The **maven-wrapper.properties** file ensures consistent Maven versions across environments. It defines the wrapper's download URL and version.

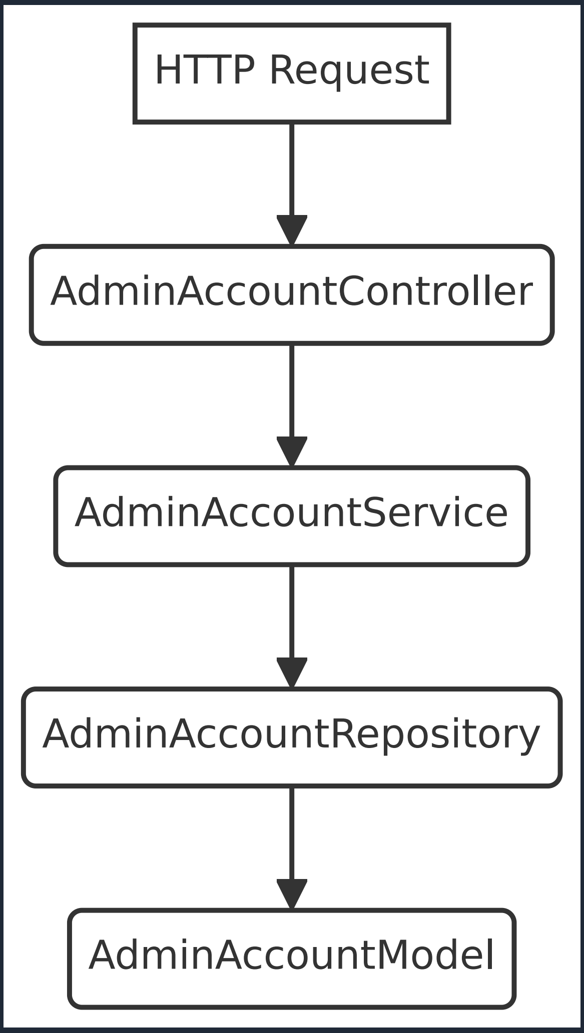
```
distributionUrl=https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/3.6.3/apache-maven-3.6.3-bin.zip
wrapperUrl=https://repo.maven.apache.org/maven2/io/takari/maven-wrapper/0.5.6/maven-wrapper-0.5.6.jar
```

- **distributionUrl**: Maven binary archive.
- **wrapperUrl**: Maven Wrapper JAR.

This package handles **administrator accounts**, including CRUD operations and persistence.

Exposes REST endpoints for admin account management.

- Annotated with `@RestController` and `@RequestMapping("/admin/accounts")`.
- Delegates business logic to **AdminAccountService**.
- Endpoints: create, retrieve, update, delete admin accounts.



JPA entity representing an admin account.

- Fields: `id`, `username`, `email`, `passwordHash`, `roles`.
- Annotated with `@Entity` and `@Table("admin_accounts")`.

Spring Data repository for **AdminAccountModel**.

- Extends `JpaRepository<AdminAccountModel, Long>`.
- Custom query: find by username or email.

Contains business rules for admin accounts.

- Validates uniqueness of email/username.
- Hashes passwords.
- Interacts with **AdminAccountRepository**.

This package mirrors admin functionality for **client accounts**.

REST controller under `/clients/accounts`.

- CRUD endpoints for clients.
- Uses **ClientAccountService**.

JPA entity for client accounts.

- Fields: `id`, `name`, `email`, `verified`, etc.

Repository interface extending `JpaRepository<ClientAccountModel, Long>`.

Business logic for clients.

- Handles registration, verification, updates.
- Integrates with **EmailVerificationService** for email confirmation.

Classes for **JWT-based security**, token issuance, validation, rotation, and cleanup.

Servlet filter to authenticate requests.

- Extracts JWT from cookie or header.
- Validates token via **JwtUtil** and **JwtKeyManager**.

Spring Security `Authentication` implementation.

- Stores user details and authorities.

Utility methods for setting and clearing JWT cookies.

Generates and parses JWTs.

- Signs tokens with keys from **JwtKeyManager**.
- Sets expiration and claims.

JPA entity for JWT signing keys.

- Fields: `id`, `kId`, `secret`, `createdAt`.

Repository for **JwtKeyModel**.

Creates new signing keys periodically.

Manages active key rotation and retrieval.

Entity representing refresh tokens.

- Fields: `id`, `token`, `status`, `expiresAt`, `userId`.

CRUD repository for refresh tokens.

Scheduled job to expire old refresh tokens.

Enum and JPA converter for token statuses: `ACTIVE`, `REVOKED`.

Components to **generate**, **persist**, and **validate** email codes.

Creates random verification codes.

Payload for sending verification email.

Entity storing code, user reference, expiry, role.

JPA repository for **EmailVerificationModel**.

Logic to send codes, check validity, and mark verified.

REST endpoints:

- `POST /auth/verify/email/request` → send code.
- `POST /auth/verify/email/confirm` → confirm code.

Removes expired/unconfirmed codes.

Shared logic for **login**, **logout**, and user details.

Endpoints under `/auth`:

- `POST /login`
- `POST /refresh-token`
- `POST /logout`

Request/response payloads: credentials and tokens.

Handles authentication:

- Validates credentials via **AuthUserDetailsService**.
- Issues JWT access and refresh tokens.

Implements `UserDetails` for Spring Security.

Loads users (admin or client) by username/email.

Manages **rentals booking** workflows.

Separate REST controllers for admin vs. client:

- Admin can approve/reject bookings.
- Client can create/cancel bookings.

Data transfer objects for booking operations.

Business logic for booking lifecycle:

- Conflict detection
- Status transitions

General endpoints:

- Query bookings by month/product or by user/email.

JPA entity:

- Fields: `id`, `userId`, list of `productIds`, `startDate`, `endDate`, `status`.

Spring Data repository.

Shared logic for booking validation and persistence.

Cron job to expire or mark old bookings.

- **BookingByMonthAndProductRequestDTO** / Response
- **BookingByUserEmailRequestDTO** / Response
- **BookingByRequestReferenceDTO**
- **BookingProductIdsConverter**
- **BookingStatusConverter** & **BookingStatus**

Spring Boot configuration classes to wire beans and tasks.

Class	Purpose
CorsConfig	CORS settings
SecurityConfig	HTTP security, filter chain
RestTemplateConfig	Defines `RestTemplate` bean
ProdDatabaseConfig / TestDatabaseConfig	Profile-based DB setup
JwtKeyRotationConfig	Schedule for key rotation
RefreshTokenCleanupTaskConfig	Binds **JwtRefreshTokenCleanupTask**
VerificationCodeCleanupTaskConfig	Binds **EmailVerificationCleanupTask**
BookingCleanupTaskConfig	Binds **BookingCleanupTask**
PaymentTransactionCleanupTaskConfig	Binds payment cleanup task
BackupTaskConfig	Backup scheduling
RentalGadgetStartupTaskConfig	Imports initial gadget data

REST endpoints for generating and retrieving rental contracts.

Wraps PDF generation or template merging logic.

Custom exceptions model various error scenarios; bodies format consistent JSON.

- **exception/**: definitions (e.g., `BookingConflictException`).
- **exception_body/**: maps exceptions to HTTP responses.
- **GlobalExceptionHandler.java**: `@ControllerAdvice` to translate exceptions to status codes.

Endpoint to retrieve unique IDs for various entities.

Generates sequences or UUIDs based on entity type.

Handles **offline cash** payments and webhooks.

- **CashPaymentController.java / WebhookController**: endpoints for initiating deposits and receiving status updates.
- **CashPaymentService**: business logic for validating and recording cash transactions.
- DTOs: **CashDepositDetailsDTO**, **CashPaymentDetailsDTO**, **CashTransactionRequestDTO**, **CashPaymentReponseDTO**

Integrates with external gateways via REST.

- **OnlinePaymentController.java / WebhookController**: endpoints for checkout and callback.
- **OnlinePaymentService**: creates payment requests, handles responses.
- Error handler: **OnlinePaymentResponseErrorHandler**
- DTOs: **OnlineCheckoutRequestDTO**, **OnlineCheckoutResponseDTO**, **OnlineFailedPaymentPayloadResponseDTO**, **OnlineSucessPaymentPayloadResponseDTO**, **OnlinePaymentDetailsDTO**
- Enum & Converter: **OnlinePaymentWebhookStatus**

Shared payment types and cleanup.

- **PaymentController.java**: general payment history endpoints.
- **PaymentService.java**: orchestrates cash and online flows.
- **PaymentItem.java**: line-item model.
- **PaymentTransactionModel.java**: JPA entity.
- **PaymentTransactionRepository.java**
- **PaymentTransactionCleanupTask.java**
- **PaymentStatus** & **PaymentStatusConverter**
- History DTOs: **PaymentTransactionHistoryRequestDTO**, **PaymentTransactionHistoryResponseDTO**

CRUD and initialization for rental gadgets.

Endpoints to list, add, update, delete gadgets.

Business rules: availability, status changes.

JPA entity: fields include images (via separate table), status.

Repository interface.

Imports initial gadget data on startup.

- **RentalGadgetDTO**: create/update payload.
- **RentalGadgetDetailsDTO**: detailed view.

- **RentalGadgetStatus** & **Converter**

Allows clients to review gadgets.

- **ReviewController.java**: endpoints for creating and viewing reviews.
- **ReviewService.java**: validates review rules.
- **ReviewModel.java** & **ReviewRepository.java**
- **ReviewDTO.java**

Code reused across modules.

- **BaseAccountModel.java**: common fields for account entities.
- **AccountDTO.java** & **AccountType** / **Converter**
- **AccountAccessDeniedHandler.java**: handles 403 errors.
- **EmailSenderService.java**: wraps SMTP or third-party email API.
- **Base64Util.java**: encoding helpers.
- **ErrorMessageBodyUtil.java**: formats validation errors.

Main Spring Boot bootstrap class.

Configure DB URL, ports, logging, profiles.

Basic unit and integration tests for email and application startup.

- **EmailSenderServiceTests.java**
- **EmailVerificationControllerTests.java**
- **EmailVerificationServiceTests.java**
- **BackendApplicationTests.java**

Defines container image: JDK, port exposure, JAR execution.

Maven Wrapper and project POM define dependencies, plugins, and build lifecycle.

{"title":"Note","content":"Each module follows a clear MVC pattern: Controller → Service → Repository → Model."}