# Student Housing: Roomies

Software Architecture Document
S3-ITS

Lex de Kort

November 25, 2022

## Version History

| Version | Date | Changes | State |
|---|---|---|---|
| 1.0 | October 7, 2022 | Initial release | Incomplete |
| 1.0 | November 25, 2022 | Add CI/CD diagram | Incomplete |

## Distribution

| Version | Date | Receivers |
|---|---|---|
| 1.0 | October 7, 2022 | Frank Coenen & Marcel Boelaars |
| 1.0 | November 25, 2022 | Frank Coenen & Marcel Boelaars |

# Contents

## 1 Introduction

The purpose of this document is to describe the general architecture of the *Student Housing system* (hereafter referred to as *Roomies*). The document will focus on four key aspects:

1. The system context
2. The overall shape of the architecture and the technologies that were chosen that fit best
3. The components that the solution will consist of and their interactions
4. The implementation of these components

This document will ensure that software engineers are easily able to understand the structure of the software on all layers. It will also allow stakeholders to determine if the project has met all the requirements as laid out in this document. In this document the C4 system is utilized to visualize the architecture of the software behind *Roomies*.

## 2 Terminology

- **Container** - A singular solution / application running separately from other solutions / applications that performs certain tasks
- **Component** - Part of a container, has a single set of responsibilities within a container
- **Front-end** - The visible elements of an application; the part of an application that users interact with
- **Back-end** - The under-the-hood logic of an application that users generally do not see

## 3 System Context

The application itself is envisioned as a system that both students and landlords can interact with. Most of the functionality will be built into the app, but certain aspects will require the use of an external system. In the case of our app, we will require an external email system to send emails to our users.
Later down the road we could also implement some other external services such as a location service or mapping service for displaying the actual location of a listing.

On the next page you will find a diagram of the C1 level, displaying how the current context is envisioned. Keep in mind that it is not a final design and is subject to change.

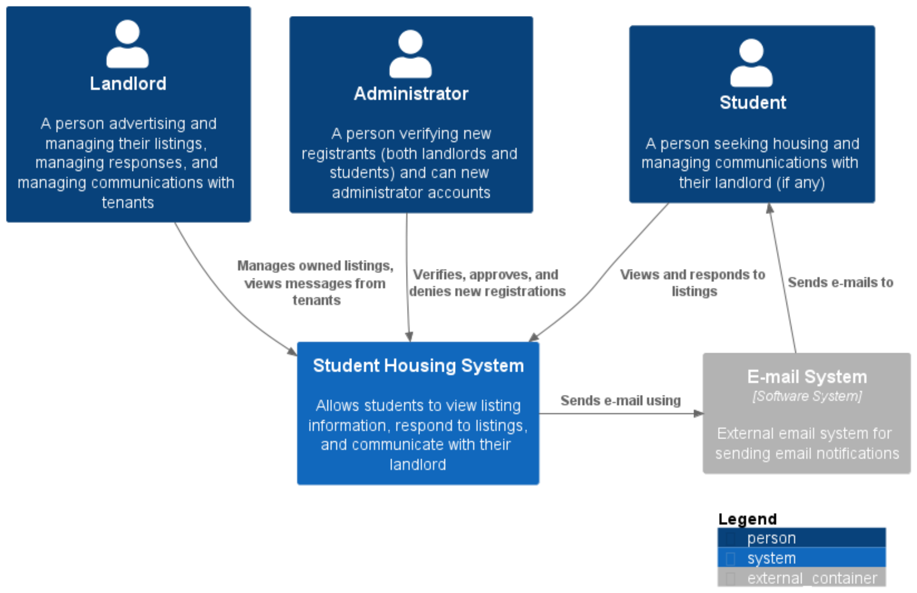
Landlord
A person advertising and managing their listings, managing responses, and managing communications with tenants
Administrator
A person verifying new registrants (both landlords and students) and can new administrator accounts
Student
A person seeking housing and managing communications with their landlord (if any)
Manages owned listings, views messages from tenants
Verifies, approves, and denies new registrations
Views and responds to listings
Sends e-mails to
Student Housing System
Allows students to view listing information, respond to listings, and communicate with their landlord
Sends e-mail using
E-mail System
[Software System]
External email system for sending email notifications
Legend
person
system
external_container

Figure 1: C1 Diagram

# 4 Containers and tech choices

## 4.1 Containers

The system itself is split up in several containers, each responsible for their own set of functionality. Users of the system will interact with the system through a single-page application, which will serve as the *front-end*. The *front-end* will take the form of a website. All necessary functionalities will be available from there. Which actions are available depends on the role of the user.

The back-end will consist of an API application that runs separately from the website. It will host several end-points that the front-end can hook onto to retrieve any information the front-end needs / requests.

By utilizing a database we can store and read any data a user might create or request, ensuring that the data persists beyond whether the application is active or not. By saving it in a database instead of locally we can also keep any sensitive information secure, ensuring it cannot be tampered with while also allowing users to access their information from anywhere (assuming there is an internet connection of course).

## 4.2 Technologies

The front-end will be built on React. Not only is React one of the most popular JavaScript frameworks on the market, it is also very flexible and easy to maintain. This puts React ahead of many of its competitors, which are often either lacking in those departments or simply not built for such purposes.
The back-end will be built with Java and Spring Boot. Spring Boot is a powerful Java framework that offers a great many powerful features that greatly improve the speed of development of the back-end. Through the use of annotations Spring Boot is capable of automating the process of Dependency Inversion / Injection, reducing development time and allowing developers to spend more time on other features instead. It also allows for an easy and convenient implementation of a RESTful API. This, too. reduces development time.

Java itself is a powerful programming language that has been around for a long time. As a result, it has a great many libraries available and has a huge community surrounding it. It is one of the most used language in the ICT field as a result. One of its core features is that Java is platform agnostic, meaning that it can run on almost any operating system out there. All these make Java a powerful programming language that, to this day, still has a great following while also still being actively developed.

On the next page you will find a diagram of the C2 level, displaying all the current envisioned containers for the system. Keep in mind that it is not a final design of the system and is subject to change.

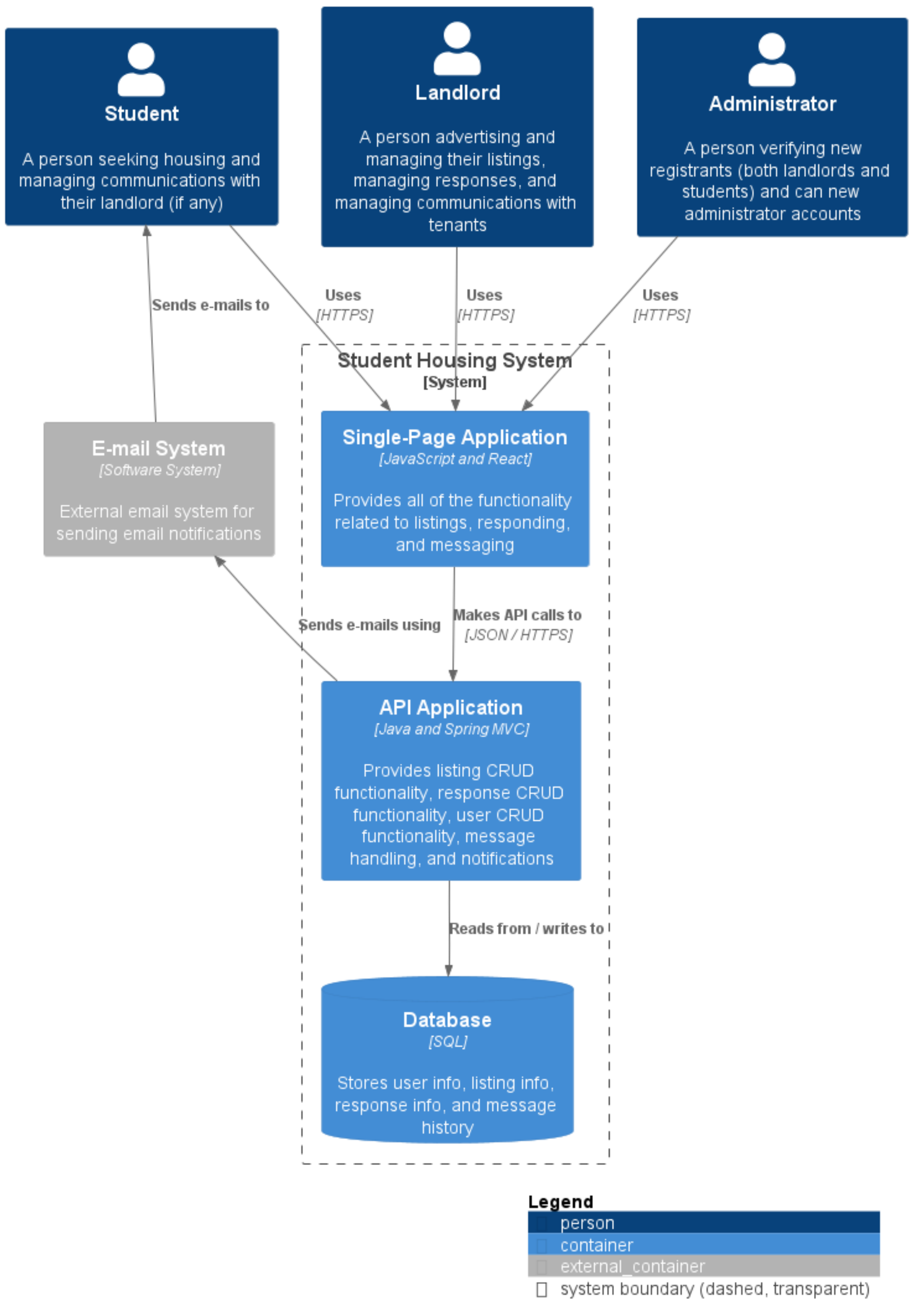
Student
A person seeking housing and managing communications with their landlord (if any)
Landlord
A person advertising and managing their listings, managing responses, and managing communications with tenants
Administrator
A person verifying new registrants (both landlords and students) and can new administrator accounts
Sends e-mails to
Uses
[HTTPS]
Uses
[HTTPS]
Uses
[HTTPS]
Student Housing System
[System]
E-mail System
[Software System]
External email system for sending email notifications
Single-Page Application
[JavaScript and React]
Provides all of the functionality related to listings, responding, and messaging
Sends e-mails using
Makes API calls to
[JSON / HTTPS]
API Application
[Java and Spring MVC]
Provides listing CRUD functionality, response CRUD functionality, user CRUD functionality, message handling, and notifications
Reads from / writes to
Database
[SQL]
Stores user info, listing info, response info, and message history
Legend
person
container
external_container
system boundary (dashed, transparent)

Figure 2: C2 Diagram

# 5 Components

The system consists of several components that each contain their own responsibilities. By separating the responsibilities over their respective classes, we can ensure that the code is logically sorted. This will lead to easily maintainable code in the long run.

## 5.1 Feature approach vs Service approach

For the components there are a few approaches one could take. The two approaches I will focus on in this document are the *Feature Approach* and the *Service Approach*. Each of these approaches has its own pros and cons.
The benefits of the *Feature Approach* are as follows:

- Each class is responsible for the behaviour of a single use case, ensuring that the Single Responsibility Design in SOLID is not violated
- Each class is easily maintainable
- It is easy to see which class is related to which use case

However, its drawbacks are:

- The more use cases you have, the more classes you get, which can quickly get out of control
- Classes have a high chance of having duplicate code

The benefits of the *Service Approach* are as follows:

- There are significantly fewer classes in this approach compared to the *Feature Approach*

However, its drawbacks are:

- Larger risk of manager classes becoming god classes
- Harder to maintain

Ultimately I have decided to with the *Service Approach*. Not only am I more familiar with this approach, with the *Feature Approach* there will inevitably be significantly more code duplication and significantly many classes that it becomes rather hard to oversee what is going on. On top of that, I know how to ensure that manager classes don't become god classes.

## 5.2 Layering

In terms of layers, the application will be separated in layers that all perform their own tasks. The classes will be spread over the following layers:

- **Controller Layer**: allows the front-end and back-end to communicate with each other, and passes information from the business layer to the front-end and from the front-end back to the business layer. This layer does not contain any business logic. The communication with the front-end is handled by use of the RESTful API implementation of the Spring Boot framework for Java.
- **Business Layer**: contains the business logic that the application relies on to perform certain actions such as, for example, creating new listings or registering new accounts. It can communicate with the persistence layer to perform CRUD actions on any data it might handle.
- **Domain Layer**: contains business objects that the business layer uses.
- **Persistence Layer**: handles the saving, storing, retrieving, and deleting of data on a storage system such as a database.

By utilizing the Spring Boot Framework for Java we can implement Dependency Inversion through the usage of annotation tags. Spring Boot will ensure that all components and layers are properly injected into each other, reducing development time by limiting repeating code.

Spring Boot will also handle the RESTful API components, reducing development time on those components. This, in turn, allows me to spend more time on other components that can't be automated quite as easily.

## 5.3 Layer Objects

Each of the three layers has its own objects for handling data. These are listed below.

- **Data Transfer Objects (DTO's)** are used in the controller layer (API)
- **Domain Objects** are used in the business layer
- **Entities** are used in the data access layer

These layers are not aware of data objects of other layers to prevent circular referencing.

## 5.4 Domain Objects

For the business layer I will utilize the following domain objects:

- **Listing Object** - Contains all data relevant to a listing such as address, description, rent per month, surface area, etc.
- **Response Object** - Contains listing ID, user ID, and the time of when the response was made.
- **Account Credentials Object** - Contains sensitive data related to user authorization / authentication. Contents are not sent to the front-end to ensure that it remains safe and secure.
- **Account Summary Object** - Contains personal information related to a user, such as email address, name, age, etc. Does not contain passwords or other sensitive data.
- **Personal Message Object** - Utilized for the messaging system between landlords and tenants. Contains timestamps, messages, and user ID's.

## 5.5 C3 Diagram

On the next page you will find a diagram of the C3 level, displaying all the current envisioned components for the application. Keep in mind that it is not a final design of the application and is subject to change.

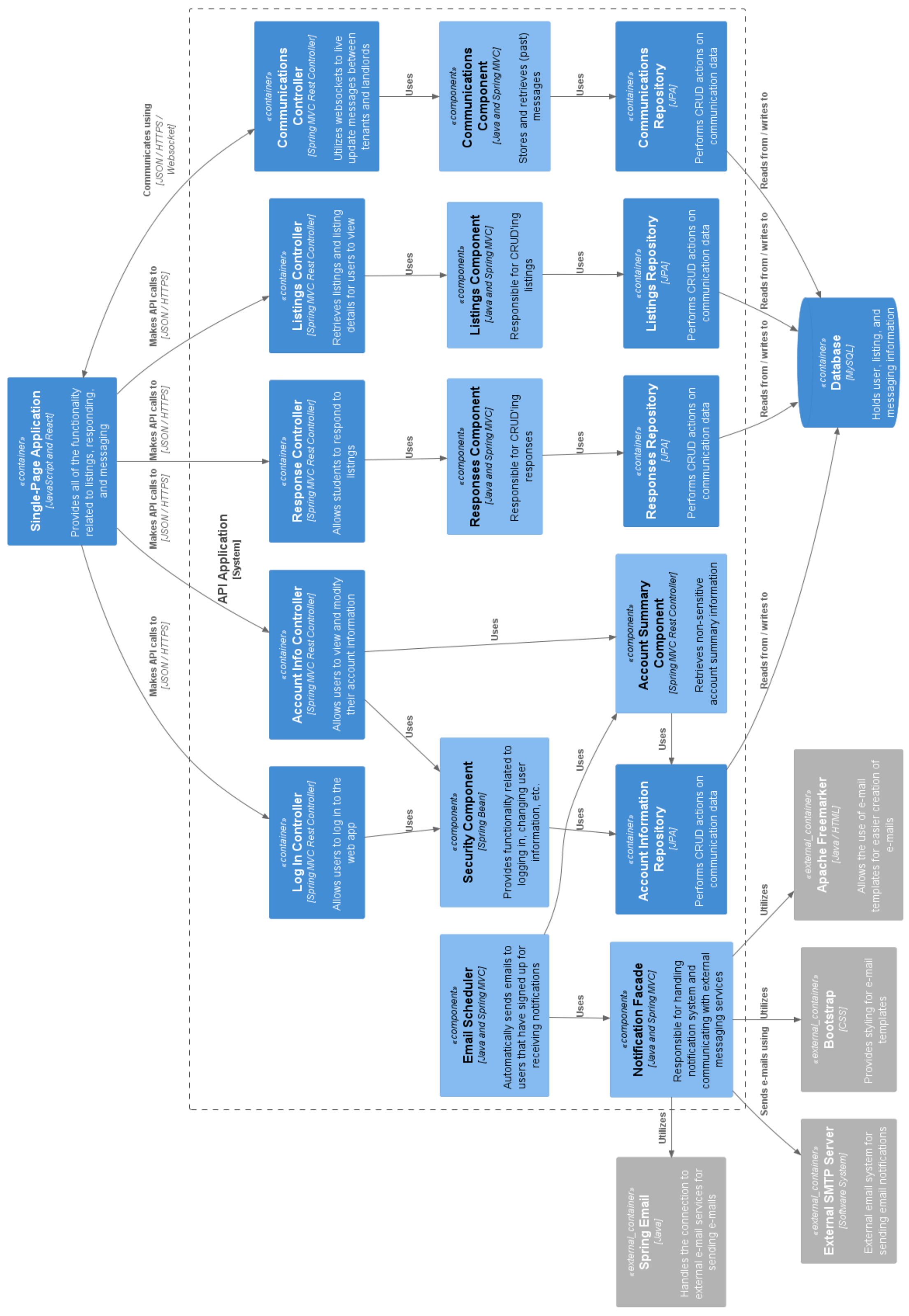
«container» Single-Page Application [JavaScript and React] Provides all of the functionality related to listings, responding, and messaging
API Application [System]
Communicates using [JSON / HTTPS / Websocket]
Makes API calls to [JSON / HTTPS]
Makes API calls to [JSON / HTTPS]
Makes API calls to [JSON / HTTPS]
Makes API calls to [JSON / HTTPS]
«container» Communication Controller [Spring MVC Rest Controller] Utilizes websockets to live update messages between tenants and landlords
«container» Listings Controller [Spring MVC Rest Controller] Retrieves listings and listing details for users to view
«container» Response Controller [Spring MVC Rest Controller] Allows students to respond to listings
«container» Account Info Controller [Spring MVC Rest Controller] Allows users to view and modify their account information
«container» Log In Controller [Spring MVC Rest Controller] Allows users to log in to the web app
Uses
«component» Communications Component [Java and Spring MVC] Stores and retrieves (past) messages
«component» Listings Component [Java and Spring MVC] Responsible for CRUD'ing listings
«component» Responses Component [Java and Spring MVC] Responsible for CRUD'ing responses
«component» Security Component [Spring Bean] Provides functionality related to logging in, changing user information, etc.
«component» Email Scheduler [Java and Spring MVC] Automatically sends emails to users that have signed up for receiving notifications
«container» Communications Repository [JPA] Performs CRUD actions on communication data
«container» Listings Repository [JPA] Performs CRUD actions on communication data
«container» Responses Repository [JPA] Performs CRUD actions on communication data
«component» Account Summary Component [Spring MVC Rest Controller] Retrieves non-sensitive account summary information
«container» Account Information Repository [JPA] Performs CRUD actions on communication data
«component» Notification Facade [Java and Spring MVC] Responsible for handling notification system and communicating with external messaging services
Reads from / writes to
«container» Database [MySQL] Holds user, listing, and messaging information
Utilizes
Sends e-mails using
«external_container» Apache Freemarker [Java / HTML] Allows the use of e-mail templates for easier creation of e-mails
«external_container» Bootstrap [CSS] Provides styling for e-mail templates
«external_container» External SMTP Server [Software System] External email system for sending email notifications
«external_container» Spring Email [Java] Handles the connection to external e-mail services for sending e-mails

Figure 3: C3 Diagram

## 5.6 CI / CD Diagram

The diagram below shows how the CI / CD pipeline is set up for the *Roomies* back-end project.

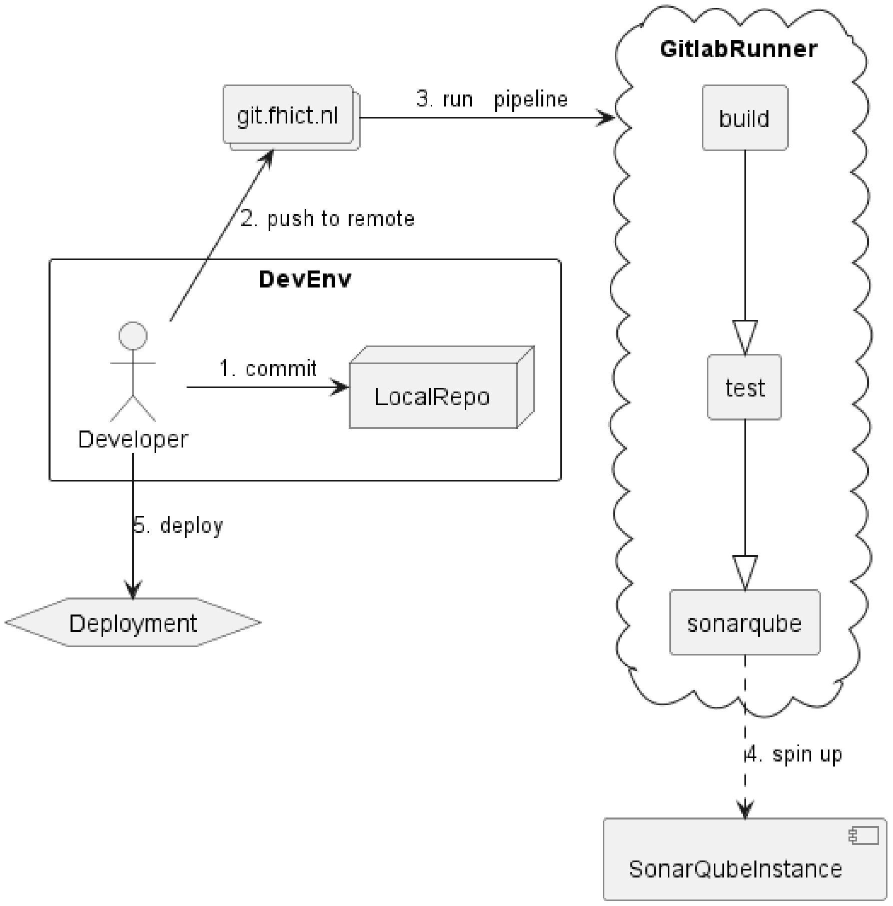

Figure 4: CI / CD Diagram