Developer Manual - Insight Coding (Artifact Comparator)

A comprehensive guide for developers contributing to the Artifact Comparator project.

Table of Contents

---

1. Project Overview

Description

Artifact Comparator is a full-stack web application designed for software engineering research. It enables researchers to conduct empirical studies by allowing participants to evaluate and compare software artifacts (source code, test cases, UML diagrams, requirements documents) through structured comparison tasks.

Tech Stack

Layer	Technology
Frontend	React 18, Material-UI (MUI) 7, Axios
Backend	Spring Boot 3.2, Java 17, Spring Security, JPA
Database	PostgreSQL 15
Auth	JWT (JSON Web Tokens)
Build	Maven (Backend), npm (Frontend)
Container	Docker, Docker Compose
Migration	Flyway

User Roles

- **Admin** - System administration, user management
- **Researcher** - Create studies, manage artifacts, analyze results
- **Participant** - Complete evaluations, take quizzes
- **Reviewer** - Review participant submissions

---

2. Getting Started

Prerequisites

Ensure you have the following installed:

- **Java 17** (JDK)
- **Node.js 18+** and npm
- **Maven 3.9+**
- **Docker** and **Docker Compose**
- **PostgreSQL 15** (or use Docker)
- **Git**

Quick Start with Docker (Recommended)

Clone the repository
Bash
git clone <repository-url>
cd S2-T9

1.

Start all services
Bash
docker-compose up --build

2. **Access the application**
    - Frontend: http://localhost:3000
    - Backend API: http://localhost:8080
    - PostgreSQL: localhost:5433

Manual Setup (Development)

Database Setup
Install PostgreSQL or use Docker:
Bash
docker run -d --name artifact-db \
  -e POSTGRES_DB=artifact_comparator \
  -e POSTGRES_USER=admin \
  -e POSTGRES_PASSWORD=admin123 \
  -p 5433:5432 \
  postgres:15-alpine

1.

Backend Setup

Bash
cd backend
mvn clean install
mvn spring-boot:run

The backend will start on http://localhost:8080

Frontend Setup

Bash
cd frontend
npm install
npm start

The frontend will start on http://localhost:3000

---

3. Project Structure

Plaintext

```
S2-T9/
├── backend/                # Spring Boot backend
│   ├── src/
│   │   └── main/
│   │       ├── java/com/artifactcomparator/
│   │       │   ├── ArtifactComparatorApplication.java  # Main entry point
│   │       │   ├── common/         # Shared utilities
│   │       │   ├── config/         # Configuration classes
│   │       │   ├── controller/     # REST API endpoints
│   │       │   ├── dto/            # Data Transfer Objects
│   │       │   ├── export/         # Export functionality (PDF, Excel)
│   │       │   ├── model/          # JPA entities
│   │       │   ├── repository/     # Spring Data repositories
│   │       │   ├── security/       # JWT & Spring Security
│   │       │   └── service/        # Business logic
│   │       └── resources/
│   │           ├── application.properties  # App configuration
│   │           ├── db/migration/          # Flyway migrations
│   │           └── templates/             # JSON templates
│   ├── Dockerfile
│   └── pom.xml             # Maven dependencies
│
├── frontend/               # React frontend
│   ├── src/
│   │   ├── App.js          # Main router & app setup
│   │   ├── index.js        # Entry point
│   │   ├── components/     # Reusable UI components
│   │   ├── context/        # React Context (AuthContext)
│   │   ├── pages/          # Page components
```

```
│   │   │   ├── admin/          # Admin pages
│   │   │   ├── participant/    # Participant pages
│   │   │   ├── researcher/     # Researcher pages
│   │   │   ├── reviewer/       # Reviewer pages
│   │   │   └── shared/         # Shared pages
│   │   ├── services/           # API service modules
│   │   ├── shared-theme/       # MUI theme customization
│   │   └── styles/             # CSS stylesheets
│   ├── public/                 # Static assets
│   ├── Dockerfile
│   └── package.json            # npm dependencies
│
├── deliverables/               # Project documentation
├── scripts/                    # Utility scripts
├── docker-compose.yml          # Production Docker setup
├── docker-compose.dev.yml      # Development Docker overrides
└── README.md                   # Project readme
```

---

4. Backend Development

Architecture Overview

The backend follows a **layered architecture**:

```
Controller → Service → Repository → Database
    ↓          ↓
   DTO       Model
```

Key Packages

Controllers (controller/)

REST API endpoints. Each controller handles a specific domain.

Controller	Responsibility
AuthController	Authentication (login, register)
UserController	User management
StudyController	Study CRUD operations

ArtifactController	Artifact upload/management
QuestionnaireController	Questionnaire management
NotificationController	User notifications
evaluation/	Evaluation task endpoints

Models (model/)
JPA entities representing database tables.

Entity	**Description**
User	User accounts with roles
Study	Research studies
Artifact	Uploaded artifacts
Questionnaire	Evaluation questionnaires
QuizSubmission	Participant quiz responses
Notification	System notifications

Services (service/)
Business logic layer.

Service	Responsibility
UserService	User operations
StudyService	Study lifecycle management
ArtifactService	File storage & artifact management
GeminiApiService	AI integration (Google Gemini)
AIQuestionGenerationService	AI-powered question generation
NotificationService	Notification handling

Creating a New Endpoint

Create/update the Model (if needed):

Java

```
// model/Example.java
@Entity
@Table(name = "examples")
@Data
public class Example {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
}
```

1.

Create the Repository:

Java

```
// repository/ExampleRepository.java
public interface ExampleRepository extends JpaRepository<Example, Long> {
    List<Example> findByName(String name);
}
```

2.

Create the Service:
Java
```
// service/ExampleService.java
@Service
@RequiredArgsConstructor
public class ExampleService {
    private final ExampleRepository exampleRepository;

    public Example create(Example example) {
        return exampleRepository.save(example);
    }
}
```
3.

Create the Controller:
Java
```
// controller/ExampleController.java
@RestController
@RequestMapping("/api/examples")
@RequiredArgsConstructor
public class ExampleController {
    private final ExampleService exampleService;

    @PostMapping
    public ResponseEntity<Example> create(@RequestBody Example example) {
        return ResponseEntity.ok(exampleService.create(example));
    }
}
```
4.

Configuration

Key configuration files:

- application.properties - Main configuration
- config/SecurityConfig.java - Spring Security setup
- config/CorsConfig.java - CORS settings

Environment Variables

Variable	Default	Description

SPRING_DATASOURCE_URL	jdbc:postgresql://localhost:5433/artifact_comparator	Database URL
SPRING_DATASOURCE_USERNAME	admin	Database username
SPRING_DATASOURCE_PASSWORD	admin123	Database password
APP_ARTIFACTS_STORAGE_DIR	/app/storage/artifacts	File storage path

---

5. Frontend Development

Architecture Overview

The frontend uses **React** with **functional components** and **hooks**.

Plaintext
App.js (Router)
└── Pages
└── Components
└── Services (API calls)

Key Directories

Directory	**Purpose**
pages/	Page-level components (routes)
components/	Reusable UI components
services/	API service modules (Axios)

context/	React Context providers
shared-theme/	Material-UI theme configuration
styles/	Global CSS styles

Creating a New Page

Create the page component:
JavaScript
```
// pages/researcher/NewPage.jsx
import React, { useState, useEffect } from 'react';
import { Box, Typography } from '@mui/material';

const NewPage = () => {
  const [data, setData] = useState([]);

  useEffect(() => {
    // Fetch data on mount
  }, []);

  return (
    <Box sx={{ p: 3 }}>
      <Typography variant="h4">New Page</Typography>
    </Box>
  );
};

export default NewPage;
```
1.

Add the route in App.js:
JavaScript
```
import NewPage from './pages/researcher/NewPage';

// Inside Routes
<Route path="/researcher/new-page" element={
  <PrivateRoute allowedRoles={['RESEARCHER']}>
    <NewPage />
  </PrivateRoute>
} />
```
2.

API Services

API calls are centralized in services/api.js:

JavaScript
```
// services/api.js
export const studyService = {
    createStudy: (data) => api.post('/api/studies', data),
    getStudyById: (id) => api.get(`/api/studies/${id}`),
    // ...
};
```

Usage in components:

JavaScript
```
import { studyService } from '../services/api';

const fetchStudy = async (id) => {
    try {
        const response = await studyService.getStudyById(id);
        setStudy(response.data);
    } catch (error) {
        console.error('Failed to fetch study:', error);
    }
};
```

Authentication Context

The AuthContext manages user authentication state:

JavaScript
```
import { useAuth } from '../context/AuthContext';

const MyComponent = () => {
    const { user, login, logout, loading } = useAuth();

    if (loading) return <div>Loading...</div>;
    if (!user) return <Navigate to="/login" />;

    return <div>Welcome, {user.name}</div>;
};
```

Styling Guidelines

- Use **Material-UI (MUI)** components
- Use **sx prop** for component-specific styles
- Global styles go in styles/ directory
- Theme customization in shared-theme/

JavaScript

```
<Box sx={{
  p: 2,
  display: 'flex',
  gap: 2,
  backgroundColor: 'background.paper'
}}>
```

---

6. Database Management

Flyway Migrations

Database schema is managed with Flyway. Migration files are in:

backend/src/main/resources/db/migration/

Naming Convention

V{version}__{description}.sql

Examples:

- V1__create_users_table.sql
- V2__create_questionnaire_tables.sql
- V25__create_study_quizzes.sql

Creating a New Migration

Create a new SQL file with the next version number:

```
SQL
-- V26__add_new_feature.sql
CREATE TABLE new_feature (
  id BIGSERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

1.
2. Restart the backend - Flyway will auto-apply migrations.

Migration Best Practices

- **Never modify** existing migration files
- Always increment version numbers
- Test migrations locally before committing
- Include rollback strategy in comments if needed

Database Connection

```
Properties
# application.properties
spring.datasource.url=jdbc:postgresql://localhost:5433/artifact_comparator
spring.datasource.username=admin
```

spring.datasource.password=admin123

---

7. API Reference

Base URL

http://localhost:8080/api

Authentication

All endpoints (except /auth/*) require JWT Bearer token:

Authorization: Bearer <token>

Core Endpoints

Authentication

Method	Endpoint	Description
POST	/auth/register	Register new user
POST	/auth/login	Login, returns JWT

Users

Method	Endpoint	Description
GET	/users/me	Get current user
GET	/users	Get all users (Admin)
PUT	/users/{id}	Update user
DELETE	/users/{id}	Delete user

Studies

Method	Endpoint	Description
GET	/studies	Get all studies
GET	/studies/{id}	Get study by ID
POST	/studies	Create study
PUT	/studies/{id}	Update study
DELETE	/studies/{id}	Delete study
GET	/studies/my-studies	Get researcher's studies

Artifacts

Method	Endpoint	Description
POST	/artifacts/upload	Upload artifact
GET	/artifacts/{id}	Get artifact
GET	/artifacts/study/{studyId}	Get study artifacts
DELETE	/artifacts/{id}	Delete artifact

Evaluation Tasks

Method	Endpoint	Description

GET	/evaluation-tasks/study/{studyId}	Get tasks for study
POST	/evaluation-tasks	Create evaluation task
PUT	/evaluation-tasks/{id}	Update task

---

8. Testing

Backend Testing

Run tests with Maven:

Bash
cd backend
mvn test

Test Structure

backend/src/test/java/com/artifactcomparator/
├── controller/ # Controller tests
├── service/ # Service tests
└── repository/ # Repository tests

Writing Tests

Java
@SpringBootTest
@AutoConfigureMockMvc
class UserControllerTest {

```
    @Autowired
    private MockMvc mockMvc;

    @Test
    void shouldReturnCurrentUser() throws Exception {
        mockMvc.perform(get("/api/users/me")
                .header("Authorization", "Bearer " + validToken))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.email").exists());
    }
}
```

Frontend Testing

Run tests with npm:

Bash
cd frontend
npm test

---

9. Docker & Deployment

Docker Compose (Production)

Bash
Build and start all services
docker-compose up --build

Run in background
docker-compose up -d

Stop services
docker-compose down

View logs
docker-compose logs -f

Docker Compose (Development)

For hot-reloading during development:

Bash
docker-compose -f docker-compose.yml -f docker-compose.dev.yml up

Services

Service	Port	Description
postgres	5433	PostgreSQL database
backend	8080	Spring Boot API
frontend	3000	React development server

Building Images Individually

Backend:

Bash
cd backend
docker build -t artifact-comparator-backend .

Frontend:

Bash
cd frontend
docker build -t artifact-comparator-frontend .

Volumes

Volume	Purpose
postgres_data	Database persistence
artifact_storage	Uploaded artifacts storage

---

10. Coding Conventions

Java (Backend)

Naming Conventions

- **Classes**: PascalCase (e.g., UserService)
- **Methods/Variables**: camelCase (e.g., getUserById)
- **Constants**: UPPER_SNAKE_CASE (e.g., MAX_FILE_SIZE)
- **Packages**: lowercase (e.g., com.artifactcomparator.service)

Code Style

Java
```
@Service
@RequiredArgsConstructor  // Use Lombok for constructor injection
@Slf4j                    // Use Lombok for logging
public class ExampleService {

    private final ExampleRepository exampleRepository;

    public Example findById(Long id) {
        return exampleRepository.findById(id)
            .orElseThrow(() -> new EntityNotFoundException("Example not found"));
    }
}
```

Best Practices

- Use **Lombok** annotations (@Data, @RequiredArgsConstructor, etc.)
- Use **Optional** for nullable returns
- Add **validation** annotations to DTOs (@NotNull, @Size, etc.)
- Log important operations using @Slf4j

JavaScript/React (Frontend)

Naming Conventions

- **Components**: PascalCase (e.g., UserProfile.jsx)
- **Functions/Variables**: camelCase (e.g., handleSubmit)
- **Constants**: UPPER_SNAKE_CASE (e.g., API_URL)
- **Files**: PascalCase for components, camelCase for utilities

Code Style

JavaScript

```
// Use functional components with hooks
const UserProfile = ({ userId }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchUser();
  }, [userId]);

  const fetchUser = async () => {
    try {
      const response = await userService.getUserById(userId);
      setUser(response.data);
    } catch (error) {
      console.error('Failed to fetch user:', error);
    } finally {
      setLoading(false);
    }
  };

  if (loading) return <CircularProgress />;

  return (
    <Box>
      <Typography variant="h5">{user?.name}</Typography>
    </Box>
  );
};

export default UserProfile;
```

Best Practices

- Use **functional components** with hooks

- Destructure props in component parameters
- Handle loading and error states
- Use **async/await** for API calls
- Keep components small and focused

SQL (Migrations)

```
SQL
-- Use uppercase for SQL keywords
-- Use snake_case for table/column names
CREATE TABLE user_profiles (
    id BIGSERIAL PRIMARY KEY,
    user_id BIGINT NOT NULL REFERENCES users(id),
    bio TEXT,
    avatar_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Add indexes for foreign keys and frequently queried columns
CREATE INDEX idx_user_profiles_user_id ON user_profiles(user_id);
```

---

11. Git Workflow

Commit Messages

Use conventional commits:

```
feat: add artifact comparison feature
fix: resolve PDF rendering issue in Chrome
docs: update developer manual
refactor: simplify authentication flow
test: add unit tests for StudyService
```

Pull Request Process

1. Create feature branch from main
2. Make changes and commit
3. Push branch and create PR
4. Request code review
5. Address feedback
6. Merge after approval

Code Review Checklist

- [ ] Code follows project conventions
- [ ] Tests are included/updated
- [ ] No hardcoded credentials
- [ ] Error handling is appropriate

- [ ] Documentation is updated if needed

---

12. Troubleshooting

Common Issues

Database Connection Failed

Error: Connection refused to localhost:5433

Solution:

Ensure PostgreSQL is running:
Bash
docker ps | grep postgres

1.
2. Check if the port is correct in application.properties
3. Verify database credentials

Frontend Can't Connect to Backend

Error: Network Error or CORS error

Solution:

1. Ensure backend is running on port 8080

Check CORS configuration in application.properties:
Properties
cors.allowed.origins=http://localhost:3000

2.
3. Verify proxy in frontend/package.json

Flyway Migration Failed

Error: Migration checksum mismatch

Solution:

1. Never modify existing migrations

If in development, reset database:
Bash
docker-compose down -v
docker-compose up

2.

Use repair script:
Bash
./scripts/fix-migration-history.sh

3.

JWT Token Issues

Error: 401 Unauthorized

Solution:

1. Check token expiration (default: 24 hours)
2. Verify jwt.secret matches in configuration
3. Clear localStorage and login again

Large File Upload Fails

Error: Request Entity Too Large

Solution:

Check file size limit in application.properties:
Properties
spring.servlet.multipart.max-file-size=100MB
spring.servlet.multipart.max-request-size=100MB

1.
2. If using nginx, increase client_max_body_size

Useful Commands

Bash
```
# View backend logs
docker-compose logs -f backend

# Access database
docker exec -it artifact-comparator-db psql -U admin -d artifact_comparator

# Clear all Docker data
docker-compose down -v --rmi all

# Rebuild specific service
docker-compose up --build backend

# Check running containers
docker ps

# Frontend dependency issues
cd frontend && rm -rf node_modules && npm install
```

Debug Mode

Backend: Set in application.properties:

Properties
```
logging.level.com.artifactcomparator=DEBUG
logging.level.org.springframework.security=DEBUG
```

Frontend: Use browser DevTools (F12):

- Console tab for errors
- Network tab for API calls
- React DevTools extension

Quick Reference

Task	Command
Start all services	docker-compose up
Start with hot-reload	docker-compose -f docker-compose.yml -f docker-compose.dev.yml up
Run backend only	cd backend && mvn spring-boot:run
Run frontend only	cd frontend && npm start
Run backend tests	cd backend && mvn test
Run frontend tests	cd frontend && npm test
Build backend	cd backend && mvn clean package
Build frontend	cd frontend && npm run build
View logs	docker-compose logs -f <service>
Stop services	docker-compose down

Reset database	docker-compose down -v

Contact & Resources

- **Project Repository:** https://github.com/CS319-25-FA/S2-T9
- **Issue Tracker:** Use GitHub Issues for bug reports
- **Documentation:** See deliverables/ folder for additional docs

---

Last Updated: December 2025