Архитектура проекта RestaurantApp

Проект реализован с использованием Java 17, Spring Boot, Maven и базой данных PostgreSQL. Приложение предназначено для бронирования столиков в ресторане и имеет веб-интерфейс на Thymeleaf. В качестве системы миграции используется Liquibase. Безопасность реализована через Spring Security (in-memory). Документация пишется в формате AsciiDoc.

Общая архитектура

Приложение построено по принципам слоистой архитектуры:

- Контроллеры (Web Layer): обрабатывают входящие HTTP-запросы и возвращают HTML-страницы.
- Сервисный слой (Service Layer): содержит бизнес-логику.
- Репозитории (Data Access Layer): взаимодействуют с базой данных через Spring Data JPA.
- DTO: передают данные между слоями.
- Конфигурационные классы и утилиты.

Используемые технологии и зависимости

- Java 17
- Spring Boot
  - Spring Web
  - Spring Data JPA
  - Spring Security
  - Spring Mail
- Thymeleaf
- PostgreSQL
- Liquibase
- Maven
- Lombok
- SLF4J + Logback
- Asciidoctor (для генерации документации)

Структура пакетов

```
de.ait.restaurantapp
 ├──── config           // Конфигурации Spring и безопасности
 ├──── controllers      // Контроллеры
 ├──── dto              // Классы DTO
 ├──── enums            // Перечисления (Enums)
 ├──── exception        // Кастомные исключения и обработка ошибок
 ├──── models           // JPA-сущности
 ├──── repositories     // Интерфейсы репозиториев
 ├──── services         // Сервисы и интерфейсы
 ├──── utils            // Утилитные классы
 ├──── RestaurantApp    // Главный класс приложения
```

Конфигурация безопасности

Spring Security подключён и настроен через собственный класс `SecurityConfig`:

- В `application.properties` указано: `spring.autoconfigure.include=org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration`, что включает стандартную автоконфигурацию безопасности.
- Безопасность реализована через два отдельных фильтра (`SecurityFilterChain`):
  - Для всех путей `/restaurant/admin/**` доступ разрешён только пользователю с ролью `ADMIN`.
  - Для публичных страниц бронирования и меню (`/restaurant`, `/restaurant/menu/`, `/restaurant/reservations/` и др.) доступ открыт для всех пользователей.
- Аутентификация выполняется через базовую HTTP-аутентификацию (`httpBasic`).
- Создан пользователь "admin" с паролем "secret" в памяти (`InMemoryUserDetailsManager`).
- Защита от CSRF-атак (`CSRF Protection`) отключена на этапе разработки для упрощения работы с запросами.

Интеграция с БД и миграции

Компонент	Конфигурация и описание
Подключение к PostgreSQL	• **URL**: `spring.datasource.url=jdbc:postgresql://localhost:5432/restaurantdb` • **Пользователь**: `spring.datasource.username=postgres` • **Пароль**: `spring.datasource.password=1111` • **Драйвер**: `spring.datasource.driver-class-name=org.postgresql.Driver` • **Диалект Hibernate**: `spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect` • **Автоматическое DDL**: `spring.jpa.hibernate.ddl-auto=none`
Управление миграциями (Liquibase)	• **Включено**: `spring.liquibase.enabled=true` • **Master changelog**: `spring.liquibase.change-log=/db/changelog/db.changelog-master.xml` • **Логирование**: `logging.level.liquibase=DEBUG` • **Структура changelog**: ◦ `db.changelog-master.xml` включает: ▪ `/db/changelog/db.changelog-1.0.xml` — создание таблиц `restaurant_tables` и `reservations` с внешним ключом (CASCADE) ▪ `/db/changelog/db.changelog-1.1.xml` — изменение таблицы `reservations`: ▪ добавление столбца `reservation_code` NOT NULL ▪ переименование `guest_number` → `guest_count` ▪ добавление столбца `is_admin` с описанием отката

Дополнительные компоненты и паттерны

Компонент / Паттерн	Описание
DTO (Data Transfer Object)	Используются для передачи данных между слоями приложения: • `EmailDto` — содержит информацию для отправки email-уведомлений. • `ReservationFormDto` — данные из формы бронирования.

Компонент / Паттерн	Описание
Глобальный обработчик ошибок	Класс `GlobalExceptionHandler`: • перехватывает исключения (например, `NoAvailableTableException`); • возвращает пользователю понятные HTTP-ответы (например, 409 Conflict).
Генерация уникальных идентификаторов	Класс `ReservationIDGenerator`: • утилитный; • создаёт уникальные коды бронирования на основе UUID и timestamp.
Конфигурация логирования (Logback)	Файл `logback.xml` содержит: • вывод логов в консоль и файл; • шаблон форматирования; • ротацию и архивирование логов; • уровень логирования — `DEBUG`.
application.properties	Содержит параметры конфигурации: • подключение к PostgreSQL; • настройки Liquibase; • SMTP-конфигурация для email; • параметры загрузки файлов; • часы работы ресторана; • количество столиков; • подключение Spring Security.
Enum ReservationStatus	Перечисление `ReservationStatus`: • определяет статус бронирования (`CONFIRMED`, `CANCELED`); • помогает контролировать жизненный цикл брони.
Главный класс приложения	Класс `RestaurantApp`: • содержит точку входа в приложение; • запускает Spring Boot с помощью `@SpringBootApplication`.

Компонент / Паттерн	Описание
Maven-конфигурация	Файл `pom.xml`: • содержит зависимости (Spring Boot, Mail, Security, Liquibase, Lombok и др.); • включает плагины сборки (Asciidoctor, Liquibase, Maven Compiler); • задаёт версии Java и библиотек.
Логирование	Используются: • SLF4J и Logback; • уровень `DEBUG` для Liquibase и ошибок сервера; • вывод как в консоль, так и в файл.
Lombok / Builder	Используются аннотации Lombok: • `@Data`, `@AllArgsConstructor`, `@NoArgsConstructor`; • уменьшают объём шаблонного кода в DTO и моделях.