perplexity

# Top Open-Source Java + React Project Folder Structure Best Practices

Based on industry standards and practices from leading open-source projects, here are proven folder structure approaches and naming conventions for Java + React full-stack applications.

## Recommended Project Structure

### Maven/Gradle-Based Structure

```
hiring-platform/
├── backend/
│   ├── src/
│   │   ├── main/
│   │   │   ├── java/
│   │   │   │   └── com/company/platform/
│   │   │   │       ├── controller/
│   │   │   │       ├── service/
│   │   │   │       ├── repository/
│   │   │   │       ├── model/
│   │   │   │       ├── dto/
│   │   │   │       ├── config/
│   │   │   │       ├── security/
│   │   │   │       ├── exception/
│   │   │   │       └── util/
│   │   │   └── resources/
│   │   │       ├── application.properties
│   │   │       ├── static/
│   │   │       └── db/migration/
│   │   └── test/
│   │       ├── java/
│   │       └── resources/
│   ├── pom.xml
│   └── Dockerfile
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   ├── hooks/
│   │   ├── services/
│   │   ├── utils/
│   │   ├── contexts/
│   │   ├── assets/
│   │   ├── config/
│   │   └── styles/
│   ├── public/
│   ├── package.json
│   └── Dockerfile
│
```

```
├── devops/
│   ├── docker/
│   ├── kubernetes/
│   ├── terraform/
│   └── ci-cd/
├── shared/
│   ├── api-contracts/
│   └── common-types/
└── docker-compose.yml
```

## Backend (Java/Spring Boot) Naming Conventions

### Package Naming

Following Oracle Java conventions and Spring Boot best practices[1] [2]:

- Use **reverse domain notation**: `com.company.platform`
- All **lowercase letters**
- Use **dots (.)** as separators
- Be **descriptive and meaningful**

### Core Package Structure

Based on proven layered architecture patterns[1] [2]:

**controller/** - REST API endpoints

```
// Examples:
UserController.java
JobApplicationController.java
AuthController.java
```

**service/** - Business logic layer

```
// Examples:
UserService.java
JobApplicationService.java
EmailService.java
```

**repository/** - Data access layer

```
// Examples:
UserRepository.java
JobApplicationRepository.java
CompanyRepository.java
```

**model/** - Entity classes

```
// Examples:
User.java
JobApplication.java
Company.java
```

**dto/** - Data Transfer Objects

```
// Examples:
UserDTO.java
CreateJobApplicationRequest.java
LoginResponse.java
```

**config/** - Configuration classes

```
// Examples:
SecurityConfig.java
DatabaseConfig.java
SwaggerConfig.java
```

**exception/** - Custom exceptions and handlers

```
// Examples:
GlobalExceptionHandler.java
ResourceNotFoundException.java
ValidationException.java
```

**security/** - Authentication and authorization

```
// Examples:
JwtTokenUtil.java
JwtAuthenticationFilter.java
SecurityService.java
```

**util/** - Utility classes

```
// Examples:
DateUtils.java
FileUtils.java
ValidationUtils.java
```

## Configuration Files Structure

```
src/main/resources/
├── application.properties          # Main configuration
├── application-dev.properties      # Development environment
├── application-prod.properties     # Production environment
├── static/                         # Static web assets
```

```
└── db/migration/                   # Database migration scripts
    ├── V1__Create_user_table.sql
    └── V2__Add_job_application_table.sql
```

## Frontend (React) Naming Conventions

### Core Folder Structure

Following React community best practices[3] [4] [5]:

**components/** - Reusable UI components

```
components/
├── common/                         # Shared components
│   ├── Button/
│   │   ├── Button.tsx
│   │   ├── Button.module.css
│   │   └── index.ts
│   └── Modal/
└── features/                       # Feature-specific components
    ├── JobListing/
    └── UserProfile/
```

**pages/** - Page-level components

```
pages/
├── HomePage/
│   ├── HomePage.tsx
│   ├── HomePage.module.css
│   └── index.ts
├── JobsPage/
└── ProfilePage/
```

**hooks/** - Custom React hooks

```
hooks/
├── useAuth.ts
├── useJobApplications.ts
└── useLocalStorage.ts
```

**services/** - API integration

```
services/
├── api/
│   ├── userService.ts
│   ├── jobService.ts
│   └── authService.ts
├── http/
```

```
│   └── httpClient.ts
└── websocket/
```

**contexts/** - React Context providers

```
contexts/
├── AuthContext.tsx
├── ThemeContext.tsx
└── UserContext.tsx
```

**utils/** - Utility functions

```
utils/
├── dateUtils.ts
├── validationUtils.ts
└── formatters.ts
```

**assets/** - Static resources

```
assets/
├── images/
├── icons/
├── fonts/
└── videos/
```

**config/** - Configuration files

```
config/
├── environment.ts
├── constants.ts
└── apiEndpoints.ts
```

**styles/** - Global styles and themes

```
styles/
├── globals.css
├── themes/
├── variables.css
└── mixins.css
```

## React Naming Conventions

Based on community standards[6] [7]:

- **Components**: Use **PascalCase** (`UserProfile.tsx`)
- **Files**: Use **PascalCase** for components, **camelCase** for utilities
- **Folders**: Use **camelCase** (`userProfile/` or `user-profile/`)

- **Hooks**: Start with **"use"** (`useAuth.ts`)
- **Context**: End with **"Context"** (`AuthContext.tsx`)
- **Services**: End with **"Service"** (`userService.ts`)

## DevOps and Infrastructure Naming

### DevOps Folder Structure

```
devops/
├── docker/
│   ├── backend/
│   │   └── Dockerfile
│   ├── frontend/
│   │   └── Dockerfile
│   └── database/
├── kubernetes/
│   ├── backend-deployment.yaml
│   ├── frontend-deployment.yaml
│   ├── ingress.yaml
│   └── configmaps/
├── terraform/
│   ├── environments/
│   │   ├── dev/
│   │   ├── staging/
│   │   └── prod/
│   ├── modules/
│   └── variables.tf
└── ci-cd/
    ├── jenkins/
    ├── github-actions/
    │   ├── build.yml
    │   ├── deploy.yml
    │   └── test.yml
    └── scripts/
        ├── build.sh
        ├── deploy.sh
        └── test.sh
```

### DevOps Naming Conventions

- **Environment-based naming**: `app-name-environment` (e.g., `hiring-platform-dev`)
- **Kubernetes resources**: Use **kebab-case** (`backend-service.yaml`)
- **Docker images**: `organization/app-name:tag`
- **Scripts**: Use **descriptive verbs** (`deploy-backend.sh`)

## API and Service Conventions

### REST API Naming

```
@RestController
@RequestMapping("/api/v1")
public class JobController {
    @GetMapping("/jobs")              // GET /api/v1/jobs
    @PostMapping("/jobs")             // POST /api/v1/jobs
    @GetMapping("/jobs/{id}")         // GET /api/v1/jobs/123
    @PutMapping("/jobs/{id}")         // PUT /api/v1/jobs/123
    @DeleteMapping("/jobs/{id}")      // DELETE /api/v1/jobs/123
}
```

### Database Naming

- **Tables**: Use **snake_case** (`job_applications`, `user_profiles`)
- **Columns**: Use **snake_case** (`created_at`, `user_id`)
- **Foreign keys**: `table_name_id` (`user_id`, `company_id`)
- **Indexes**: `idx_table_column` (`idx_users_email`)

## Integration Best Practices

### Shared API Contracts

```
shared/
├── api-contracts/
│   ├── openapi/
│   │   ├── user-api.yaml
│   │   └── job-api.yaml
│   └── generated/
│       ├── java/
│       └── typescript/
└── common-types/
    ├── enums/
    └── constants/
```

### Environment Configuration

```
config/
├── environments/
│   ├── development.yml
│   ├── staging.yml
│   └── production.yml
└── secrets/
    ├── dev-secrets.yml
    └── prod-secrets.yml
```

This structure follows **proven enterprise patterns** from successful open-source projects, ensuring maintainability, scalability, and team collaboration efficiency. The separation between frontend and backend allows for **independent development and deployment**, while shared folders enable **code reuse and consistency** across the entire stack [1] [2] [7].

⁂

1. https://symflower.com/en/company/blog/2024/spring-boot-folder-structure/
2. https://dev.to/imajenasyon/folder-structure-backend-java-2402
3. https://www.robinwieruch.de/react-folder-structure/
4. https://dev.to/itswillt/folder-structures-in-react-projects-3dp8
5. https://blog.webdevsimplified.com/2022-07/react-folder-structure/
6. https://dev.to/vishesh-tiwari/javascript-clean-code-series-8ci
7. https://www.geeksforgeeks.org/reactjs/folder-structure-for-a-react-js-project/