



CV Scanner

Unified Coding Standards Document

Version: 3.0.0
Date: August 20th, 2025
Status: Final
Document Type: Coding Standards
Project: CV Scanner - Automated CV Analysis System

Team Members

Marcelo Parsotam
Unaisah Hassim
Talhah Karodia
Abdullah Pochee
Ronan Smart

Table of Contents

Unified Coding Standards Document.....	2
AI Module - Python (FastAPI)	2
1. Project Overview	2
2. File Structure	2
3. Coding Conventions.....	2
4. Naming	2
5. Tools.....	3
6. Design Practices	3
API Module - Spring Boot.....	3
1. Structure.....	3
2. Naming & Style	3
3. Formatting	3
4. REST API & Error Handling.....	3
5. Security	4
UI Module - React + TypeScript	4
1. File Structure	4
2. Naming & Component Style.....	4
3. Component Layout.....	4
4. Styling & State	4
5. API & Error Handling	5
6. Accessibility	5
Key Rules Summary	5

Unified Coding Standards Document

This document unifies the coding standards for all major components of the CV-Scanner Project:

- AI Module (FastAPI + Python)
- API Module (Java Spring Boot)
- UI Module (React + TypeScript)

AI Module - Python (FastAPI)

1. Project Overview

The AI module processes CVs using BART zero-shot classification with dynamically configurable categories.

2. File Structure

CV-Scanner/AI

```
|— app.py
|— bart_model.py
|— config_store.py
|— worker.py
|— ai_tags.py
|— requirements.txt
```

3. Coding Conventions

- Language: Python 3.9+
- Style: PEP 8, 4-space indentation
- Commenting: Inline (#) and docstrings
- Grouped imports: Standard → Third-party → Local

4. Naming

- Variables: snake_case (pdf_bytes)
- Functions: snake_case (process_pdf_file)
- Constants: UPPER_CASE (LABELS)
- Classes: PascalCase (CvParser)
- Files: snake_case (app.py)

5. Tools

- black - Auto formatting
- flake8 - Linting
- isort - Import ordering
- mypy - Type checking

6. Design Practices

- Modular functions with single responsibility
- Async FastAPI endpoints for I/O operations
- Clear error messages via HTTPException
- Configuration management through JSON files
- Hot-reload capability for categories

API Module - Spring Boot

1. Structure

CV-Scanner\API\api\src\main\java\com\example\api

```
|— ApiApplication.java
|— AuthController.java
|— CVController.java
|— application.properties
```

2. Naming & Style

- Classes: PascalCase (CVController)
- Methods: camelCase (extractTextFromPdf)
- Variables: camelCase (passwordEncoder)
- Constants: UPPER_SNAKE_CASE (SUPPORTED_TYPES)

3. Formatting

- Indentation: 4 spaces
- Braces: same-line
- Max Line: 100-120 chars
- Annotate classes/methods with Javadoc

4. REST API & Error Handling

- Proper HTTP codes (400, 404, 500)
- JSON response with consistent structure
- Use SLF4J for logging
- Input validation for all endpoints

5. Security

- Password hashing with BCrypt
- CORS configuration for cross-origin requests
- Role-based access control
- SQL injection prevention with parameterized queries

UI Module - React + TypeScript

1. File Structure

CV-Scanner\UI\CV-Scanner\src\pages

```
|— App.tsx
|— pages/
|   |— UploadCVPage.tsx
|   |— UserManagementPage.tsx
|   |— Search.tsx
|   |— Settings.tsx
|   |— Help.tsx
|   |— LandingPage.tsx
|   |— Login.tsx
|   |— AddUserPage.tsx
|   |— CandidatesPage.tsx
|— assets/
```

2. Naming & Component Style

- Components: PascalCase (UploadCVPage)
- Variables: camelCase (handleFileChange)
- Types/Interfaces: PascalCase (UserData)
- Files: PascalCase (UserManagementPage.tsx)

3. Component Layout

1. useState hooks
2. useEffect hooks
3. Event handlers
4. JSX return with consistent styling

4. Styling & State

- Use MUI sx prop for styling
- Consistent color scheme throughout application
- Responsive design for mobile/desktop
- Global state management with React Context

5. API & Error Handling

- Use async/await with fetch API
- Error boundaries for component-level errors
- Loading states for async operations
- User-friendly error messages

6. Accessibility

- Semantic HTML elements
- Proper ARIA labels
- Keyboard navigation support
- Screen reader compatibility

Key Rules Summary

- Use clear naming conventions across all modules
- Write modular, documented, and testable code
- Validate inputs and handle errors gracefully
- Maintain consistent REST API structure and styles
- Follow accessibility best practices in UI components
- Implement proper security measures for authentication and authorization