Just Your KidieZ

Architecture Document

Date	**:**	**6/18/2023**
Version	**:**	**4**
State	**:**	**Released**
Author	**:**	**Aya Shikh Suliman**

Version history

Version	Date	Author(s)	Changes	State
0.1	3/23/2023	Aya Shikh Suliman	C1, C2 and C3 models Sequence diagram SOLID principles Introduction Design choices	Released
1	4/1/2023	Aya Shikh Suliman	Sprint 2 review feedback	Released
1.1	4/14/2023	Aya Shikh Suliman	C4 model	Released
2	4/14/2023	Aya Shikh Suliman	C4 model	Released
2.1	5/12/2023	Aya Shikh Suliman	CI diagram	Released
3	5/12/2023	Aya Shikh Suliman	C4 model	Released
4	618/2023	Aya Shikh Suliman	CI diagram	Released

Table of contents:

1. Introduction:

This document provides a comprehensive architectural overview of the system, using several different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

2. Design decisions

2.1 Why Spring Boot?

Spring Boot helps developers create applications that just run. Specifically, it lets you create standalone applications that run on their own, without relying on an external web server, by embedding a web server such as Tomcat or Netty into your app during the initialization process.

2.2 Why react?

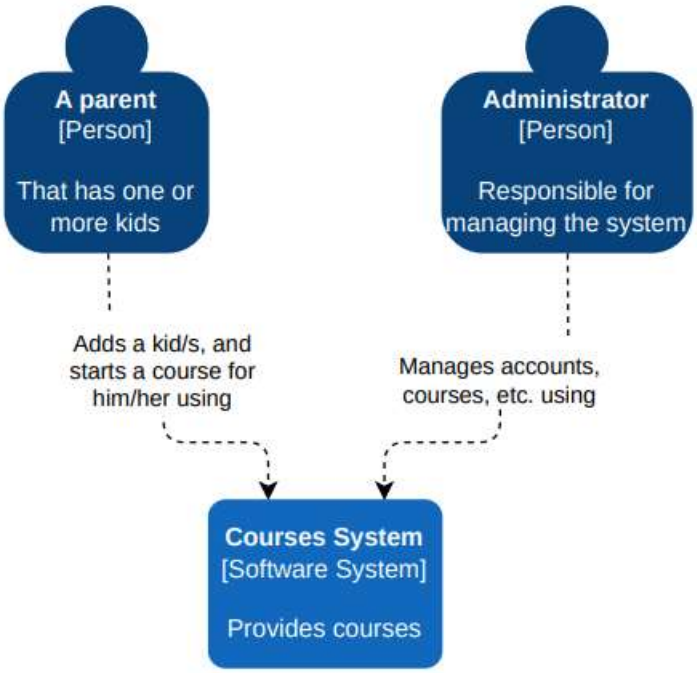
React basically allows developers to utilize individual parts of their application on both the client-side and the server-side, which ultimately boosts the speed of the development process. In simple terms, different developers can write individual parts and all changes made won't cause the logic of the application.

2.3 Why MySQL?

MySQL is ideal for storing application data, specifically web application data. Additionally, you should use MySQL if you need a relational database which stores data across multiple tables. As MySQL is a relational database, it's a good fit for applications that rely heavily on multi-row transactions.

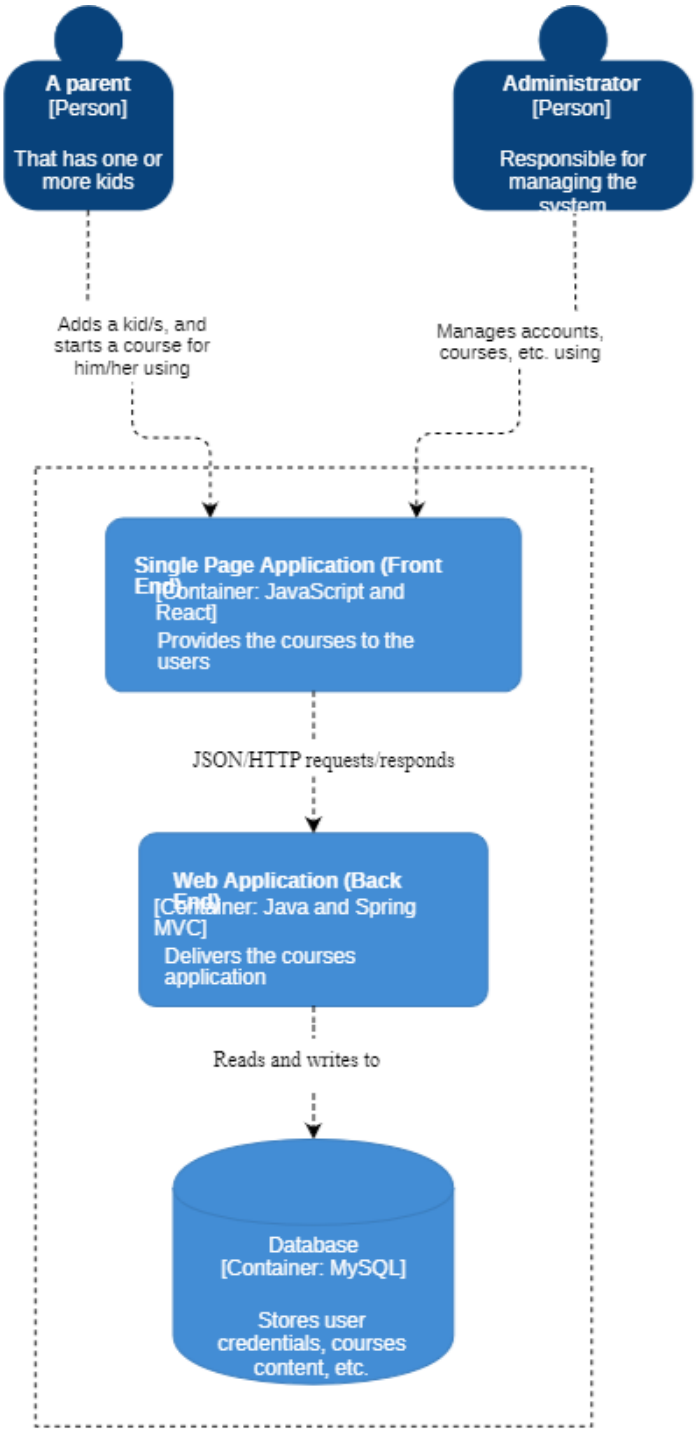
3. C4 models

3.1 C1 model



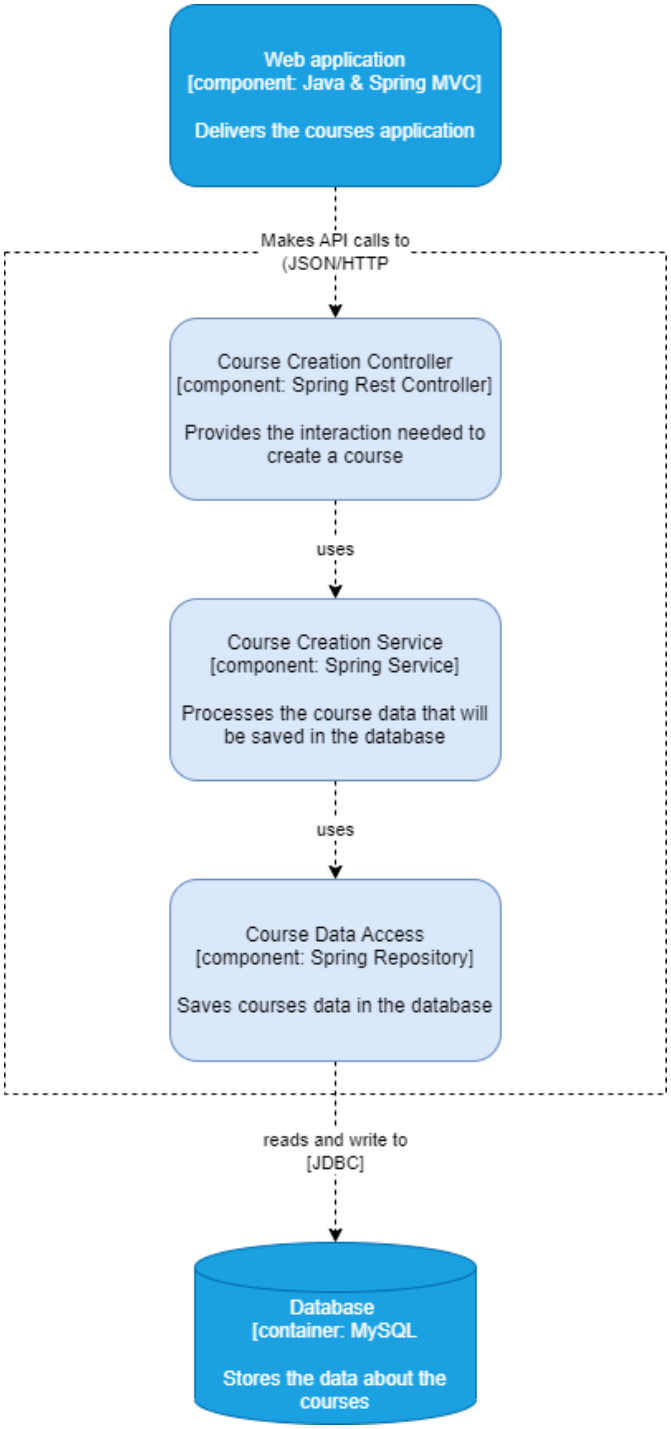
In this diagram, a high-level view of our system is displayed. It shows the boundaries of our system and who interacts with it. It shows that there are two users (parent, and admin) interacting with the courses system that is going to be built.

3.2 C2 model



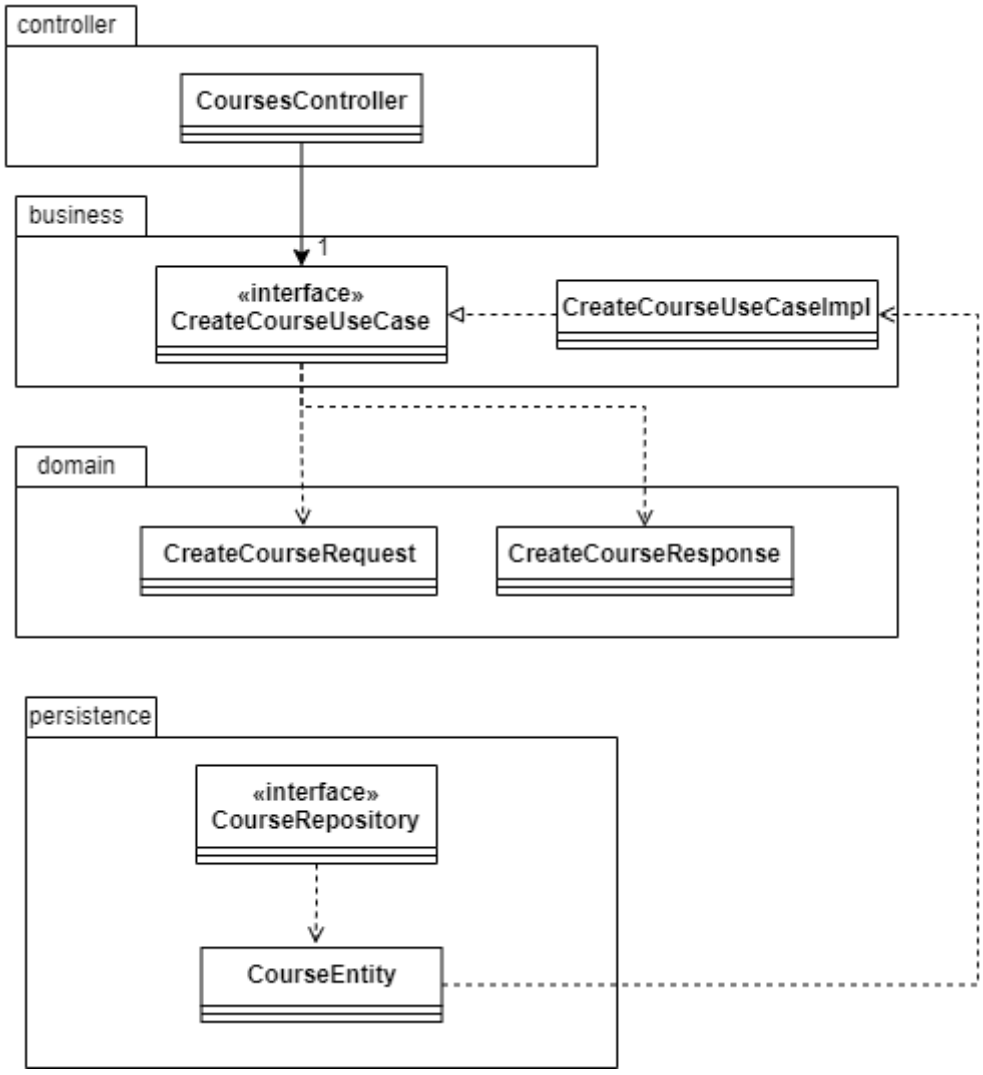
Here we go more in-depth into the course's application itself. It shows that the users will only directly interact with the front end (React). And the front end will then interact with the backend (Java & Spring MVC). All the data which needs to be saved will be stored in the MySQL database.

3.3 C3 model



In this diagram, we dive into the backend (Java & Spring MVC) API application. We show the different components inside and how they are connected. The single-page application is making API calls, which are connecting to the controllers from the API application. Then the controller layer uses the course creation service which is the business layer. After that the business layer uses the course data access layer which is the persistence layer which is off course reads and writes to the outside database.

3.4 C4 Model:


controller
CoursesController
business
1
«interface»
CreateCourseUseCase
CreateCourseUseCaseImpl
domain
CreateCourseRequest
CreateCourseResponse
persistence
«interface»
CourseRepository
CourseEntity

4. Software principles

4.1 The SOLID principle:

In this project, the SOLID principles are going to be used.

4.1.1 Single responsibility principle

For each topic, a class is going to be created so that each class is going to be responsible for one thing in the application.

4.1.2 Open-closed principle

The classes are going to be open for extension but closed for modification so that the source code is set and cannot be changed.\

4.1.3 Liskov substitution principle

It is still not sure if this principle is going to be used. After diving more in the code, a decision is going to be made.

4.1.4 Interface segregation principle

Each interface in the application is going to be responsible for one thing/topic. No big interfaces are going to be implemented.

4.1.5 Dependency inversion

High-level modules are going to depend on abstractions rather than concrete implementations.

4.2 The KISS principle:

KISS stands for Keep It Simple, Stupid. This software design principle states that designs, solutions, systems, and products work best if they're kept simple. In this project, simplicity will be favored over complexity and complexity is going to be avoid as much as possible.

4.3 The DRY principle:

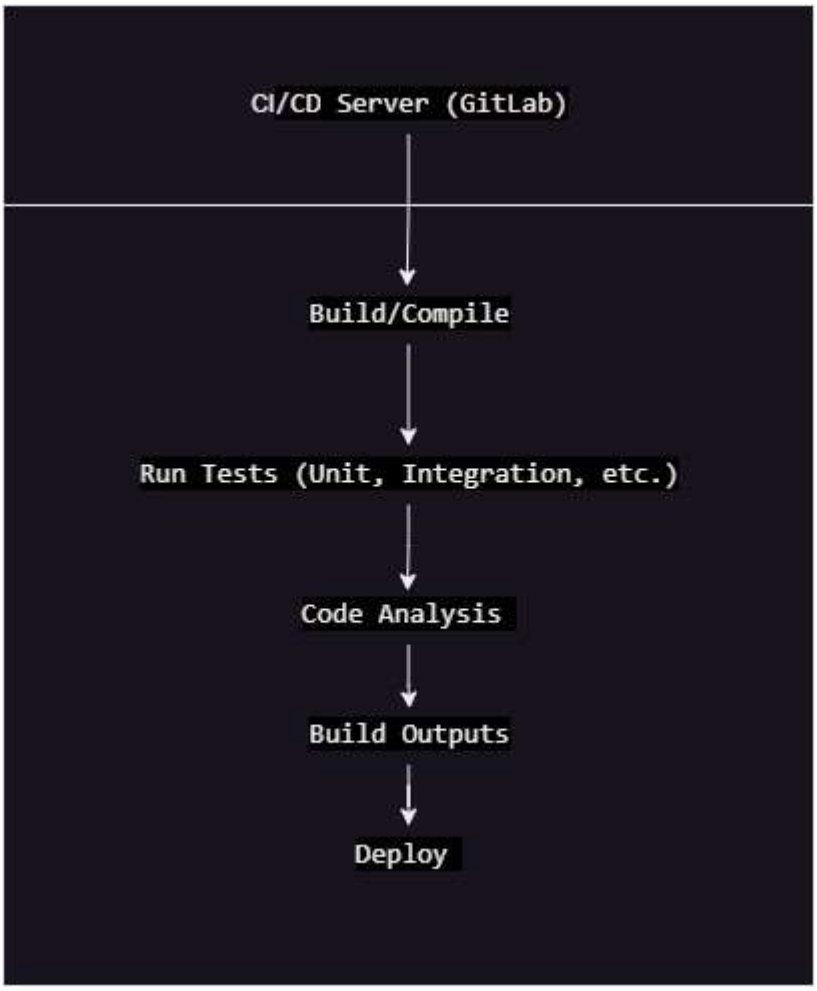
The DRY (don't repeat yourself) principle is a best practice in software development that recommends software engineers to do something once, and only once. In this project, this principle will be applied as much as possible.

4.4 The YAGNI principle:

YAGNI principle ("You Aren't Gonna Need It") is a practice in software development which states that features should only be added when required. In this project, this principle will be used when applicable.

5. CI Diagram

In this diagram:



1**. Developer's Machine:** Developers write code and make changes on their local machines using their preferred programming tools.

2. **Version Control System (Git):** Developers commit and push their code changes to a remote version control system, such as Git, to track and manage the codebase.

3. **CI/CD Server (GitLab CI):** The CI/CD server is responsible for managing the CI/CD process. It continuously monitors the version control system for changes and triggers the CI pipeline accordingly.

4**. Build/Compile:** The CI server retrieves the latest code changes and initiates the build process. It compiles the source code and generates the necessary binaries or artifacts.

5. **Run Tests:** The CI server runs various types of tests, such as unit tests, integration tests, or other automated tests, to ensure the code changes are functioning as expected.

6. **Code Analysis:** The CI server performs code analysis tasks, such as linting or static analysis, to identify potential issues, bugs, or violations of coding standards.

8. **Deploy:** The CI server can optionally deploy the generated artifacts to a staging environment or a production environment, depending on the project's requirements.