**Слайд 2.**

Основная цель проекта – это разработать централизованную систему здравоохранения, позволяющую усовершенствовать информационную и программную поддержку основных процессов поликлиник.

**Слайд 3.**

Современная медицинская сфера сталкивается с проблемами, такими как длительное ожидание приема у врача, неэффективная организация расписания, трудности доступа к медицинской истории и огромное количество бумажных документов.

Проект "ЕСАП" предлагает централизованную электронную систему для решения этих проблем и улучшения качества медицинского обслуживания.

**Слайд 4.**

Разработанная система предоставляет следующий функционал:

* Врач:
* Может просматривать расписание приема пациентов;
* Может создавать, изменять и удалять записи на прием;
* Может просматривать медицинскую историю пациента;
* Может добавлять записи;
* Может вносить диагнозы и рекомендации для пациентов;
* Регистратор:
* Может создавать новых пациентов в системе;
* Может записывать пациентов на прием к врачам;
* Может просматривать и редактировать информацию о пациентах;
* Может проверять доступность и свободное время в расписании врачей;

**Слайд 5.**

В основу структуры системы заложен принцип REST API (Representational State Transfer Application Programming Interface) — это архитектурный подход, который используется для создания и предоставления веб-сервисов. Он основан на протоколе HTTP и включает в себя набор методов запросов, таких как GET, POST, PUT и DELETE. Каждый метод имеет свое предназначение и позволяет взаимодействовать с ресурсами.

Это позволяет разделить систему на два независимых проекта. Такой подход повышает эффективность создания приложения, позволяет решить проблему конфликта общего кода и упрощает разделение задач между разработчиками.

**Слайд 6.**

Для системы была разработана следующая диаграмма классов. Основными классами-сущностями разрабатываемого прототипа программной системы являются: роль, врач, поликлиника, пациент, медицинская карта и запись в ней, анализ, расписание врача и записи встреч.

**Слайд 7.**

Также в процессе разработки прототипа программной системы была разработана диаграмма классов уровня бизнес-логики.

В ходе разработки системы были реализованы следующие компоненты бизнес-слоя:

* AppointmentService;
* ClinicService;
* DoctorDetailsService;
* DoctorService;
* MedicalCardService;
* PatientService;
* RegistrationService;
* ScheduleService.

**Слайд 8.**

Далее рассмотрим подробнее сервис с логикой работы расписания ScheduleService.

Данный метод используется для создания нового расписания:

1. Операции, выполняемые внутри метода, выполняются в рамках одной транзакции базы данных. Это обеспечивает целостность данных и автоматическую управляемость транзакциями.
2. Сначала метод ищет врача (Doctor) по указанному идентификатору. Если врач не найден, выбрасывается исключение NotFoundException.
3. Затем метод проверяет, существует ли уже расписание для указанной даты и врача. Если расписание уже существует, выбрасывается исключение NotCreateException с сообщением о том, что расписание уже существует.
4. Если расписание еще не существует, создается новый объект расписания на основе dto объекта. Затем проверяется корректность временного интервала расписания.
5. Если временной интервал некорректен (например, длительность меньше или равна 0 или не кратна 30 минутам), выбрасывается исключение NotCreateException с сообщением о некорректном времени расписания.
6. В конце метода расписание сохраняется в базе данных.

**Слайд 9-13.**

…

**Слайд 14.**

В процессе работы была обнаружена проблема, связанная с "N+1" запросами, которая приводит к долгой обработке запросов. Для решения этой проблемы были разработаны собственные SQL-запросы, исключившие использование ORM.

**Слайд 15.**

Данный класс ApiExceptionHandler является обработчиком исключений для REST API. Он использует аннотацию @RestControllerAdvice, чтобы обрабатывать исключения, возникающие в контроллерах при обработке запросов.

У класса есть несколько методов-обработчиков (handleNotCreate, handleOrderNotFound, authenticationException), каждый из которых обрабатывает конкретный тип исключения.

Если происходит исключение типа NotCreateException, то метод handleNotCreate создает объект ApiError с соответствующим статусом BAD\_REQUEST и сообщением об ошибке. Аналогично, для исключений типа NotFoundException и AuthenticationException создаются соответствующие объекты ApiError.

Таким образом, класс ApiExceptionHandler обрабатывает исключения и возвращает ответы с информацией об ошибке и соответствующим статусом HTTP.

**Слайд 15.**

На данном слайде представлена демонстрация работы прототипа пс.