

# SMART-JOB

## 1. Overview.

The **Smart-Job Application** is a full-stack web platform that allows job seekers and employers to interact through job listings, applications, interviews, and messaging. The system is designed to be scalable, modular, and secure using industry-standard design patterns and Spring Boot architecture principles.

The platform supports:

- User registration and login with role-based access.
- Job listing and advanced search.
- Real-time notifications and email alerts.
- Applying for available jobs.

## 2. Design Patterns Used

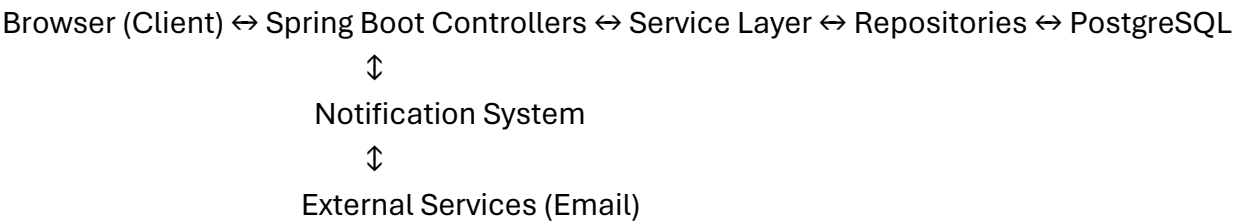
Below is a detailed list of design patterns applied in the application and their specific use cases:

Pattern	Applied In	Purpose / Problem Solved
<b>Factory</b>	Auth Service	Dynamically instantiate different types of users (Admin, Recruiter, Seeker) at runtime.
<b>Strategy</b>	Auth, Search, Job, Admin	Enable interchangeable login strategies, filtering algorithms, and report generation.
<b>Chain of Responsibility</b>	Auth Service	Break login into modular steps: validation, authentication, role-checking.
<b>Singleton</b>	JWT Utility, Security	Maintain one instance of stateless services (e.g., token utility) throughout the application.

<b>Observer</b>	Job Service, Notification	Decouple event publishing (job posted) from consumers (notifications, email).
<b>Repository</b>	All services (Persistence)	Centralize and abstract access to database using Spring Data JPA.
<b>Adapter</b>	Email Service	Wrap third-party email APIs like SendGrid into app-specific interfaces.
<b>State</b>	Application Service	Handle application lifecycle: Applied → In Review → Shortlisted → Hired/Rejected.

### 3. System Architecture

#### 3.1 High-Level Architecture



#### 3.2 Components

Component	Description
<b>Frontend</b>	Thymeleaf templates rendered by Spring Boot; provides the UI layer.
<b>Backend</b>	RESTful controllers + business services written in Java/Spring Boot.
<b>Authentication</b>	Spring Security + JWT for token-based stateless auth.
<b>Database</b>	PostgreSQL; relational schema for all core entities.
<b>Notification System</b>	Event-driven alert system for in-app and email alerts.
<b>Email Integration</b>	External email API (SendGrid or SMTP) for transactional messages.

## 3.3 Data Flow

### *User Registration & Login*

1. User submits credentials (POST /auth/login)
2. AuthService verifies identity using Chain of Responsibility.
3. If valid, JWT is generated and returned.
4. JWT is included in subsequent requests (as Proxy guards).

### *Job Posting*

1. Recruiter submits job post via UI.
2. JobService persists it via Repository.
3. Observer pattern triggers NotificationService.
4. EmailService (Adapter) sends external alerts.

### *Search Jobs*

1. User applies filters (location, title, type).
2. Strategy pattern executes filtering algorithm.
3. Results are returned and displayed.

### *Apply to Job*

1. ApplicationService creates application.
2. State Pattern manages lifecycle: Applied → Hired.
3. NotificationService alerts recruiter.

## 4. Component/Service Breakdown

### **Auth Service**

- **Purpose:** Manage registration, authentication, and authorization.
- **Design Patterns:**
  - **Factory:** Create different user types.
  - **Strategy:** Allow OAuth or traditional login in future.
  - **Chain of Responsibility:** Break down login process.
  - **Singleton:** Token service maintains one instance across app.

## Job Service

- **Purpose:** Manage CRUD for job posts, implement search and filtering.
- **Design Patterns:**
  - **Observer:** Notify users when new jobs are posted.
  - **Strategy:** Filter jobs using interchangeable filter strategies.

## Notification Service

- **Purpose:** Real-time or asynchronous alerts on system events.
- **Design Patterns:**
  - **Observer:** Subscribed to Job, Interview, Application events.
  - **Repository:** Persist notification history.

## Email Service

- **Purpose:** Send confirmation and notification emails.
- **Design Patterns:**
  - **Adapter:** Integrate external services (SMTP/SendGrid).
  - **Observer:** Triggered via NotificationService. **Search & Filter Module**
- **Purpose:** Provide flexible job search experience.
- **Design Pattern:**
  - **Strategy:** Filter by category, location, experience, etc.

## Application Service

- **Purpose:** Allow job seekers to apply and track applications.
- **Design Pattern:**
  - **State Pattern:** Manages states such as Applied, Rejected, Hired.

## 5. Technology Stack

Layer	Technology / Tool
Backend	Java, Spring Boot, Spring Security
Frontend	Thymeleaf, HTML, CSS, Bootstrap
Database	PostgreSQL

<b>ORM</b>	Spring Data JPA, Hibernate
<b>Authentication</b>	JWT, Spring Security
<b>Notifications</b>	In-App Alerts
<b>Email</b>	JavaMailSender, SendGrid API
<b>Logging</b>	SLF4J + Logback
<b>Build Tool</b>	Maven or Gradle