Software Architecture Document(SAD)

for RealtyHub

Radu Andrei Popescu

# General information:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** |  | **:** |  | **7 April 2023** |
| **Version** |  | **:** |  | **1.0** |
| **State** |  | **:** |  | **Finished** |
| **Author** |  | **:** |  | Radu Andrei Popescu |

## Version history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 1.0 | 7/04/2023 | Radu Andrei Popescu | Document’s main structure and content | Finished |
|  |  |  |  |  |

## Distribution history

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Receivers** |
| 1.0 | 14/04/2023 | Mrs. Gupta, Mrs. Pejic, Mr.Faruk |
|  |  |  |

Table of contents

# Contents

[General information: 1](#_Toc7950)

[Version history 1](#_Toc7951)

[Distribution history 1](#_Toc7952)

[Table of contents 2](#_Toc7953)

[Introduction 3](#_Toc7954)

[Architecture overview 3](#_Toc7955)

[SOLID explanations 7](#_Toc7956)

[Single Responsibility Principle 7](#_Toc7957)

[Open Closed Principle 7](#_Toc7958)

[Interface Segregation Principle 8](#_Toc7959)

[Dependency Inversion Principle 8](#_Toc7960)

[Architectural decisions 8](#_Toc7961)

[Technologies used 8](#_Toc7962)

# Introduction

This document is focused on describing the architecture of the software system, which is a critical aspect of the overall AGILE SCRUM software development process. The architecture represents a blueprint for the development team, outlining how the system is organized and how its different components work together. By documenting the architecture in a SAD, we are aiming to provide a clear and concise description of our system's design, as well as the reasoning behind all key architectural decisions so that future maintenance is easy to conduct. The SAD is a continuously evolving document that will evolve over the course of this project as the system’s design changes and evolves. That being said, this document should be considered a starting point for discussions about the system architecture and should be updated as needed to reflect changes in the system design or requirements.

# Architecture overview

1. The following diagram represents a high-level view of our system’s design. This is also known as the C1 diagram.

A picture containing text, screenshot, diagram

Description automatically generated

1. Further we have the C2 diagram that represents a zoom in on the Software System component of the C1 diagramA picture containing text, screenshot, diagram, design

   Description automatically generated

1. Next up we have the C3 diagram that represents an even greater amount of detail on how the back-end part of the software system is built.

A screenshot of a diagram

Description automatically generated with low confidence

1. The next level is C4, which is the dependency diagram of the solution.

A screen shot of a computer

Description automatically generated with low confidence

# SOLID explanations

## Single Responsibility Principle

The system I am designing complies with the Single Responsibility Principle since it has a very clear separation of concerns by layers and classes. The Controller layer is responsible for receiving the HTTP requests, calling the business rules, and then returning the HTTP response code and body where applicable. The Business layer is where the core app’s operations and validation are implemented. This layer contains manager classes and the interfaces these classes implement. This layer also facilitates access to the data(persistence) layer. The Persistence layer is the one responsible for storing the data in the external database.

## Open Closed Principle

The system I am designing will comply with the Open Closed Principle because of how the Persistence layer is structured. The layer’s foundation is an interface that contains the CRUD operations. This interface is then implemented by the classes responsible for communicating with the storage service. Since the interface is at the core of this design the layer is open for extension, so that there can be more storage systems used but closed for modification because of the interface that makes sure every class implementing it has the CRUD operations.

## Interface Segregation Principle

The system fulfils the rules of the Interface Segregation principle because no client is forced to depend on methods it does not use. Each client of any interface is using every method of the interface it implements.

## Dependency Inversion Principle

The system respects the rules of the Dependency Inversion Principle because the separation between the application’s architectural layers is done through interfaces. This way high level modules do not depend on low level modules. Both parts depend on abstractions. This principle is also a key component of proper unit testing in order to make sure that the testing data does not get stored in the database.

# Architectural decisions

The system is being designed with a hybrid approach in mind that is combining the “API first approach” with the “Continuous integration and Deployment approach”.

The API-first approach means that a RESTful API was designed before any implementation details were considered. This decision was made to ensure that the system's RESTful API is well-designed, easy to use, and can evolve independently of the implementation.

The “Continuous integration and Deployment approach” is especially practical for a software development project using the AGILE SCRUM methodology as it facilitates the delivery process while ensuring that the deployed pieces of code are working perfectly and passing the software quality tests. designed for continuous integration and deployment, which means that code changes are automatically Put simply, this decision was made to improve the system's quality, reliability, and speed of delivery.

These architectural decisions were made to achieve the system's design goals, such as scalability, flexibility, resilience, and maintainability. While these decisions may impose some constraints on the system's design and implementation, they are intended to enable the system to meet its functional and non-functional requirements while providing a solid foundation for future maintenance and extensibility.

# Technologies used

* Why Java?
  + - **Platform independent**: Java is a platform-independent language, meaning it can be deployed on any operating system without any changes to the code.
    - **Security**: Java has built-in security features that can help protect my system from attacks like cross-site scripting and SQL injection.
    - **Framework-friendly**: Java offers a wide range of libraries and frameworks that can help you build a RESTful API quickly and easily, such as Spring.
* Why Spring Boot?
  + **Rapid development**: Spring Boot provides a range of tools and libraries that make the development process easier, allowing me to build a RESTful API quickly and efficiently. o **Integration with other technologies**: Spring Boot is also compatible with other Java technologies, such as the Hibernate ORM, making it easy to work with databases.
  + **Testing**: Spring Boot makes it easy for me to test my API using tools like JUnit and Mockito, ensuring that the code is robust and reliable.
* Why MySQL?
  + **Performance**: MySQL is optimized for speed and can perform well even when dealing with complex queries or large datasets. It is also designed to work well with in-memory caching systems, which can improve my system’s performance further. o **Integration with Hibernate ORM**: Hibernate is a popular ORM framework for Java. It works well with MySQL, making it easy to map object-oriented designs to a relational database structure.
  + **Ease of use**: MySQL is easy to install, set up and use, with a range of tools available to help me manage and monitor my relational database system.
* Why REACT.js?
  + **Component-based architecture**: React.js is built using the concept of components, which represent reusable and encapsulated pieces of code that can be used to build complex user interfaces. This makes it easy to create and manage intuitive and good-looking user interfaces.
  + **Virtual DOM**: React.js uses a virtual DOM(Document Object Model), which is a lightweight representation of the HTML DOM. This allows React.js to update the user interface efficiently and minimizes the amount of time it takes to render changes using a process called diffing(basically spotting the differences between what I am trying to render and the current version of the HTML DOM).
  + **Compatibility with other specified technologies**: React.js can be used with the previously stated technologies such as Java and Spring Boot, in order to enhance the functionality and performance of the website.