IHSAN DOGRAMACI BILKENT UNIVERSITY

CS319: Object-Oriented Software Engineering

Deliverable-4

Group: Insight Coding

Group Members:	Mustafa Mert Gülhan	22201895
	Ece Bulut	22202662
	Yasemin Altun	22202739
	Altay İlker Yiğitel	22203024
	Efe Erdoğmuş	22203553

Instructor: Anıl Koyuncu

Submission Date: 30 November, 2025

Design Goals

1. Role-Based Access and Security Priority

Why this goal:
The system deals with the sensitive data related to scientific research and has multiple types of users(researchers, participants, admins, reviewers)
The critical functionality that needs to be addressed here is the prevention of access by each type of user and the unauthorized user to the data related to an authorized user.

Tradeoffs:
Chosen: Using role-based permissioning in conjunction with JWT authentication and the Spring Security framework.
Rejected Alternative: A reduced single-role model where the entire system has access would harm the integrity and confidentiality of the findings.
Impact: The approach has the effect of adding complexity to implementing the model but this is necessary for software operability and institutional requirements

2. Scalability and Multi-Study Support

Why this goal:
The system has to support parallel studies performed by different researchers, where each investigation may involve multiple individuals submitting their reviews. There must also be the ability to store and access large artifacts.

Tradeoffs:
Chosen:Create Backend and Frontend services running in Docker containers, and include a functionality-rich database.
Rejected Alternative: Everything in one service-> could potentially cause instabilities and performance issues when under a high load, violating software engineering principles.
Impact: The initial configuration takes longer; the system has the ability to handle multiple studies running together with a large number of subjects.

3. Usability and Intuitive User Experience

Why this goal:
Participants with different skill sets need to be able to undergo the evaluations related to improving the user experience, and the researchers need rapid study setup and monitoring tools.

Tradeoffs:
Chosen: Use the Material-UI component library to offer rich dashboards, side-by-side comparison screens, and dynamic task assignment.
Rejected Alternative: Keeping the user interface minimal with basic forms would raise the barriers for the participants and the researchers, reducing the usability.
Impact: The size of the frontend would include a larger package size and longer development time.

4. Extensibility & Artifact Type Flexibility

Why this goal:
Research in software engineering matures (code, UML models, requirements, LLM results). The Platform must be able to support new types of artifacts in the future without changing its architecture. Custom criteria and builders of questionnaires allow study flexibility.

Tradeoffs:
Chosen: Modular artifact upload system with customizable evaluation criteria and annotation tools
Rejected Alternative: Hard-coded types of artifact evaluation criteria → Research would be less applicable & constantly reworked
Impact: Increased complexity of data models and validating logic but allows long-term platform relevance

5. Data Integrity & Reproducible Research

Why this goal:
The results of research must be trustworthy; anything assessed by human evaluation must be such that the data can't be inadvertently altered or destroyed. Other researchers would be able to replicate these results by examining data that had been exported to determine what data had actually been gathered. Export features let researchers analyze data in tools like Excel.

Tradeoffs:
Chosen: Strict data entry rules including validation checks, automatic saving of data, and export options for full data sets.
Rejected Alternative: Users would be allowed to type anything into text fields.
Result: Inaccurate and inconsistent data would be gathered.
Impact: The users receive more error messages if they provide incorrect data; nevertheless, results from research can be trustworthy and publishable.

Connectors in Insight Coding

1. REST API (HTTP)

Connects: Frontend ↔ Backend
Why: Standard web protocol, works in all browsers, easy to debug
Alternative: WebSockets (overkill – no real-time chat needed), GraphQL (too complex)

2. Spring Data JPA

Connects: Backend ↔ PostgreSQL
Why: Auto-generates SQL, prevents injection attacks, less code
Alternative: Raw SQL (tedious to write, lots of errors), NoSQL (ineffective storage solution for relational data)

3. Docker Network

Connects: All containers together
Why: Auto service discovery, isolation, consistency across platforms
Alternative: Localhost setup (messy ports, deployment challenges), Legacy (Deprecated in favor of networks)

4. JWT Tokens

Connects: Auth across requests
Why: Stateless, scalable, no server-side session storage
Alternative: Session cookies (requires shared session store, harder to scale)

5. React Context

Connects: Component state sharing
Why: Built-in, simple, sufficient for auth state
Alternative: Redux (overkill for small state), prop drilling (unmaintainable)

6. Docker Volumes

Connects: File persistence
Why: Survives restarts, fast file I/O, keeps DB lean
Alternative: Database BLOBs (slow, increases DB size), container storage (loses data if restarted)

Architectural Style: Layered Architecture + Client–Server

Architecture Used

Three-Tier Layered Architecture:

Presentation Layer — React Frontend (UI Components & Pages)
Business Logic Layer — Spring Boot Backend (Controllers & Services)
Data Layer — PostgreSQL Database (Entities & Repositories)

Client–Server Pattern:
Browser acts as client, Spring Boot API as server

Stateless REST communication between layers

Rejected Alternatives:
Monolithic — Tight coupling, no modern frontend integration
Microservices — Too complex for project size; would require separate services for users, artifacts, studies
Event-Driven — No asynchronous event workflows needed
Single-App MVC — Doesn't support a React-first architecture

Subsystem Decomposition Diagram