начинается после получения ботом сообщения `/start`, обработчик данной команды представлен в листинге 8. Ответ бота на команду старта выглядит как текст с приветствием и просьбой нажать на сгенерированную кнопку ниже для продолжения работы.

Листинг 8. Функция-обработчик команды `/start`

```
@bot.message_handler(commands=['start'])
def start(message):
    db.auth = False
    db.subj_choice = False
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    btn1 = types.KeyboardButton("👉 Привет, запиши меня в очередь")
    markup.add(btn1)
    bot.send_message(message.from_user.id, "👉 - Привет, Я - бот-помощник для  
организации очереди на сдачу лабораторных работ! Для записи в очередь нажми на кнопку  
ниже", reply_markup=markup)
```

Далее любое входящее текстовое сообщение от пользователя обрабатывается в функции `get_text_messages`, полный код которой представлен в Приложении 1. Данная функция содержит набор условных операторов, проверяющих содержимое

сообщения и генерирующих ответ пользователю. Так, например, при получении от пользователя сообщения о создании новой очереди на текущую дату, выполняется фрагмент кода этой функции, представленный в листинге 9.

Листинг 9. Обработка команды «Создать очередь на сегодня»

```
elif message.text == 'Создать очередь на сегодня':
    conn, cur = connectDB()
    queue_date = datetime.datetime.now()
    date = queue_date.date()
    time = queue_date.time()
    text = 'Очередь {day}-{month}-{year} {hour}:{min}'.format(day=date.day,
                                                                month=date.month, year=date.year, hour=time.hour,
                                                                min=time.minute)
    db.queue_date[text] = queue_date
    print(text, db.queue_date[text])
    cur.execute("INSERT INTO Queue(QueueID, StartDate, subject_id) VALUES
(gen_random_uuid(), %s, %s)",
                (queue_date, db.current_subject.id))
    cur.execute('select QueueID from Queue where StartDate = %s', (queue_date,))
    cqid = cur.fetchone()[0]
    cur.close()
    conn.commit()
    conn.close()
    db.current_queue.id = cqid
    db.current_queue.date = queue_date
    db.current_queue.subject_id = db.current_subject.id
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    btn1 = types.KeyboardButton('Записаться')
    btn2 = types.KeyboardButton('Вернуться к списку очередей')
    markup.add(btn1, btn2)
    bot.send_message(message.from_user.id, text + ' создана',
reply_markup=markup)
```

При этом всю логику созданного Телеграм-бота можно описать следующим образом:

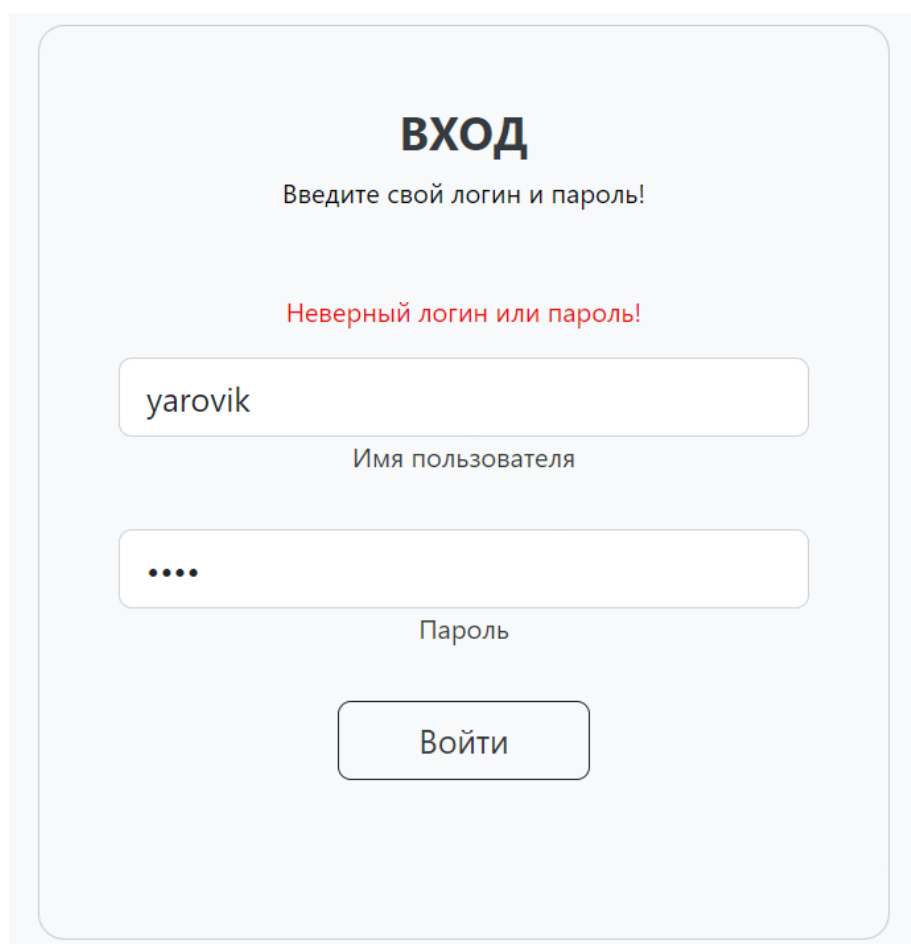
1. Пользователь вводит команду `"/start"`, запуская бота. После чего при желании пользователя записаться в очередь, бот запрашивает логин для авторизации в электронном дневнике. При этом у неавторизованных пользователей нет возможности записаться в очередь.

2. В случае успешной авторизации, бот, сделав запрос к базе данных, выведет список кнопок с текущими курсами студента. Пользователю необходимо выбрать предмет, в очередь по которому он хочет записаться.
3. После выбора предмета бот делает запрос в базу данных для получения информации об актуальных по дате очередях. Возможны следующие варианты событий в соответствии с полученной выборкой данных:
  - 3.1. Актуальных очередей нет, бот отправит сообщение с предложением создать новую очередь. После создания очереди, в нее можно будет записаться, либо вернуться к списку актуальных очередей.
  - 3.2. Список актуальных очередей не пуст. Пользователь может выбрать существующую очередь или создать новую.
4. Когда очередь выбрана, бот предложит просмотреть очередь (узнать, кто уже в нее записан), подтвердить запись в очередь или вернуться к списку очередей.
5. При выборе кнопки «Просмотреть очередь» бот выводит сообщение со списком записавшихся в очередь. После просмотра можно подтвердить запись, удалить свою запись или же вернуться к списку актуальных очередей.
6. При выборе «Вернуться к списку очередей» бот выведет список актуальных по дате очередей и даст возможность создать новую очередь (см. пункт 3).
7. При выборе «Подтвердить запись» бот выведет сообщение с текстом «Вы записаны в очередь» с указанием, в какую очередь пользователь записался. После подтверждения можно просмотреть очередь, удалить свою запись или же вернуться к списку актуальных очередей.
8. Очередь очищается автоматически по истечении суток. Таким образом, все очереди, созданные раньше текущего дня, будут очищены и не будут отображаться в списке актуальных очередей.

## 4. Тестирование приложения

Цель тестирования заключается в проверке работоспособности основных функций приложения, связанных с хранением, обработкой и визуализацией информации, а также поиске недочетов реализации, которые выражаются в неудобстве использования приложения.

Первоначально было необходимо проверить корректную работоспособность функции авторизации пользователей: при наличии соответствующей записи в базе данных после попытки входа, пользователь будет перенаправлен на главную страницу, в противном случае будет показано сообщение о неверных данных аккаунта. Для проверки был выполнен вход в аккаунт пользователя, которого нет в базе данных. На рисунке 4 показано, что получено сообщение об ошибке операции входа.



**ВХОД**

Введите свой логин и пароль!

Неверный логин или пароль!

yarovik

Имя пользователя

....

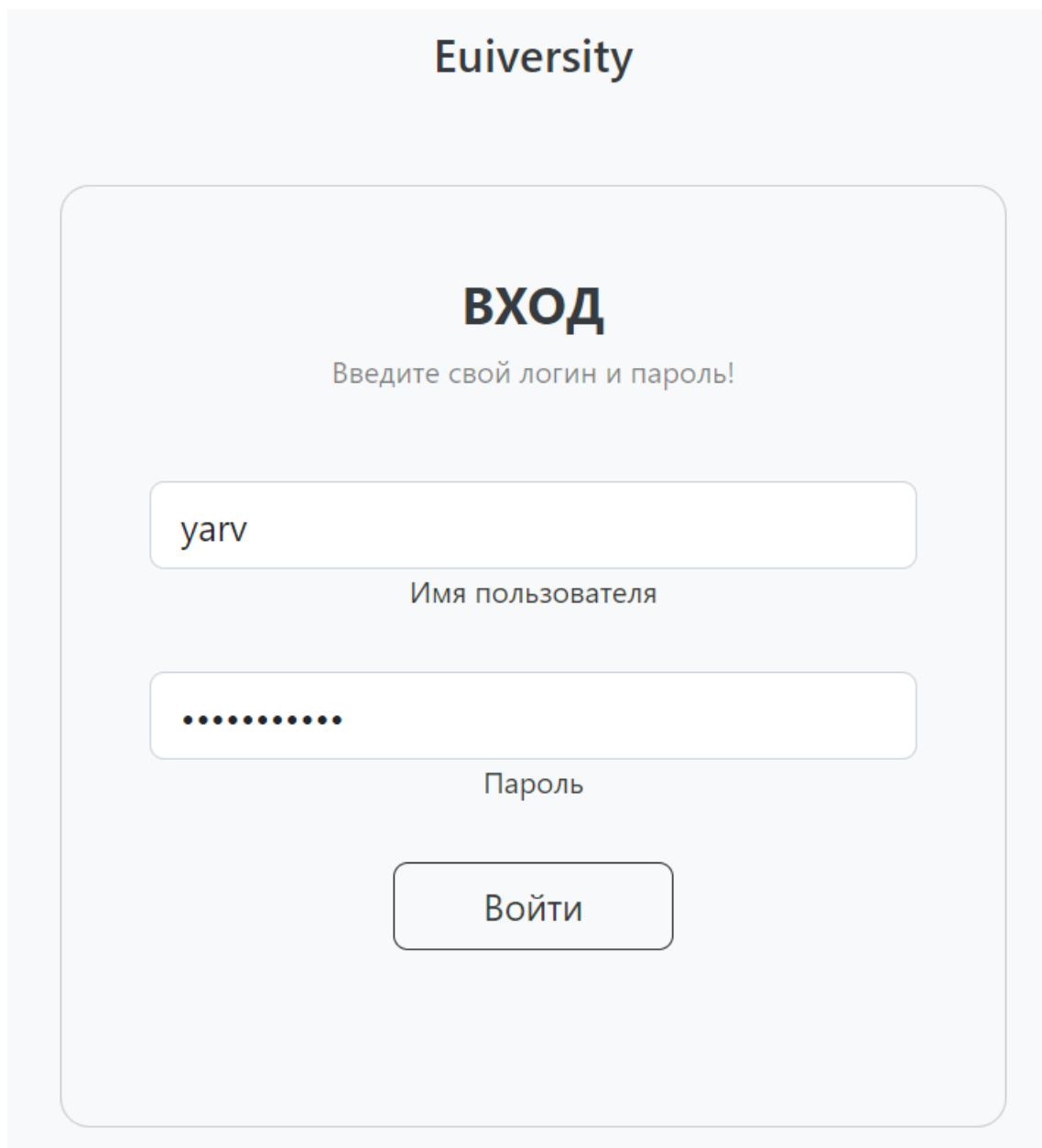
Пароль

Войти

Рисунок 4. Неуспешная попытка авторизации пользователя



Далее была произведена попытка входа в аккаунт существующего в базе пользователя (рисунок 5). После успешной авторизации пользователь был перенаправлен на домашнюю страницу приложения, что показано на рисунке 6.



The image shows a login interface for 'Euiversity'. At the top, the word 'Euiversity' is displayed in a dark blue font. Below it, the word 'ВХОД' (Login) is centered in a large, bold, black font. Underneath 'ВХОД', the text 'Введите свой логин и пароль!' (Enter your login and password!) is centered in a smaller, gray font. There are two input fields: the first one contains the text 'yary' and is labeled 'Имя пользователя' (Username) below it; the second one contains ten black dots representing a password and is labeled 'Пароль' (Password) below it. At the bottom of the form, there is a button labeled 'Войти' (Login) in a black font.

Рисунок 5. Успешная попытка авторизации пользователя

## Euiversity

Добро пожаловать в электронный дневник студента!

Студент: Яровикова Анастасия Сергеевна

Группа: ИУ9-61Б

[Выйти](#)

Текущие курсы:

**БАЗЫ ДАННЫХ**

**РПиРП**

**КУРСОВАЯ РАБОТА**

Очередь на сдачу лабораторных работ

**ЗАПИСАТЬСЯ**

Рисунок 6. Домашняя страница приложения

Также на данном этапе тестирования была проверена защищенность всех страниц приложения от прямого перехода, минуя авторизацию. При корректной работе сервиса попытки неавторизованного пользователя перейти на другие страницы будут остановлены, а самому пользователю будет доступна только страница входа.

Следующим этапом тестирования стала проверка функциональности главной (домашней) страницы приложения. На данной странице должны отображаться данные о студенте (ФИО и группа), список текущих дисциплин в виде кнопок с возможностью перехода по ним на страницы с детальной информацией о курсах, а также кнопка со ссылкой на Телеграм-бот для записи в очередь на сдачу лабораторных работ. Для проверки были созданы соответствующие записи в таблицах базы данных, а благодаря использованию шаблонов языка Golang в реализации, данные отображались корректно в соответствии с авторизованным пользователем (рисунок 6).

Далее была проверена корректность отображаемых данных на страницах дисциплин. Отметим, что любой учебный курс имеет свои особенности:

- может заканчиваться экзаменом или зачетом;
- может содержать один или несколько модулей;
- каждый модуль может включать как все виды образовательных занятий (лабораторные работы, лекции, семинары, рубежные контроли), так и только часть из них.

В соответствии с чем на странице курса должны отображаться только корректные разделы, например, если в курсе нет семинарских занятий, то пользователю не видна таблица с информацией о семинарах. Такая ситуация показана на рисунках 7-10, где для тестирования был выбран курс «Базы данных», который содержит два модуля, лекции, лабораторные работы и два рубежного контроля, но не имеет в программе семинаров.

Экзамен		
Дата	Минимальный балл	Максимальный балл
2023-01-15	18	30

Рисунок 7. Страница курса. Экзамен

Euiversity

[Вернуться к курсам](#)

Базы данных

Лабораторные работы

Модуль 1									
Номер	Дата	Дедлайн	Название работы	Тема	Мин./ макс. балл	Попытка сдачи	Полученный балл	Бонусный балл	Комментарий преподавателя
№1	2022-09-09	2022-09-16	ER	Моделирование данных с использованием модели сущность-связь.	5/8	1	8	1	БД отлично спроектирована!
№2	2022-09-16	2022-09-23	SOM	Моделирование данных с использованием модели семантических объектов.	4/5	0	0	0	-
№3	2022-09-23	2022-09-30	ER->R	Преобразование модели "сущность-связь" в реляционную модель.	5/7	0	0	0	-
№4	2022-09-30	2022-10-07	SOM->R	Преобразование модели семантических объектов в реляционную модель.	4/5	0	0	0	-
Модуль 2									
Номер	Дата	Дедлайн	Название работы	Тема	Мин./ макс. балл	Попытка сдачи	Полученный балл	Бонусный балл	Комментарий преподавателя
№5	2022-10-07	2022-10-14	DB files	Операции с базой данных, файлами, схемами.	2/3	0	0	0	-
№6	2022-10-14	2022-10-21	Constraints	Ключи, ограничения, значения по умолчанию.	2/3	0	0	0	-
№7	2022-10-21	2022-10-28	Views	Представления и индексы.	2/3	1	1	0	ЛР сдана позже дедлайна
№8	2022-10-28	2022-11-11	SP/Cursor/F	Хранимые процедуры, курсоры, пользовательские функции.	3/4	0	0	0	-

Рисунок 8. Страница курса. Лабораторные работы

Лекции. Посещаемость				
Модуль 1				
Лекция	Дата	Тема		<div>Присутствие</div> <div>Бонусный балл</div>
№1	2022-09-10	Введение		<div>+</div> <div>0</div>
№3	2022-10-10	Модель "сущность-связь"		<div>+</div> <div>1</div>
Модуль 2				
Лекция	Дата	Тема		<div>Присутствие</div> <div>Бонусный балл</div>
№7	2022-11-20	СУБД SQL Server 2012 – Представления. Индексы.		<div>+</div> <div>0</div>
№14	2022-12-15	СУБД SQL Server 2012 – Создание распределенных баз данных.		<div>+</div> <div>0</div>

Рисунок 9. Страница курса. Лекции

## Рубежные контроли

Модуль 1						
РК	Дата	Мин./макс. балл	Попытка сдачи	Вариант	Полученный балл	Комментарий преподавателя
№1	-	5/10	0	0	0	-
Модуль 2						
РК	Дата	Мин./макс. балл	Попытка сдачи	Вариант	Полученный балл	Комментарий преподавателя
№2	2022-12-23	5/10	1	29	5	-

Рисунок 10. Страница курса. Рубежные контроли

Последним этапом являлось тестирование Телеграм-бота. Для проверки работоспособности бота было смулировано взаимодействие нескольких пользователей с разных устройств. Ниже, на рисунках 11-20 проиллюстрирована работа созданного Телеграм-бота, показывающая корректное выполнение команд и соответствие логике приложения.

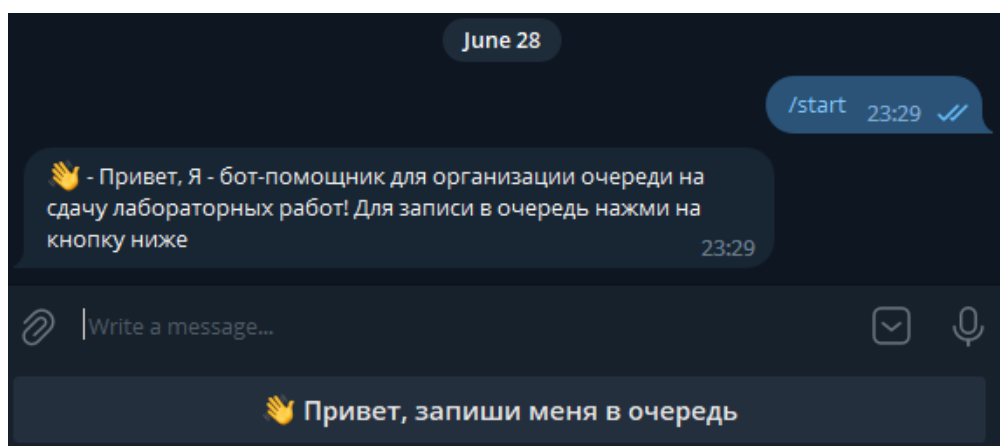


Рисунок 11. Стартовое меню.

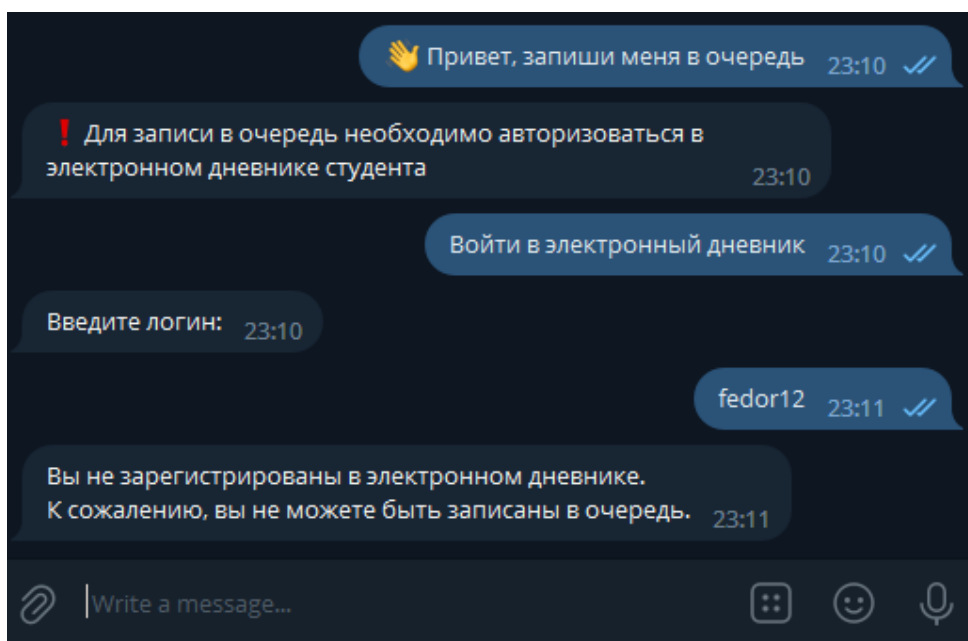


Рисунок 12. Неуспешная авторизация

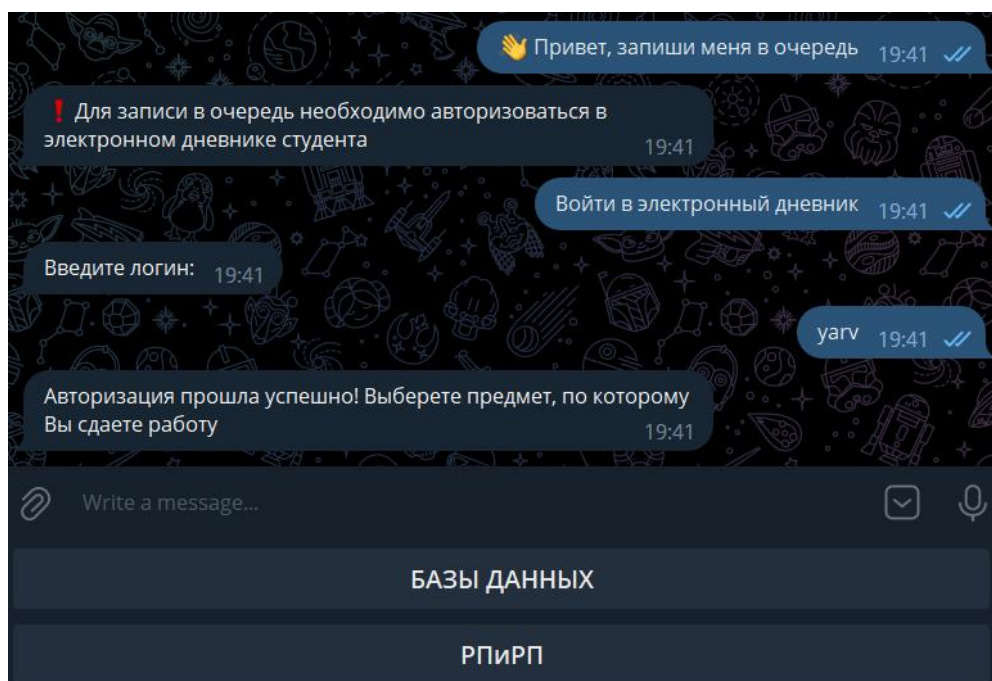


Рисунок 13. Успешная авторизация. Выбор дисциплины

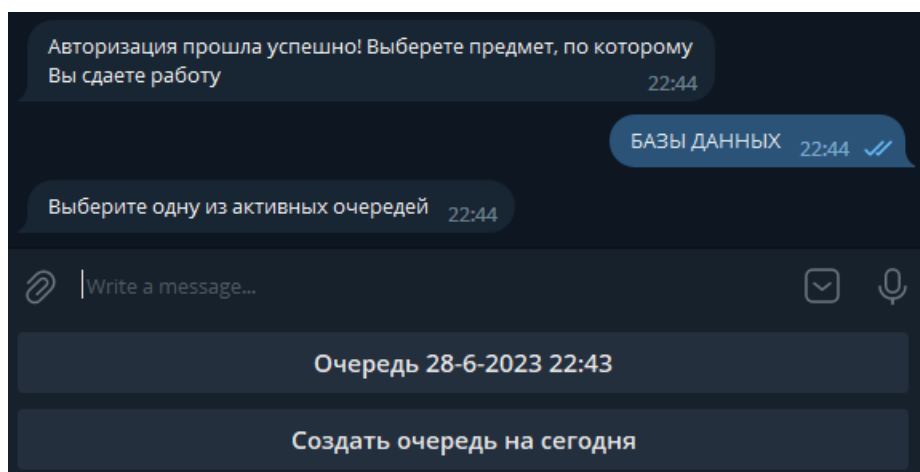


Рисунок 14. Представление актуальных очередей

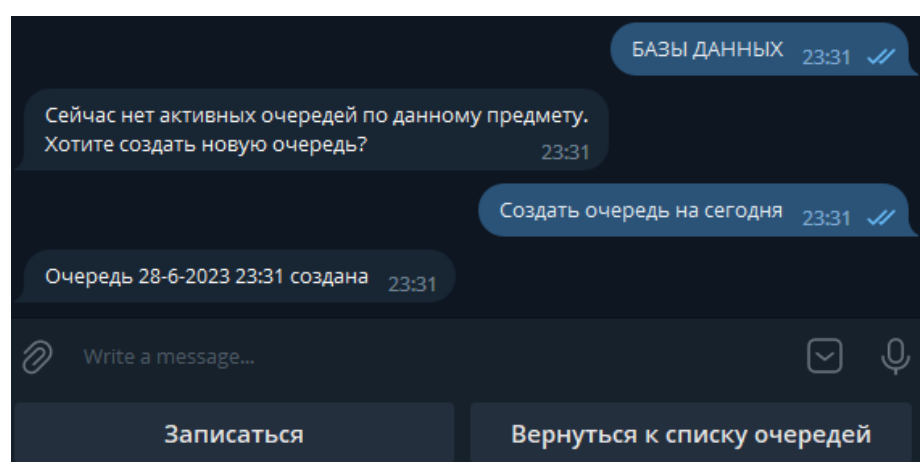


Рисунок 15. Создание новой очереди

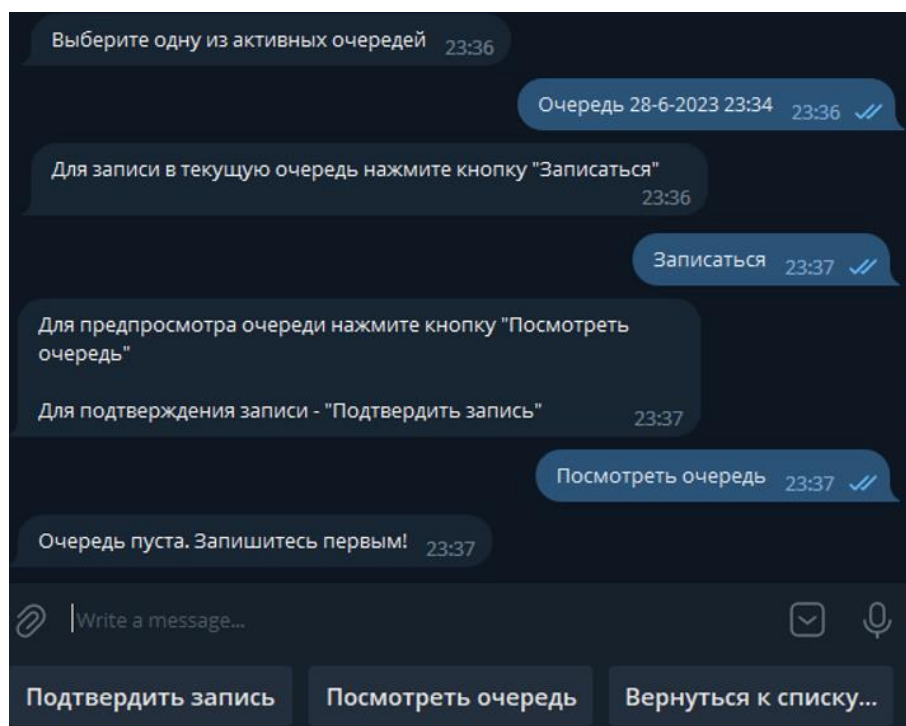


Рисунок 16. Запись в очередь с предпросмотром

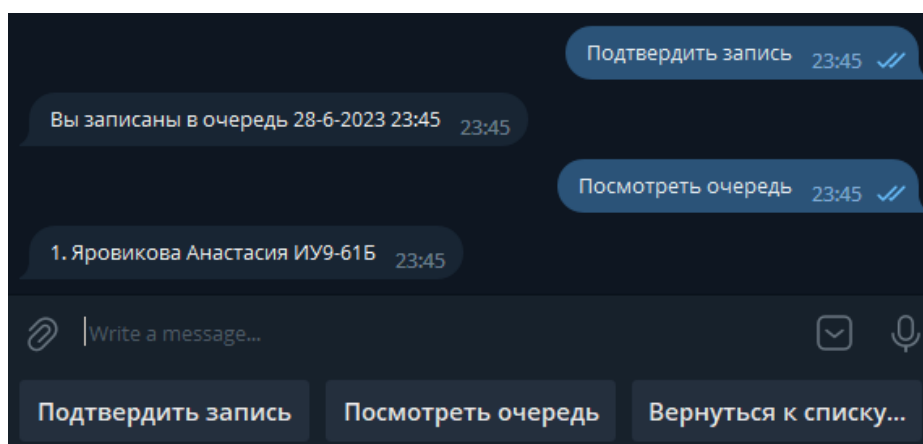


Рисунок 17. Подтверждение записи. Просмотр очереди



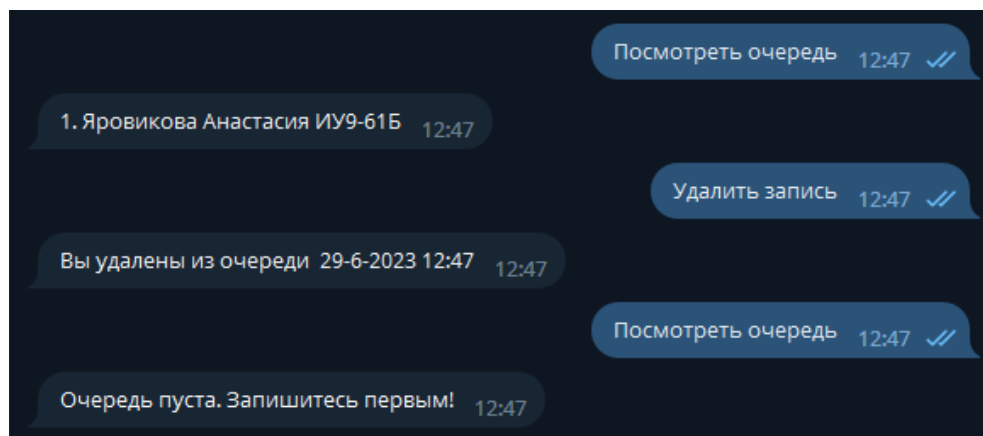
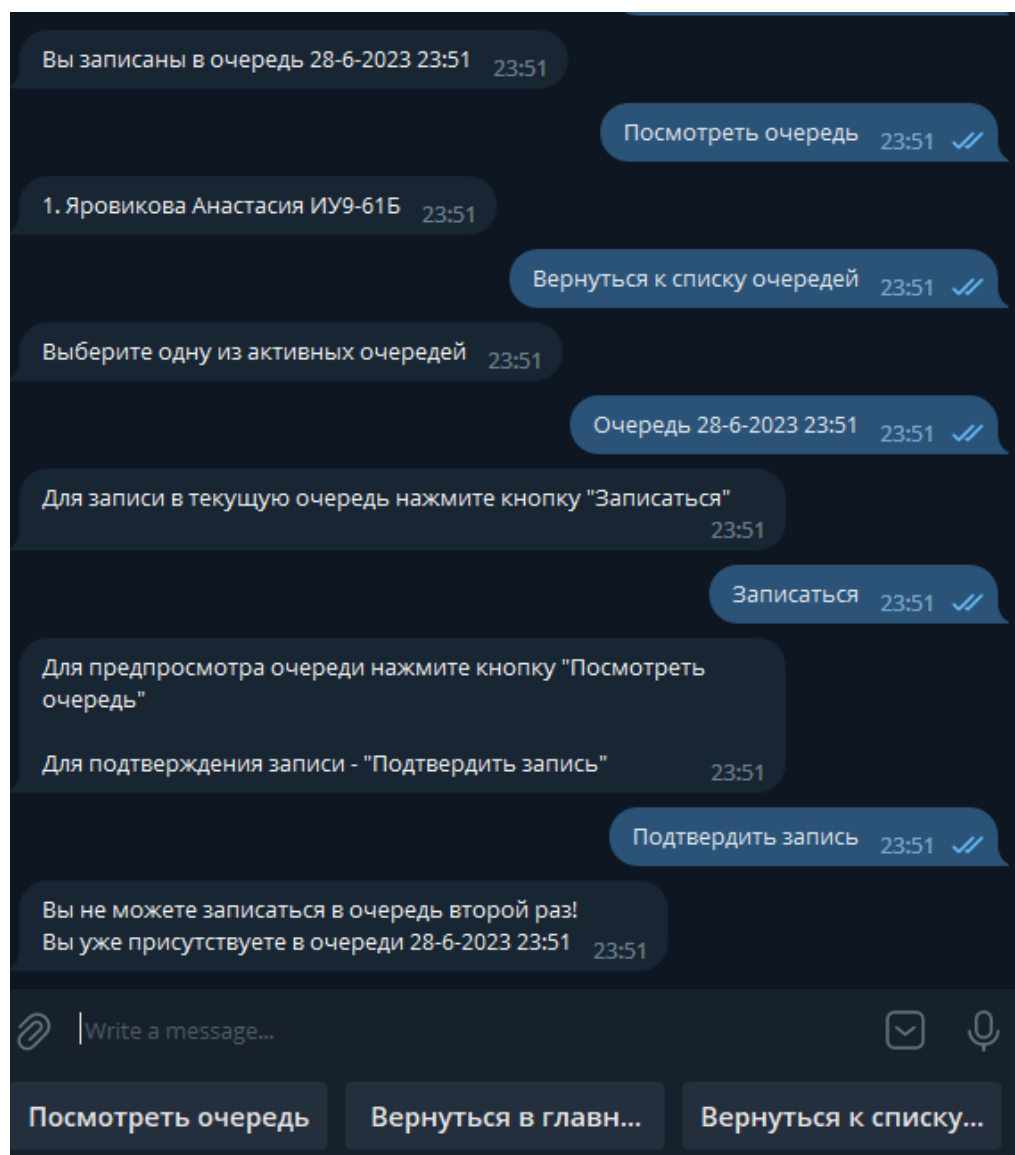


Рисунок 18. Удаление из очереди



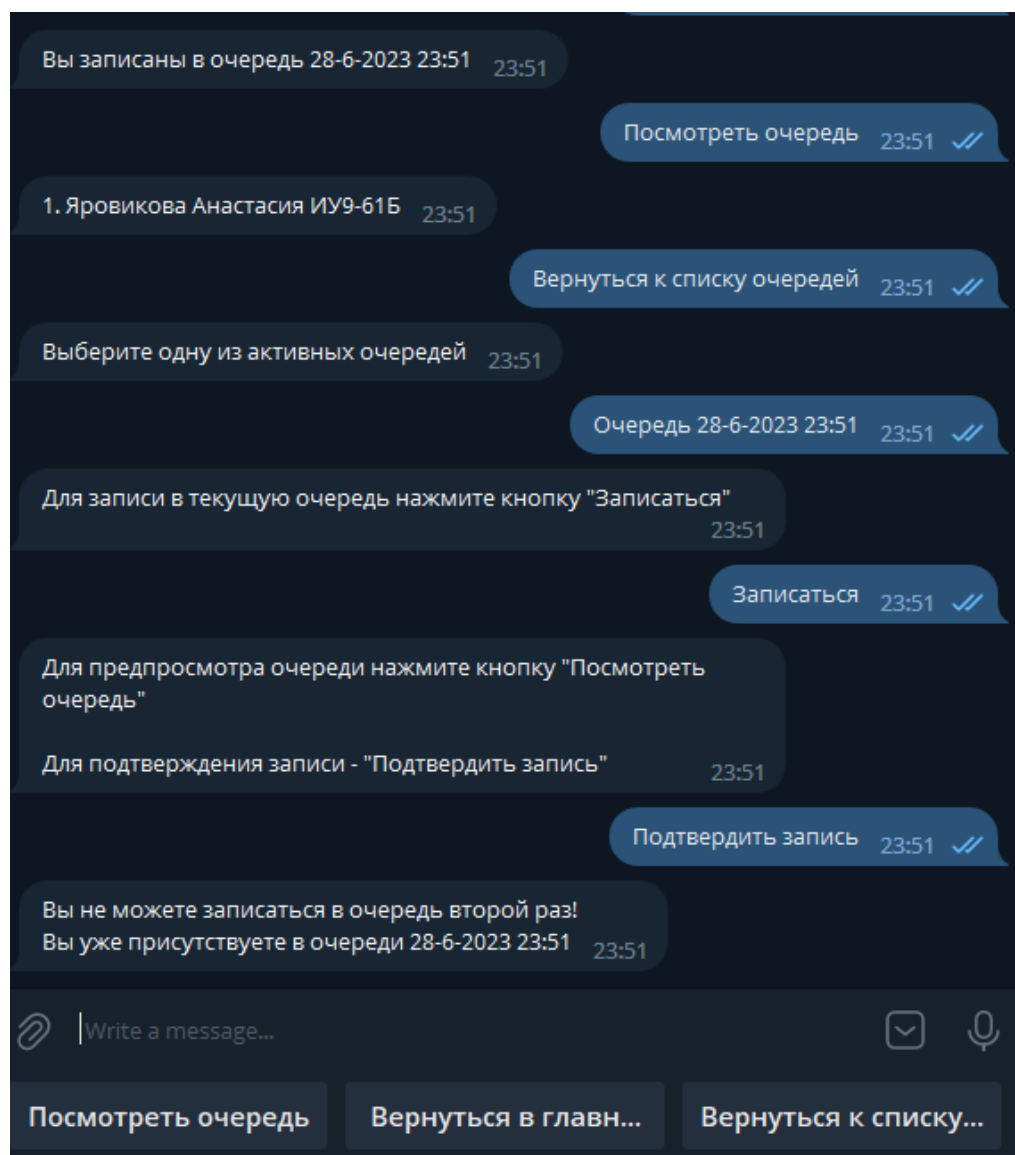


Рисунок 19. Попытка повторной записи в очередь при условии уже существующей записи

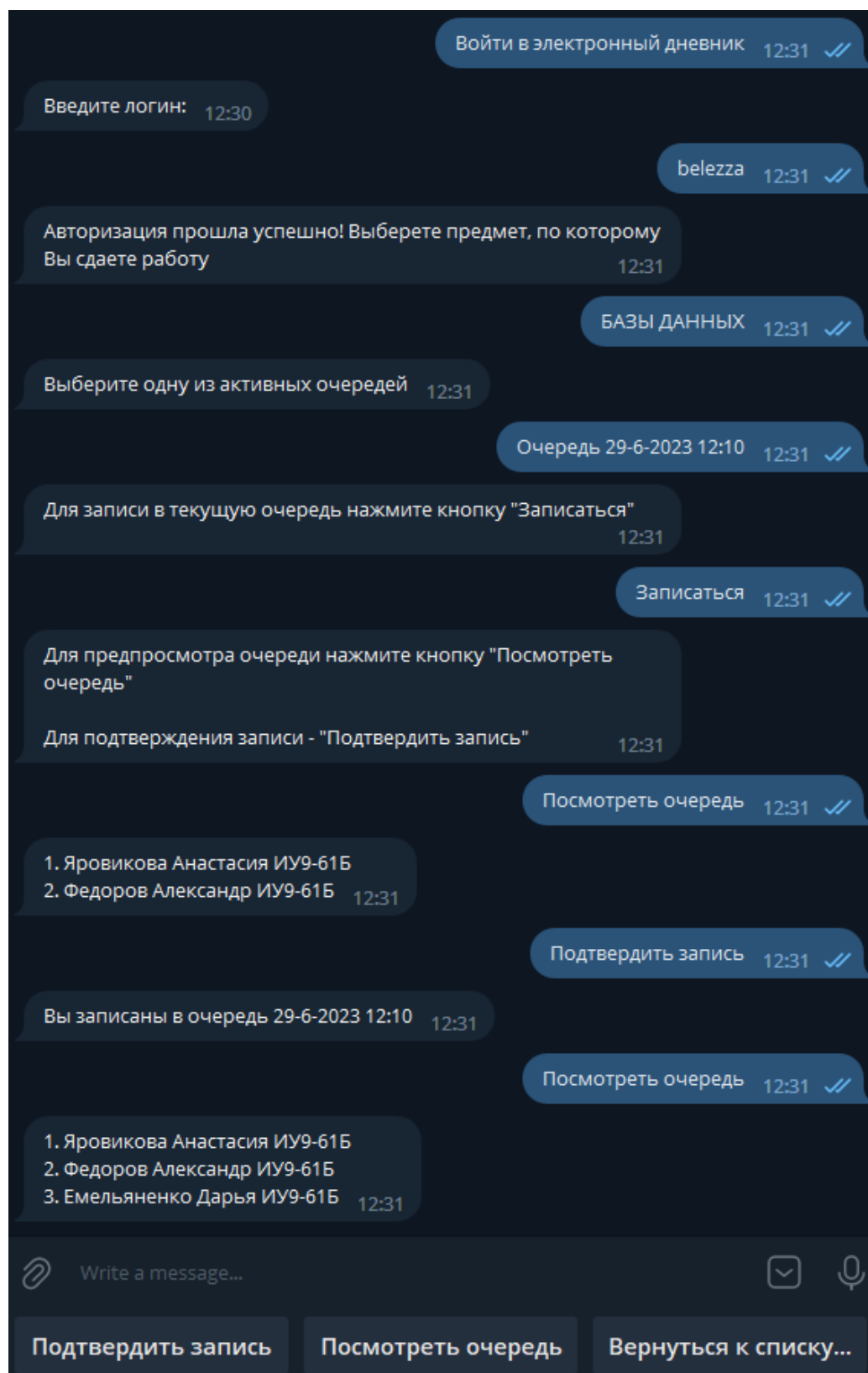


Рисунок 20. Запись в очередь новых пользователей

Отметим, что выявленные на этапе тестирования потенциальные проблемы были устранены, вследствие чего был сделан вывод о корректной работе всех функций приложения в соответствии с предопределенной логикой, а финальная версия разработанного проекта продемонстрировала свою готовность к использованию.

## ЗАКЛЮЧЕНИЕ

В результате данной курсовой работы было успешно создано клиент-серверное приложение для хранения данных об академической успеваемости и посещаемости студентов, которое позволяет пользователям получать детальную информацию о текущих курсах и учебных заданиях, а также следить за своим образовательным прогрессом. Кроме того, был создан уникальный Телеграм-бот, который предоставляет студентам решение вопроса формирования организованной очереди на сдачу лабораторных работ. Для реализации приложения была использована система управления базами данных PostgreSQL, языки программирования Golang и Python, а также фреймворк Bootstrap.

В процессе работы были изучены и применены на практике основы проектирования баз данных, а также методы работы с ними с помощью языка SQL. Были разработаны модель «сущность-связь» и реляционная модель базы данных. К созданной базе были написаны запросы на языке SQL, позволяющие по различным критериям получать необходимые записи из таблиц.

Также были изучены основы веб-разработки с использованием возможностей языка Golang, а именно шаблонов для генерации динамических страниц и сетевых интерфейсов для обработки взаимодействий по сети.

Нельзя не отметить то, что проект имеет потенциал для дальнейшего развития и совершенствования. Одной из возможностей для улучшения пользовательского опыта является добавление учебного расписания в приложение. Это позволит студентам эффективно организовывать свой образовательный процесс, отслеживая предстоящие занятия. Также стоит обратить внимание на развитие созданного Телеграм-бота. Бот может быть доработан путем добавления возможности топологической сортировки очереди по различным критериям, например по количеству сданных работ у студентов. Эти улучшения помогут расширить функциональные возможности проекта и сделать разработанное приложение более привлекательным для пользователей.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Фёдоров А.А. Проектирование баз данных: учебник для вузов. – Москва, Издательство «Горячая линия-Телеком», 2017.
2. Смирнов И.И. Введение в базы данных: учебное пособие. – Санкт-Петербург, Издательство «БХВ-Петербург», 2016.
3. Петров В.В. Базы данных: учебник для вузов. Москва, Издательство «ИНФРА-М», 2018.
4. PostgreSQL: The World's Most Advanced Open Source Relational Database – URL: [www.postgresql.org](http://www.postgresql.org) (дата обращения: 28.06.2023)
5. The Go Programming Language – URL: [go.dev](http://go.dev) (дата обращения: 28.06.2023).
6. Go Packages – net package – URL: [pkg.go.dev/net](http://pkg.go.dev/net) (дата обращения: 28.06.2023).
7. Go Packages – json package – URL: [pkg.go.dev/encoding/json](http://pkg.go.dev/encoding/json) (дата обращения: 28.06.2023).
8. Go Packages – template package – URL: [pkg.go.dev/html/template](http://pkg.go.dev/html/template) (дата обращения: 28.06.2023).
9. Pure Go Postgres driver – URL: [pkg.go.dev/github.com/lib/pq](http://pkg.go.dev/github.com/lib/pq) (дата обращения: 28.06.2023).
10. pyTelegramBotAPI – URL: [pypi.org/project/pyTelegramBotAPI](http://pypi.org/project/pyTelegramBotAPI) (дата обращения: 28.06.2023).
11. Telegram Bot API. – URL: [core.telegram.org/bots/api](http://core.telegram.org/bots/api) (дата обращения: 28.06.2023).

## Приложение 1. Функция-обработчик входных сообщений боту

```
@bot.message_handler(content_types=['text'])
def get_text_messages(message):
    if message.text == '👋 Привет, запиши меня в очередь':
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        btn1 = types.KeyboardButton('Войти в электронный дневник')
        markup.add(btn1)
        bot.send_message(message.from_user.id, '! Для записи в очередь необходимо авторизоваться в электронном дневнике студента', reply_markup=markup)

    elif message.text == 'Войти в электронный дневник':
        bot.send_message(message.from_user.id, 'Введите логин:')
        conn, cur = connectDB()
        cur.execute('SELECT * from Users')
        usrs = cur.fetchall()
        for user in usrs:
            db.user_data.add(user[1])
        cur.execute('SELECT SubjectID, Description from Subject')
        subjects = cur.fetchall()
        for s in subjects:
            db.subj_data[s[1]] = s[0]
        cur.close()
        conn.close()

    elif message.text in db.user_data and not db.auth:
        db.subj_choice = False
        db.login = message.text
        conn, cur = connectDB()
        cur.execute('SELECT StudentID from Student where user_id = (SELECT UserID from Users where Login=%s)', (db.login,))
        db.current_student_id = cur.fetchone()[0]
        db.auth = True
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        for s in db.subj_data.keys():
            btn1 = types.KeyboardButton(s)
            markup.add(btn1)
        bot.send_message(message.from_user.id, 'Авторизация прошла успешно! Выберите предмет, по которому Вы сдаете работу', reply_markup=markup)

    elif message.text == 'Вернуться в главное меню':
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        for s in db.subj_data.keys():
            btn1 = types.KeyboardButton(s)
            markup.add(btn1)
        bot.send_message(message.from_user.id, 'Выберете предмет, по которому Вы сдаете работу', reply_markup=markup)
```



```

elif message.text not in db.user_data and not db.auth:
    bot.send_message(message.from_user.id, 'Вы не зарегистрированы в электронном
дневнике.\nК сожалению, вы не можете быть записаны в очередь.')

elif message.text in db.subj_data.keys() and not db.subj_choice:
    db.subj_choice = True
    db.current_subject = db.Subject(db.subj_data[message.text], message.text)
    conn, cur = connectDB()

    cur.execute('delete from Queue where subject_id = %s and StartDate < %s',
                (db.current_subject.id, datetime.datetime.now().date()))
    conn.commit()

    cur.execute('SELECT * from Queue where subject_id = %s and StartDate >= %s',
                (db.current_subject.id, datetime.datetime.now().date()))
    active_queues = cur.fetchall()
    if len(active_queues) == 0:
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        btn1 = types.KeyboardButton('Создать очередь на сегодня')
        markup.add(btn1)
        bot.send_message(message.from_user.id, 'Сейчас нет активных очередей по
данному предмету.\nХотите создать новую очередь?', reply_markup=markup)
    else:
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        for q in active_queues:
            date = q[1].date()
            time = q[1].time()
            name = 'Очередь {day}-{month}-{year}
{hour}:{min}'.format(day=date.day, month=date.month,
                                                                year=date.y
ear, hour=time.hour, min=time.minute)
            btn1 = types.KeyboardButton(name)
            markup.add(btn1)
        markup.add(types.KeyboardButton('Создать очередь на сегодня'))
        bot.send_message(message.from_user.id, 'Выберите одну из активных
очередей', reply_markup=markup)
        cur.close()
        conn.close()

elif message.text == 'Создать очередь на сегодня':
    conn, cur = connectDB()
    queue_date = datetime.datetime.now()
    date = queue_date.date()
    time = queue_date.time()
    text = 'Очередь {day}-{month}-{year} {hour}:{min}'.format(day=date.day,
                                                                month=date.month, year=date.year, hour=time.hour,
                                                                min=time.minute)

```

```

db.queue_date[text] = queue_date
print(text, db.queue_date[text])
cur.execute("INSERT INTO Queue(QueueID, StartDate, subject_id) VALUES
(gen_random_uuid(), %s, %s)",
            (queue_date, db.current_subject.id))
cur.execute('select QueueID from Queue where StartDate = %s', (queue_date,))
cqid = cur.fetchone()[0]
cur.close()
conn.commit()
conn.close()
db.current_queue.id = cqid
db.current_queue.date = queue_date
db.current_queue.subject_id = db.current_subject.id
markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn1 = types.KeyboardButton('Записаться')
btn2 = types.KeyboardButton('Вернуться к списку очередей')
markup.add(btn1, btn2)
bot.send_message(message.from_user.id, text + ' создана',
reply_markup=markup)

elif re.fullmatch("Очередь \\d{1,2}\\-\\d{1,2}\\-\\d{4} \\d{1,2}\\:\\d{1,2}",
message.text):
    print(db.queue_date)
    norm_date = db.queue_date[message.text]
    conn, cur = connectDB()
    cur.execute('SELECT * from Queue where StartDate = %s', (norm_date,))
    cur_queue = cur.fetchone()
    db.current_queue.id = cur_queue[0]
    db.current_queue.date = cur_queue[1]
    db.current_queue.subject_id = db.current_subject.id
    cur.close()
    conn.close()
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    btn1 = types.KeyboardButton('Записаться')
    btn2 = types.KeyboardButton('Вернуться к списку очередей')
    markup.add(btn1, btn2)
    bot.send_message(message.from_user.id, 'Для записи в текущую очередь нажмите
кнопку "Записаться"', reply_markup=markup)

elif message.text == 'Вернуться к списку очередей':
    conn, cur = connectDB()
    cur.execute('SELECT * from Queue where subject_id = %s and StartDate >= %s',
                (db.current_subject.id, datetime.datetime.now().date()))
    active_queues = cur.fetchall()
    if len(active_queues) == 0:
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        btn1 = types.KeyboardButton('Создать очередь на сегодня')
        markup.add(btn1)

```

```

        bot.send_message(message.from_user.id, 'Сейчас нет активных очередей по
данному предмету.\nХотите создать новую очередь?', reply_markup=markup)
    else:
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        for q in active_queues:
            print(q)
            date = q[1].date()
            time = q[1].time()
            name = 'Очередь {day}-{month}-{year}
{hour}:{min}'.format(day=date.day, month=date.month,
                                                                year=date.y
ear, hour=time.hour, min=time.minute)
            btn1 = types.KeyboardButton(name)
            markup.add(btn1)
            markup.add(types.KeyboardButton('Создать очередь на сегодня'))
            bot.send_message(message.from_user.id, 'Выберите одну из активных
очередей', reply_markup=markup)
            cur.close()
            conn.close()

    elif message.text == 'Записаться':
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        btn1 = types.KeyboardButton('Подтвердить запись')
        btn2 = types.KeyboardButton('Посмотреть очередь')
        markup.add(btn1, btn2)
        bot.send_message(message.from_user.id, 'Для предпросмотра очереди нажмите
кнопку "Посмотреть очередь"\n\nДля подтверждения записи - "Подтвердить запись"',
reply_markup=markup)

    elif message.text == 'Посмотреть очередь':
        conn, cur = connectDB()
        cur.execute('SELECT NumInQueue, student_id from StudentQueue where queue_id =
%s order by NumInQueue', (db.current_queue.id,))
        cur_queue = cur.fetchall()
        print(cur_queue)
        print(db.current_queue.id)
        if len(cur_queue) == 0:
            queue_list = 'Очередь пуста. Запишитесь первым!'
        else:
            queue_list = ''
            for s in cur_queue:
                cur.execute('SELECT Surname, StudentName, group_id from Student where
StudentID = %s', (s[1],))
                student = cur.fetchone()
                cur.execute('SELECT GroupName from StudentGroup where GroupID = %s',
(student[2],))
                g = cur.fetchone()

```

```

        stud = '{surname} {name} {group}'.format(surname=student[0],
name=student[1], group=g[0])
        queue_list += '{num}. {name}\n'.format(num=s[0], name=stud)
    cur.close()
    conn.close()
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    btn1 = types.KeyboardButton('Подтвердить запись')
    btn2 = types.KeyboardButton('Посмотреть очередь')
    btn3 = types.KeyboardButton('Вернуться к списку очередей')
    markup.add(btn1, btn2, btn3)
    if len(cur_queue) > 0:
        btn = types.KeyboardButton('Удалить запись')
        markup.add(btn)
    bot.send_message(message.from_user.id, queue_list, reply_markup=markup)

elif message.text == 'Подтвердить запись':
    conn, cur = connectDB()
    cur.execute('SELECT 1 FROM StudentInQueue WHERE student_id = %s and queue_id
= %s',
                (db.current_student_id, db.current_queue.id,))
    res = cur.fetchall()

    if len(res) == 1:
        cur.close()
        conn.close()
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        btn1 = types.KeyboardButton('Посмотреть очередь')
        btn2 = types.KeyboardButton('Вернуться в главное меню')
        btn3 = types.KeyboardButton('Вернуться к списку очередей')
        markup.add(btn1, btn2, btn3)
        date = db.current_queue.date.date()
        time = db.current_queue.date.time()
        text = 'Вы не можете записаться в очередь второй раз!\nВы уже
присутствуете в очереди {day}-{month}-{year} {hour}:{min}'.format(
            day=date.day, month=date.month, year=date.year, hour=time.hour,
min=time.minute)
        bot.send_message(message.from_user.id, text, reply_markup=markup)
    else:
        cur.execute('INSERT INTO StudentInQueue(student_id, queue_id, NumInQueue)
VALUES (%s, %s, (SELECT COUNT(*) + 1 from StudentInQueue where queue_id = %s))',
                    (db.current_student_id, db.current_queue.id,
db.current_queue.id))
        cur.close()
        conn.commit()
        conn.close()
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        btn1 = types.KeyboardButton('Посмотреть очередь')
        btn2 = types.KeyboardButton('Вернуться в главное меню')

```

```

markup.add(btn1, btn2)
date = db.current_queue.date.date()
time = db.current_queue.date.time()
text = 'Вы записаны в очередь {day}-{month}-{year} {hour}:{min}'.format(
    day=date.day, month=date.month, year=date.year, hour=time.hour,
min=time.minute)

    bot.send_message(message.from_user.id, text, reply_markup=markup)
elif message.text == 'Удалить запись':
    conn, cur = connectDB()
    cur.execute('delete from StudentQueue where student_id =%s',
(db.current_student_id,))
    conn.commit()
    cur.close()
    conn.close()
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    btn1 = types.KeyboardButton('Посмотреть очередь')
    btn2 = types.KeyboardButton('Вернуться в главное меню')
    markup.add(btn1, btn2)
    date = db.current_queue.date.date()
    time = db.current_queue.date.time()
    text = 'Вы удалены из очереди {day}-{month}-{year} {hour}:{min}'.format(
        day=date.day, month=date.month, year=date.year, hour=time.hour,
min=time.minute)
    bot.send_message(message.from_user.id, text, reply_markup=markup)

```