High-level structure modules and their relationships



Here's a breakdown of the modules and their responsibilities:

1. **User Management**:
   * **User Controller**: Handles user registration and login requests.
   * **User Service**: Implements the business logic for user management, including registration, authentication, and profile updates.
   * **User Repository**: Responsible for storing and retrieving user data from the database.
2. **Policy Management**:
   * **Policy Controller**: Handles policy proposal submission, review, and status updates.
   * **Policy Service**: Implements the business logic for policy proposal management, including input validation and proposal status tracking.
   * **Policy Review Service**: Handles the review and approval process for policy proposals, including quote generation.
   * **Policy Repository**: Responsible for storing and retrieving policy-related data from the database.
3. **Payment Management**:
   * **Payment Controller**: Handles premium payment requests.
   * **Payment Service**: Implements the business logic for payment processing, including integration with a payment gateway.
4. **Policy Tracking**:
   * **Policy Tracking Controller**: Provides endpoints to retrieve policy details and status.
   * **Policy Tracking Service**: Implements the logic for policy status tracking and premium reminder emails.
   * **Reminder Service**: Handles the scheduling and sending of premium reminder emails.
5. **Reporting and Analytics**:
   * **Report Controller**: Provides endpoints to fetch policy-related statistics and reports.
   * **Report Service**: Implements the logic for data aggregation and analysis to generate policy reports.
6. **Security and Authentication**:
   * **Security Config**: Configures Spring Security and handles JWT-based authentication.
   * **JWT Token Provider**: Manages the generation, verification, and user details extraction for JWT tokens.
7. **Error Handling and Logging**:
   * **Global Exception Handler**: Handles and logs exceptions across the application.
   * **Logging Configuration**: Sets up the logging framework and specifies the logging levels and destinations.
8. **Configuration and Utilities**:
   * **Application Properties**: Stores configurable values, such as database credentials and payment gateway details.
   * **Utility Classes**: Provide common functionality, such as input validation, email sending, and date/time operations.

Springboot Project Architecture



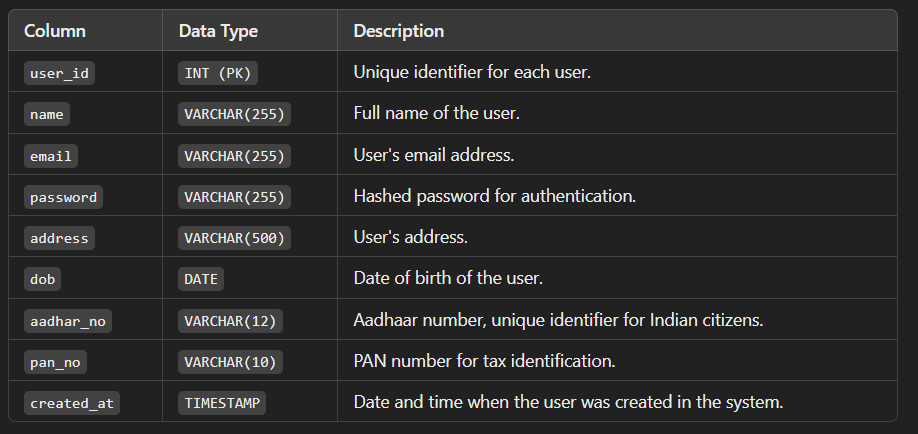
Breakdown of the directory structure and the responsibilities of each component:

1. **com.example.autoinsurance**:
   * AutoInsuranceApplication.java: The main application class that bootstraps the Spring Boot application.
2. **config**:
   * SecurityConfig.java: Configures Spring Security and handles JWT-based authentication.
   * WebConfig.java: Configures web-related settings, such as CORS and request/response handling.
3. **controller**:
   * AuthController.java: Handles user authentication, including registration and login.
   * PaymentController.java: Handles premium payment requests.
   * PolicyController.java: Handles policy management, including proposal submission and review.
   * ReminderController.java: Handles premium reminder email sending.
   * ReportController.java: Handles reporting and analytics-related requests.
4. **entity**:
   * Policy.java: Represents the policy entity.
   * User.java: Represents the user entity.
   * Payment.java: Represents the payment entity.
5. **exception**:
   * GlobalExceptionHandler.java: Handles and logs exceptions across the application.
   * PolicyNotFoundException.java: Custom exception for policy-related errors.
   * UserNotFoundException.java: Custom exception for user-related errors.
6. **repository**:
   * PolicyRepository.java: Provides data access methods for the policy entity.
   * UserRepository.java: Provides data access methods for the user entity.
   * PaymentRepository.java: Provides data access methods for the payment entity.
7. **service**:
   * AuthService.java: Implements the business logic for user authentication.
   * PaymentService.java: Implements the business logic for premium payment processing.
   * PolicyReviewService.java: Implements the business logic for policy review and approval.
   * PolicyService.java: Implements the business logic for policy management.
   * ReminderService.java: Implements the business logic for premium reminder email sending.
   * ReportService.java: Implements the business logic for reporting and analytics.
8. **util**:
   * JwtTokenProvider.java: Provides functionality for generating and verifying JWT tokens.
   * ValidationUtils.java: Provides utility methods for input validation.
9. **resources**:
   * application.properties: Stores the application's configuration, such as database credentials and payment gateway details.
   * logback.xml: Configures the logging framework for the application.

**Detailed ER Diagram Explanation**

**1. USER Table**

This table stores personal information about users who own vehicles and are eligible for insurance policies.



**2. VEHICLE Table**

This table keeps information about each vehicle that users own and insure.

A screenshot of a computer

Description automatically generated

**3. PROPOSAL Table**

This table tracks insurance proposals submitted by users. Each proposal links a user, vehicle, policy, and reviewing officer.

A screenshot of a computer

Description automatically generated

**4. OFFICER Table**

This table stores information about officers responsible for reviewing and approving proposals.

A screenshot of a computer

Description automatically generated

**5. POLICY Table**

This table contains details about the types of insurance policies available for vehicles.

A screenshot of a computer

Description automatically generated

**6. PAYMENT Table**

This table records payment transactions for proposals.

A screenshot of a computer

Description automatically generated

Summary of Relationships

* **USER** owns **VEHICLEs** and submits **PROPOSALs**.
* **VEHICLE** is insured through **PROPOSALs**.
* **PROPOSAL** is submitted by **USERs**, reviewed by **OFFICERs**, and based on **POLICY**.
* **POLICY** is associated with **PROPOSALs**.
* **PAYMENT** is generated for **PROPOSALs** after approval.

**Feature of our Database**

This design offers **structured policy management** through centralized policy data, **secure and transparent proposal tracking** by linking each proposal to users, vehicles, and officers, **efficient payment tracking** with clear links to each proposal for easy financial auditing, and **data consistency.**