

**Design document**

**Author** : Andrii Matviienko

**Version**: 1.0

**Date**: 28.03.2025

[**Context :** 2](#_Toc194085763)

[**C1:** 3](#_Toc194085764)

[**Software Principles Applied:** 4](#_Toc194085765)

[**С2:** 5](#_Toc194085766)

[**Software Principles Applied:** 6](#_Toc194085767)

[**С3:** 7](#_Toc194085768)

[**Software Principles Applied:** 8](#_Toc194085769)

# **Context :**

This design document defines the architecture of WiredSpace, explains technology choices, and provides C4 Model diagrams. It ensures clarity, consistency, and a shared understanding among developers and stakeholders.

# **C1:**

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

This **System Context (C1) diagram** shows how different entities interact with **WiredSpace**:

1. **Users** create, consume, and interact with social media content through the system.
2. **Admins** manage, moderate, and configure the platform.
3. **WiredSpace** handles API requests and processes interactions.
4. **Email System** sends confirmation emails and notifications to users.

## **Software Principles Applied:**

* **KISS (Keep It Simple, Stupid)** – The architecture ensures simplicity by clearly defining roles for each component, reducing unnecessary complexity.
* **YAGNI (You Ain’t Gonna Need It)** – Features that are not immediately required are not included to keep the system lightweight.

# **С2:**

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

The **Container Diagram (C2)** illustrates the high-level architecture of WiredSpace. The **Frontend (React)** provides the user interface and interaction layer. The **Backend (Spring Boot)** processes business logic and manages API requests. The **Database (MySQL)** stores user data and content. The **Email System** sends notifications and confirmations via SMTP. **Users** and **Admins** interact with the system through the frontend.

## **Software Principles Applied:**

* **SOLID**
  + **Single Responsibility Principle (SRP):** Each service (frontend, backend, database, email system) has a well-defined responsibility.
  + **Dependency Inversion Principle (DIP):** The backend communicates with the database via repositories, avoiding direct dependencies.
* **DRY (Don't Repeat Yourself):** Common functionalities such as authentication and notifications are centralized to avoid redundant code.

# **С3:**

Изображение выглядит как текст, снимок экрана, Самоклеющийся листок, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

С3 diagram represents the high-level architecture of the **WiredSpace** API application.

1. The **Frontend** (React-based) allows users to interact with the system by sending API requests.
2. The **API Application** (Spring Boot) consists of controllers, services, and repositories that handle business logic and data persistence.
3. The **Post Controller** and **User Controller** manage HTTP requests, delegating logic execution to their respective services.
4. The **Post Service** and **User Service** execute business logic and interact with repositories for data persistence.
5. The **Post Repository** and **User Repository** handle database operations using MySQL.
6. The **Email Service** manages communication with the **Email System**, which sends emails to users.
7. The **Database (MySQL)** stores user-generated content and system information.

## **Software Principles Applied:**

* **SOLID**
  + **Open/Closed Principle (OCP):** Services are designed to be extensible without modifying existing code.
  + **Liskov Substitution Principle (LSP):** Repositories and services follow interfaces, ensuring flexibility in implementation.
* **DRY:** Reusable components (e.g., shared services for logging and notifications) minimize code duplication.
* **KISS:** The API architecture follows clear separation of concerns, making it easy to understand and maintain.
* **YAGNI:** Advanced optimizations (e.g., caching, load balancing) are deferred until necessary to avoid premature complexity.