Date: 24-06-2022

Group: S3-CB01

Student: Oleksandr Gurianov

Version: 4.0

Software Architecture

Table of content

How is SOLID guaranteed

Single-responsibility principle

Every class has its single responsibility so that there is no more than one reason for it to change.

Open-closed principle

Every management class is designed in a way that keeps it closed for modification but opened for a later extension.

Liskov substitution principle

Some subclasses use references from their base classes (e.g., a User can be an Administrator or Customer, a pet can be a Cat or Dog, etc.).

Interface segregation principle

This principle is not used at the moment.

Dependency inversion principle

Every class in the business layer is connected via abstraction to the corresponding class in the database layer. Therefore, classes in the business layer are dependent on the classes in the database layer. Meanwhile, every class in the presentation layer is connected via another abstraction to the corresponding class in the business layer. Thus, such dependency allows classes in the presentation layer to connect to multiple classes in the business layer.

Important Design Decisions

Web framework

My choice fell on the open-source, microservice-based Java web framework called Spring Boot because it allows you to quickly build the application jar and execute it without altering the deployment. Furthermore, it automatically generates opinionated dependencies to make building configuration easier.

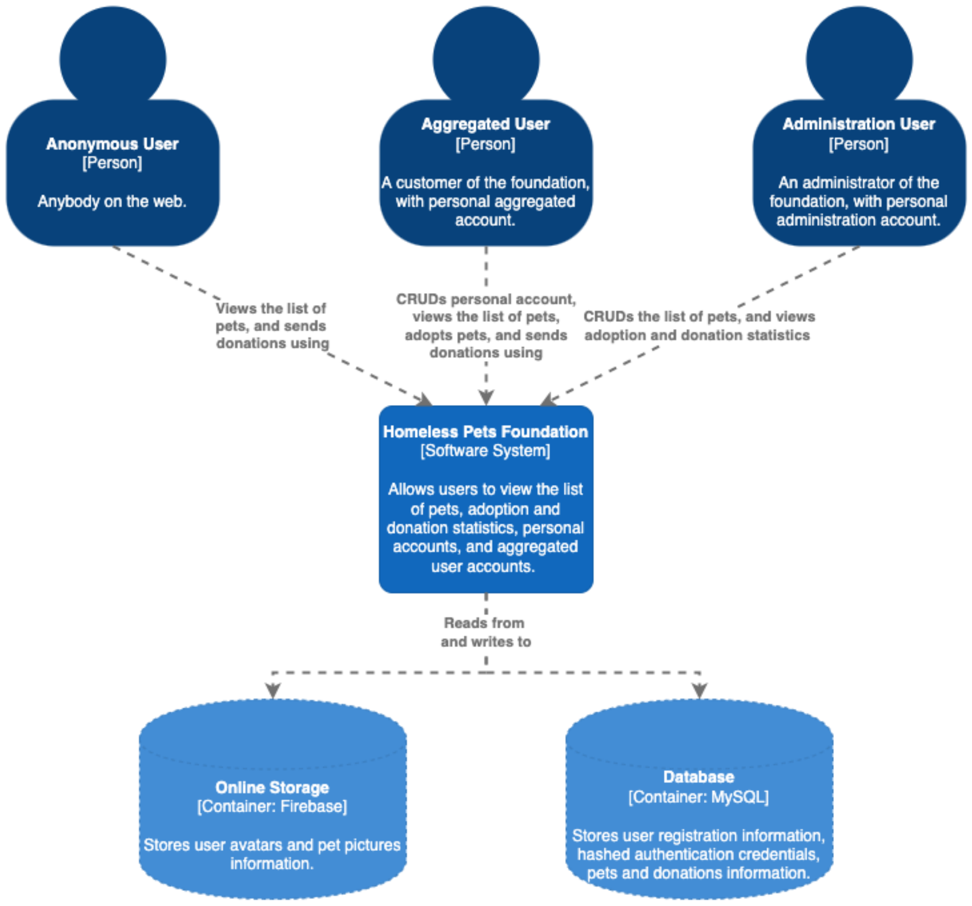
Front-end library

React, an open-source front-end JavaScript library was my choice because it enables programmers to design massive web apps that can alter data without requiring a page reload. Its primary goal is to be quick, scalable, and easy to use. Furthermore, this library is used by many significant corporations worldwide and has a large user base.

Database

Because MySQL, an open-source relational database management system, is used to create highly accessible and scalable internet applications, I believe it is the best option for my future web application. As I am more experienced with SQL-type databases, this knowledge will aid me in developing my web application using this type of database.

System context (C1)

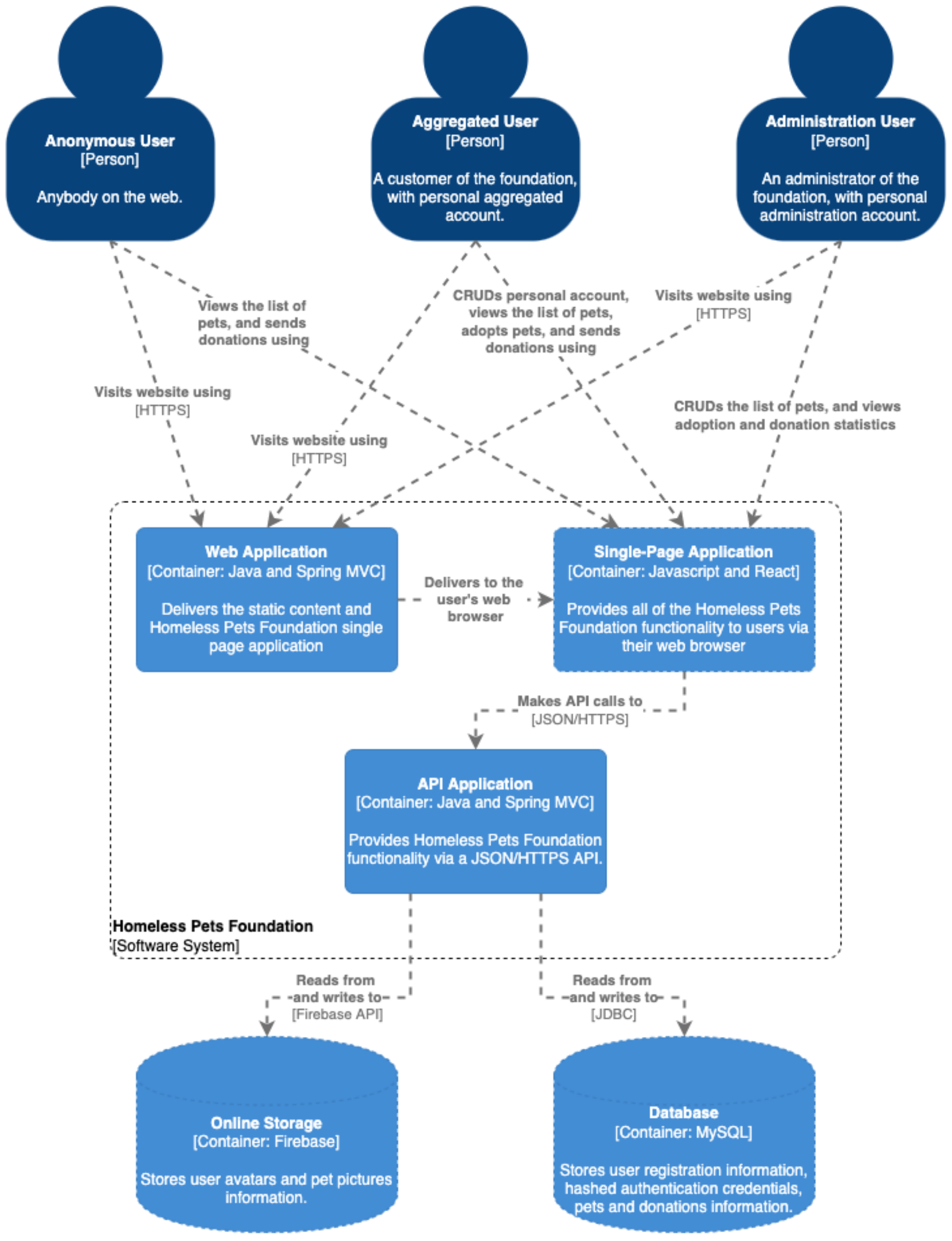

Anonymous User
[Person]
Anybody on the web.
Aggregated User
[Person]
A customer of the foundation, with personal aggregated account.
Administration User
[Person]
An administrator of the foundation, with personal administration account.
Views the list of pets, and sends donations using
CRUDs personal account, views the list of pets, adopts pets, and sends donations using
CRUDs the list of pets, and views adoption and donation statistics
Homeless Pets Foundation
[Software System]
Allows users to view the list of pets, adoption and donation statistics, personal accounts, and aggregated user accounts.
Reads from and writes to
Online Storage
[Container: Firebase]
Stores user avatars and pet pictures information.
Database
[Container: MySQL]
Stores user registration information, hashed authentication credentials, pets and donations information.

System Context diagram for Homeless Pets Foundation

Explanation:

The diagram represents three user roles: Anonymous User (anybody on the web), Aggregated User (customer), and Administration User (administrator). They use the Homeless Pets Foundation software system which accesses two databases when used: Firebase to store pictures and MySQL to store other data.

Containers (C2)

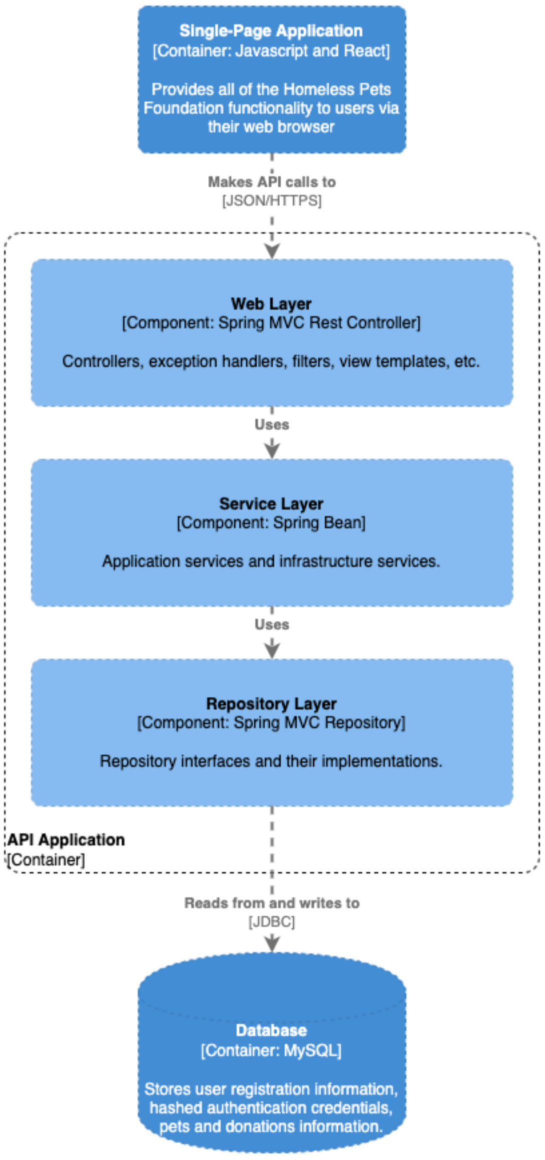

Anonymous User
[Person]
Anybody on the web.
Aggregated User
[Person]
A customer of the foundation, with personal aggregated account.
Administration User
[Person]
An administrator of the foundation, with personal administration account.
Views the list of pets, and sends donations using
Visits website using [HTTPS]
CRUDs personal account, views the list of pets, adopts pets, and sends donations using
Visits website using [HTTPS]
Visits website using [HTTPS]
CRUDs the list of pets, and views adoption and donation statistics
Web Application
[Container: Java and Spring MVC]
Delivers the static content and Homeless Pets Foundation single page application
Delivers to the user's web browser
Single-Page Application
[Container: Javascript and React]
Provides all of the Homeless Pets Foundation functionality to users via their web browser
Makes API calls to [JSON/HTTPS]
API Application
[Container: Java and Spring MVC]
Provides Homeless Pets Foundation functionality via a JSON/HTTPS API.
Homeless Pets Foundation
[Software System]
Reads from and writes to [Firebase API]
Reads from and writes to [JDBC]
Online Storage
[Container: Firebase]
Stores user avatars and pet pictures information.
Database
[Container: MySQL]
Stores user registration information, hashed authentication credentials, pets and donations information.

Container diagram for Homeless Pets Foundation

Explanation:

The diagram represents a deeper look into the Homeless Pets Foundation software system. The users use the Web Application to visit the website, and the Single-Page Application to perform various actions. The software system consists of a Web Application that accesses the API Application through a Single-Page Application, then a MySQL database via JDBC.

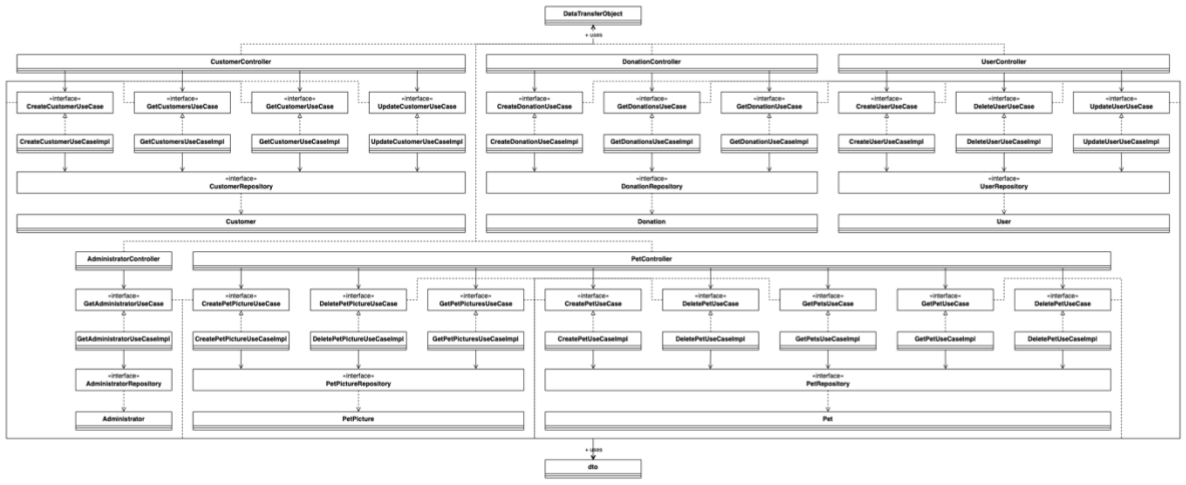
Components (C3)


Single-Page Application
[Container: Javascript and React]
Provides all of the Homeless Pets Foundation functionality to users via their web browser
Makes API calls to
[JSON/HTTPS]
Web Layer
[Component: Spring MVC Rest Controller]
Controllers, exception handlers, filters, view templates, etc.
Uses
Service Layer
[Component: Spring Bean]
Application services and infrastructure services.
Uses
Repository Layer
[Component: Spring MVC Repository]
Repository interfaces and their implementations.
API Application
[Container]
Reads from and writes to
[JDBC]
Database
[Container: MySQL]
Stores user registration information, hashed authentication credentials, pets and donations information.

Component diagram for Homeless Pets Foundation – API Application

Explanation:
The diagram represents a deeper look into an API Application. The users make API calls to it through a Single-Page Application. The API Application container represents a three-layer design that consists of a Controllers, Services, and Repositories. Finally, the Repository accesses the MySQL database via JDBC.

Code (C4)

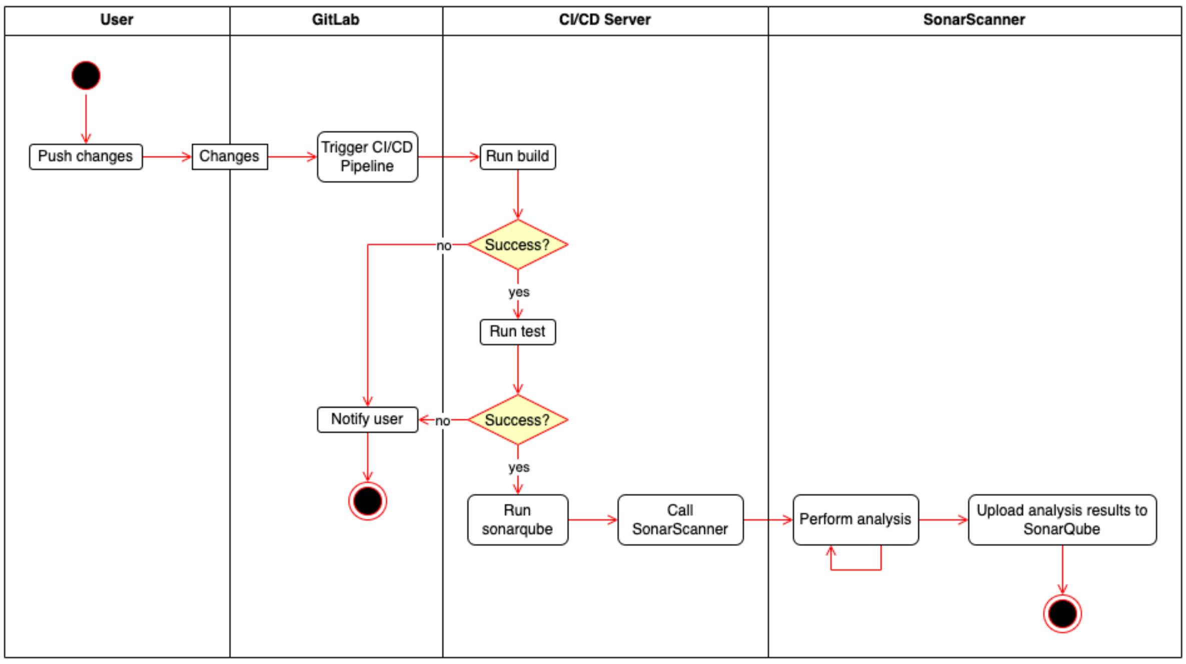

DataTransferObject
+ uses
CustomerController
DonationController
UserController
«interface» CreateCustomerUseCase
«interface» GetCustomersUseCase
«interface» GetCustomerUseCase
«interface» UpdateCustomerUseCase
«interface» CreateDonationUseCase
«interface» GetDonationsUseCase
«interface» GetDonationUseCase
«interface» CreateUserUseCase
«interface» DeleteUserUseCase
«interface» UpdateUserUseCase
CreateCustomerUseCaseImpl
GetCustomersUseCaseImpl
GetCustomerUseCaseImpl
UpdateCustomerUseCaseImpl
CreateDonationUseCaseImpl
GetDonationsUseCaseImpl
GetDonationUseCaseImpl
CreateUserUseCaseImpl
DeleteUserUseCaseImpl
UpdateUserUseCaseImpl
«interface» CustomerRepository
«interface» DonationRepository
«interface» UserRepository
Customer
Donation
User
AdministratorController
PetController
«interface» GetAdministratorUseCase
«interface» CreatePetPictureUseCase
«interface» DeletePetPictureUseCase
«interface» GetPetPicturesUseCase
«interface» CreatePetUseCase
«interface» DeletePetUseCase
«interface» GetPetsUseCase
«interface» GetPetUseCase
«interface» DeletePetUseCase
GetAdministratorUseCaseImpl
CreatePetPictureUseCaseImpl
DeletePetPictureUseCaseImpl
GetPetPicturesUseCaseImpl
CreatePetUseCaseImpl
DeletePetUseCaseImpl
GetPetsUseCaseImpl
GetPetUseCaseImpl
DeletePetUseCaseImpl
«interface» AdministratorRepository
«interface» PetPictureRepository
«interface» PetRepository
Administrator
PetPicture
Pet
+ uses
dto

Code diagram for Homeless Pets Foundation – API Application

Explanation:

The diagram represents an even deeper look into an API Application. It reveals a more in-depth look its' five-layered design which consist of a Controller, UseCase interfaces, UseCase implementations, Repositories, and Entities. The controller accesses the UseCase implementation(s) through the corresponding interface(s), as well as the corresponding dtos. The Repository accesses the corresponding entity to manipulate data, as well as the Controller accesses the DataTransferObject.

CI/CD Activity diagram



CI/CD Activity diagram of GitLab Repository for Homeless Pets Foundation

Explanation:

When the user commits and pushes changes to the Gitlab Repository it triggers the CI/CD pipeline. First, the pipeline builds the project, in case of failure the pipeline stops and notifies the user. If the build succeeds, the pipeline tests the project and stops in case of failure to notify the user. If the tests succeed too, the pipeline calls SonarScanner to analyze pushed changes. SonarScanner continuously performs the analysis and sends the results after completion to SonarQube.