Федеральное государственное образовательное бюджетное
учреждение высшего образования
**«Финансовый университет при Правительстве
Российской Федерации»**

Департамент анализа данных, принятия решений и финансовых технологий

Курсовая работа по дисциплине
«Современные технологии программирования»
на тему:

«Информационная система аэропорта»

Выполнил:
студент ПИ19-1В
Галкин Н.В.

Научный руководитель:
к.п.н Горохова Р.И.

Москва

2021

Обзорный план

Титульный лист Оглавление Введение

1. Постановка задачи
2. Описание предметной области
3. Актуальность автоматизации
4. Описание программы
    1. Алгоритмические решения
    2. Описание интерфейса программы
    3. Состав приложения
5. Назначение и состав классов программы

   Заключение Список литературы Приложение. 1. Исходный код программы

0. Введение

Аэропорт это предприятие повышенной важности для любого государства. Аэропорты, для некоторых стран и городов, в силу их географического положения, являются точками жизненной опоры общества. Только через них в таких местах есть возможность снабжения важнейшими припасами.

В крупных же городах, аэропорт это произведение технического искусства, включающее в себя все последние достижения техники, огромные финансовые потоки, доскональный контроль и наблюдение за всеми системами организации и её участниками, безупречное следование должностным инструкциям.

Информационная система аэропорта позволяет управлять и контролировать все эти процессы с большей точностью, своевременностью, прозрачностью.

1. Постановка задачи

Вкратце, Задача этой курсовой работы состоит в следующем - отработка навыка проектирования системы баз данных, тренировка построения архитектуры приложения. Использование профессионального навыка создания мобильных

Подробнее, В серверной части курсовой работы, я планирую реализовать следующие элементы:

Базы данных

- БД самолетов
- БД рейсов

- БД основного персонала(пилоты, стюарды, инженеры, админ персонал, охрана, сортировка, другие)
- БД билетов к каждому рейсу.

Автоматизация

К каждой базе данных будет создан класс её обслуживания - сервисный класс. Например, к бд билетов по рейсу, будет реализована возможность покупки билета, выбора места, место будет отмечаться далее в бд как недоступное. Кроме того, можно будет отменить бронь места.

Рейсы планируется генерировать автоматически вместе с информацией по ним.

В фронтенд части работы, я сделать акцент на следующем функционале:

1. Поиск рейсов и покупка билетов для пассажира
2. Взаимодействие с базой данных работников
3. Взаимодействие с базой данных рейсов
4. Взаимодействие с базой данных самолетов

В целом, реализовать весь функционал информационной системы аэропорта крайне сложно

даже команде из специалистов, занятых этим на полной ставке. В реальной системе требуется автоматизация огромного количества функций персонала администрации аэровокзала. К тому же, такая система должна быть защищена по высшим государственным стандартам, с соответствующими аудитами и сертификацией. Соответственно, моя работа совершенно не претендует на полноту реализации такой системы. Своей основной задачей я вижу примерное, далеко не исчерпывающее описание баз данных, базовые интеракции с ними, позволяющие управлять ими в полной мере, но не оптимизированные для разных сотрудников(погрузчику может быть удобней работать с индивидуальными номерами багажа, а регистрации пассажиров - с их ФИО и документами)

К вопросу, о том, почему я выбрал разработку приложения для айфона в качестве реализации клиентской части, а не JavaFX. Целью курсовой работы является оттачивание профессиональных навыков, а я специализируюсь на айос разработке, именно поэтому я и принял это решение.

2. Описание предметной области

Поскольку аэропорт это предприятие особой сложности, включающее в себя большое количество различных систем, людей, профессий, техники и прочего, я сконцентрируюсь только на следующих частях этой системы: Базы данных информационной системы аэропорта, сервизный слой, содержащий в себе функции для обработки запросов к базе данных и выполнения прочих рутинных задач, веб-слой, обрабатывает поступающие соединения из интернета и возвращает обратно клиенту результат обработки. Кроме того, поверх этих слоев будет создан слой защиты информации(авторизации). Отдельно от сервера, стоит фронт-энд часть - iOS приложение. Оно будет связывать с сервером по сети.

3. Актуальность автоматизации

Аэропорт это очень массивное предприятие, в котором важна точность работы - каждая минута опоздания самолета, каждая поломка, потеря багажа, кризисная ситуация - могут стоить огромных сумм денег и даже человеческих потерь. Именно поэтому так важно автоматизировать такой тип систем. Поскольку

благодаря налаженной автоматизации, многократно снижается риск человеческой ошибки, снижаются издержки на обслуживание как самих вычислительных мощностей, так и издержки инфраструктуры аэропорта.

Кроме того, появляется пространство для повышения прибыли предприятия за счет более точно анализа происходящих в нем процессов и более гибкого/простого управления ими.

Конечно, человеческое участие, как и везде, полностью не исключается. Все еще остаются ситуации, где техника не сможет справиться без надзора специалиста. С другой стороны, такие ситуации редки и от наблюдателя в первую очередь требуется навык разрешения кризисных ситуаций. В таком случае, не так важно, через какие интерфейсы производится управление, сколько важна хорошая задукоментированность системы, её устойчивость, доступ к любым «кишкам» системы напрямую.

Другой крайне полезной особенностью информационных систем в крупных предприятиях является возможность достичь синнергичности всех данных и подсистем. Например, по количеству въезжающих на парковку машин и прибывающих на

ж/д станцию аэропорта людей, можно заранее настроить поток воздуха, который вентиляция должна поставить в аэропорт. Или же, можно проследить все точки взаимодействия лица с аэропортом; в целях безопасности, повышения прибыли, эффективности, скорости работы.

4. Описание программы

Архитектура программы - DDD - Main controller + Service + REPO + Domain.

Перечисление функций программы: FlightService:

- ListAll() - выводит список всех рейсов выполняемых из аэропорта.
- findByCity(String startCity, String endCity) - выполняет поиск рейсов осуществляющих перелет из startCity в endCity
- findByDate(Date date) - выполняет поиск рейсов, вылетающих в указанную дату.
- findById(Long id) - выполняет поиск среди рейсов по указанному id.

- findByCountries(String startCountry, String endCountry) - выполняет поиск среди рейсов по указанной стране вылета и стране прилета.
- addNew(Flight flight) - создает новую запись в базе данных рейсов с внесенными данными.
- deleteFlightById(Long id) - выполняет поиск среди рейсов по указанному id и в случае успешного поиска - удаление
- Геттеры и сеттеры к каждому поля класса

EmployeeSevice

- ListAll() - выводит список всех сотрудников аэропорта
- addNew(Employee employee) - создает новую запись в базе данных сотрудников с внесенными данными.
- findByRole(String role) - поиск среди сотрудников по указанной занятости
- findByLicense(String license) - поиск среди сотрудников, в списках лицензий, который имеет соответсвующий сотрудник
- findByVisa(String country) - поиск среди сотрудников, по спискам виз сотрудников
- findByCountryOfOrigin(String country) - поиск среди сотрудников по гражданству

- deleteEmployeeById(Long id) - выполняет поиск среди сотрудников по указанному id и в случае успешного поиска - удаление

Эти два сервиса еще были реализованы на сервисном слое, позже я отказался от этой схемы по указанным в другой главе причинам.

BaggageService - по сути реализован в двух классах - контроллере и в репозитории. В контроллере «бизнес-логика», в репозитории указан только sql запрос соответствующей функции, поскольку репозиторий это интерфейс, а не класс. Так что, далее будет идти листинг функций, отвечающий за работу с соответствующей базой данных.

Baggage

- listAllBaggage() - выводит список всех багажей, находящихся в аэропорте.
- findBaggageByOwnerFullname(String ownerFullName) - возвращает багаж(1+), принадлежащий указанному лицу.
- findBaggageByFlightId(Long flightId) - возвращает багаж(1+), который будет перевозиться на указанному рейсе.

- findBaggageByTargetFlightName(String targetFlightName) - возвращает багаж(1+), по указанному рейсу, однако, имя рейса тут вводится в международном формате.
- findBaggageByStartCountry(String startCountry) - возвращает багаж(1+), который был отправлен из укащанной страны.
- findBaggageByMiddleCountry(String middleCountry) - возвращает багаж(1+), который должен лететь через указанную страну, это может быть полезно для рейсов с пересадками.
- findBaggageByEndCountry(String endCountry) - озвращает багаж(1+), по указанной стране назначения.
- addBaggage(@Baggage baggage) - добавляет указанный багаж в базу данных.
- deleteBaggageById(Long id) - выполняет поиск среди багажей по указанному id и в случае успешного поиска - производит удаление.

Planes

- addPlane(Planes plane) - добавляет указанный самолет в базу данных.

- listAllPlanes() - выводит список всех самолетов, находящихся в аэропорте.
- findPlanesByInternationalNumber(String number) - производит поиск по базе самолетов, выводит самолет с указанным международным идентификатором
- findPlanesByRussianNumber(String russianNumber) - выполняет поиск и возвращает самолет по внутреннему всероссийскому индентификатору
- findPlanesByOwnerCompany(String company) - возвращает все самолеты, принадлежащие указанной компании(авиа)
- findPlanesByPersonnelCapacity(Integer personnelCapacity) - возвращает все самолёты по пассажирскому объему равные указанному
- findPlanesByBaggageCapacity(Integer baggageCapacity) - возвращает все самолёты по багажному объему равные указанному
- findPlanesByFlightHistoryId(Long flightID) - возвращает самолет, если в списке истории его полетов есть айди указанного полета
- deletePlaneById(Long id) - выполняет поиск среди самолетов по указанному id и в случае успешного поиска - производит удаление.

5. Алгоритмические решения

В работе, я принял решение использовать реактивный подход и соответственно реактивные фреймворки, поскольку что такой способ написания программ мне хорошо знаком по айос разработке. Мои основные фреймворки там реактивные. Да и, на мой взгляд, когда речь идет о RESTful JSON системе, гораздо лучше взаимодействия описываются не при декларативном, а при реактивном подходе.

Архитектура у меня многослойная, описанная в следующих главах. Такая архитектура позволяет разделить код, чтобы не создавать «кашу» в основном контроллере - massive controller, что особо важно, когда проект растет в размерах и количестве программистов, его поддерживающих. С другой стороны, как я указал ранее, я отказался от этого подхода на определенном моменте, поскольку мой проект не такой большой и функции достаточно маленькие, чтобы компактно размещаться в репозитории и контроллере.

Каких-то особых алгоритмических решений я не реализовывал, кроме как архитектурных. Пока писал с сервисным слоем, неоптимизированно извлекал из базы

данных все данные по конкретной таблице, а потом искал уже внутри этих данных с помощью встроенной функции java filter. Потом, я перешел к более оптимизированному и правильному подходу: писал запросы в чистиком sql коде из репозитория. В теории, это должно ускорить запросы в o(n)-раз и уменьшить обьем занимаемой памяти в те же o(n)-раз.

В фронтенд части курсовой работы об интерфейсе которой, я, подробней расскажу в следующей главе, я разделил приложение на несколько «таб»-экранов. В целом, приложение достаточно простое, в основном состоит из List(графических списков), таб-бар реализует перемещение между экранами с помощью кнопок расположенных внизу пользовательского интерфейса, LazyVGrid - та же таблица, но с расширенными возможностями по настройки столбиков, пространства между ними прочего. И список и решетка реализуют так называемую «ленивую подгрузку» - это значит, что элементы таблицы подгружаются в оперативную память и редерятся только в момент их непосредственного отображения пользователю. Кроме того, возможность взаимодействия с базой данных на функциональных экранах, таких как: список сотрудников, список самолетов, список багажа -

реализована через смахивание карточек, стандартной функции многих программ в этой операционной системе. Т.е. Если пользователь хочет удалить багаж из базы данных - он может это сделать, включив режим редактирования по кнопку сверху, смахнув карточку багажа, подтвердив своё действие через уведомление. Тоже самое и с добавлением элементов в таблицу. По тапу по карточке в любом из списков - открывается страница с детальными данными по этой записи. Будь-то чемодан, мы бы увидели данные его владельца, направление его движения и прочее, т.е. полная строка из базы данных по этому объекту.

Обращение к любым другим функциям, не реализованным в приложении, возможно через веб интерфейс стандартных http запросов, где, аргументы к запросу можно передавать через знак амперсанд в строке браузера. Но так же, никто не мешает их делать и через консоль браузера. К тому же, для создания нового объекта, будь-то самолет или багаж, необходимо прислать JSON файл с полным описанием создаваемого объекта.

6. Описание интерфейса программы

Поскольку сервер информационной системы аэропорта содержит большое количество функций для разных видов персонала, то и доступ, даже на уровне интерфейса должен быть раздельным. Но это в реальных условиях. В мое же случае, я принял решение сконцентрироваться на реализации клиентского интерфейса покупки билетов, а так же, на реализации упрощенного доступа к базам данных самолетов, рейсов, багажа. Под упрощенным, я понимаю, минимально необходимый функционал для работы с базой данных. Т.е. создание, удаление, поиск(не только по id). В «билетной» же части, я хотел создать достаточно приятный интерфейс и плюс-минус все реальные поля в существующих системах по продаже билетов. Поля такие как: выбор места(мест) на рейсе(с временным бронированием до оплаты), экран с указанием документов, имен и т.д., страничка оплаты(реальный сервис оплаты, я конечно подключать не планирую), отправка итога на север. Кроме того, я планирую реализовать в iOS приложении клиентское табло с рейсами и гейтами аэропорта. **Табло рейсов** На табло рейсов в графически приятной форме отображается

список рейсов с указанием времени отлета и прилета(зависит от того, прибывающий это или отбывающий рейс), авиакомпания, направление полета(если исходящий), стойка регистрации(если исходящий), гейт(если исходящий)

Их состав примерно таков

Экран билеты: Таблица с рейсами из нашего аэропорта, с указанием города отправления, названия рейса, авиакомпании, времени полета, городе назначения, стоимостью. Данные в этой таблице можно отфильтровать с помощью функций сверху: город назначения, временной интервал, длительность полета, количество людей для которых покупаются билеты.

По нажатию на любую из карточек рейсов, пользователь перенесится на следующий экран - выбор места. Там перед ним схематично обозначенная таблица с местами(как при реальной рассадке в самолете), места пронумерованы, занятые - отмечены, количество и расстановка мест зависит от количество мест на модели самолета. Пользователь может выбрать одно и более мест, далее нажать на «ок» и будет перенесен на экран оформления билетов. В зависимости от количества отмеченных мест, представлено выбранное количество

полей для документов на каждого летящего. После устранения этой формальности, пользователь может нажать на кнопку подтвердить и будет перевесе на экран с оплатой. Оплата не функционирует и приведена лишь как дизайн макет, после «оплаты» - отправляется запрос на сервер и по успешному ответу оттуда, человек становится обладателем своих билетов(запрос всегда успешный и всегда показываются билеты)

Весь «красивый, сжатый функционал» будет доступен в виде приложения под айфон. Навигация по разделам будет осуществляться с помощью «бара вкладок» в нижней части экрана. Первый экран - с билетами будет доступен любому пользователю скачавшему приложение. Как и второй экран с электродным табло аэропорта со сведениями о рейсах. Все прочие экраны могут быть доступны только после ввода токена, который в теории должен быть получен у администратора системы. В токене содержатся сведения о роли пользователя и его пароль. Поскольку подключение планируется осуществлять по защищённому протоколу, никакой угрозы безопасности такой подход обозначать не должен. Введенный токен будет сохранен на устройстве и доступ к доступным экранам потерян не будет.

7. Состав приложения

Архитектура приложения вкратце

- Контроллер связи с сетью
- Домен - образец данных в базе данных
- Репозиторий базы данных
- Сервисный слой для обработки «бизнес-логики»
- Защитный слой, частично интегрирован со всеми вышеуказанными слоями приложения

Использованные библиотеки

Сервер

- JAVA
- JAVA-Reactor
- R2DBC
- Mustache
- WebFlux
- FlyWay
- JDBC
- Lombok
- PostgreSQL

айОС приложение

- Swift
- Foundation
- SwiftUI
- Introspect

8. Назначение и состав классов программы

Архитектура сервеной части такая:

- Main Countroller - отвечает за связи с сетью, принятием сетевых запросов и возвращение информации по запрашиваемым функциям.
- Domain - разметка(модель) базы данных для самого приложения, по этой разметке, Фреймворк, отвечающий за связь с сервером базы данных понимает чего от него ожидать.
- Repo - сокр. от репозиторий - интерфейс протокола ReactiveCrudRepository, через который уже непосредственно производится подключение к базе данных.
- Сервисный слой - здесь описывается «бизнес-логика» приложения. Первое время, я использовал его как посредник между базой данных и главным контроллером, но потом принял решение, реализовывать запросы в интерфейсе репозитория,

а сами функции делать в мэйн контроллере. Так что сейчас у меня уживаются две архитектуры в одном приложении. Я понимаю, что это неправильно, но я принял такое решение по следующим доводам: времени мало и сервисный слой это лишняя редекларация уже существующей, достаточно короткой и несложной логически функции. Так почему бы мне не писать взаимодействия сразу в контроллере. К тому же, понятно, что проект учебный и расти во что-то серьезное он не будет, потому проблемы архитектуры не скажутся на качестве продута на дальних этапах разработки.

- Защитный слой - слой отвечающий за безопасность доступа к приложению, в базе данных хранит захешированные ключи доступа к приложению(чтобы даже в случае утечки ключи в базе данных оказались бесполезными для взломщика). Ограничивает подключение к определенному функционалу из вне без соответствующего ключа доступа. Реализует разграничение доступов по типу пользовательского допуска. Является гарантом надежности системы и безопасности сотрудников, пассажиров.

9. Заключение

В ходе проведенной работы я действительно узнал для себя много нового. Узнал, о существовании различных фрейворков для джава Спринг, с помощью которых мне было очень легко поднять сервер, создать все связи с базой данных. В сравнении с моим прошлым опытом подобной работы на Питоне и фреймворке Фласк, этот был куда приятнее, проще. Постоянно было ощущение, что сейчас я что-то напишу и оно просто будет работать, без «танцев с бубном» - удивительный опыт. С другой, архитектурный стороны, мне было интересно и, как я и планировал выбирая именно эту тему курсовой работы, полезно учиться проектировать информационную систему столь крупного предприятия. Мне пришлось подумать о ее многослойности, разделении функционала на разные доменные области, базы данных, чтобы если что-то пойдет не так у одного направления работы системы, остальные не последовали бы за ним и могли бы продолжить работу до восстановления полноценной работы изначальной системы и лишь потом бы потребовалась перезагрузка. Кроме того, было необходимо продумать сценарии использования системы, подумать как сделать

интерфейс для пользователя удобней, в особенности для покупателя билетов.

10. Список литературы

1. Гоппа, В.Д. Введение в алгебраическую теорию информации / В.Д. Гоппа. - М.: ФИЗМАТЛИТ, 2021. - 112 с.
2. Дронов, В. JavaScript в Web-дизайне / В. Дронов. - М.: СПб: БХВ, 2020. - 880 с.
3. Кожевин, Н.В Аэровокзалы / Н.В Кожевин. - М.: ЁЁ Медиа, 2019. - 498 с.
4. Луни, К. Oracle 8i. Настольная книга администратора / К. Луни, М. Терьо. - М.: ЛОРИ, 2021. - 702 с.
5. Овчаров, Л.А. Автоматизированные банки данных / Л.А. Овчаров, С.Н. Селетков. - М.: Финансы и статистика, 2019. - 262 с.
6. Усов, В. Основы разработки приложений под iOS, iPadOS и macOS. 6-е изд. дополненное и переработанное. - СПб.: Питер, 2021. - 544 с.
7. Давыдов, Станислав IntelliJ IDEA. Профессиональное программирование на Java. Наиболее полное руководство / Станислав Давыдов , Алексей Ефимов. - М.: БХВ-Петербург, 2019. - 800 с.

8. Нотон Java. Справочное руководство. Все, что необходимо для программирования на Java / Нотон, Патрик. - М.: Бином, 2019. - 448 с.
9. Рассел, Р. Защита от хакеров коммерческого сайта / Р. Рассел. - М.: Книга по Требованию, 2020. - 548 с.
10. Сьерра, К. Изучаем Java / К. Сьерра. - М.: Эксмо, 2021. - 350 с.
11. Шаньгин, Владимир Федорович Информационная безопасность и защита информации / Шаньгин Владимир Федорович. - М.: ДМК Пресс, 2019. - 239 с.
12. Эванс, Б. Java. Новое поколение разработки / Б. Эванс. - М.: Питер, 2021. - 654 с.
13. Буралев, Ю. В. Безопасность жизнедеятельности на транспорте. Учебник / Ю.В. Буралев, Е.И. Павлова. - М.: Транспорт, 2021. - 200 с.
14. Сапронов, Ю. Г. Экспертиза и диагностика объектов и систем сервиса / Ю.Г. Сапронов. - М.: Academia, 2021. - 224 с.
15. Юркин, Ю.А. Аэродромы и аэропорты. Учебное пособие / Юркин Ю.А. - М.: МГТУ ГА, 2019 - 400 с.

Приложение. 1. Исходный код программы