DESIGN DOCUMENT

Dish Dive

SLOBODAN STARCEVIC

Architecture

As the world of software has many languages, frameworks and libraries to pick from it is very important to make an outline of what technologies will be used to build a project. In this chapter we will talk about the constraints given by the semester and the choices made by myself.

Semester Constraints

The semester gives 2 constraints;

1. Use Java for back-end
2. Use JavaScript for front-end

As they are 2 very popular languages and widely used in the industry, they make for not just robust building blocks for any project but also a rich addition to my CV for internships and jobs in the future. Thus, I would not challenge these constraints and gladly accept them.

Own Choices

As the semester already put 2 big constraints on what to use, our choice will be easier to single out the possibilities when it comes to the choice of frameworks for example.

Backend

As we are restricted to Java for the back end, I have done my research on what frameworks are available for the backend;

1. Spring Framework – Most robust and versatile backend frameworks for Java. Has many tools available for DI and IoC for easy configuration and loose coupling of modules, making the application more flexible.

2. Apache Struts – A framework that uses the MVC pattern and tried to go for easy implementation and clean designs. It does seem like there are security in its "Object Graph Navigation Library". I did not dive in too deeply, but the vulnerabilities seem to mention command execution from the outside which seems to be a big vulnerability and turns me away. Besides that, it seems to be less commonly used than Spring and less rich in features.

There seems to be many more framework and all seem to specialise into something else but as I read more and more into it, I kept on seeing more sprint recommendation. Spring seems to be the industry standard when it comes to backend REST APIs. Not only does it have a big history and has been widely used and extended by millions of people but looking at it from an implementation perspective it seems to have everything you need. I like the 'Beans' part of Spring because it makes using dependencies and setting up DI or IoC very easy. Besides that, I was also impressed by the Spring documentation which just gave me more reason to use it.

Frontend

For the frontend we are restricted to JavaScript which will in turn also make it easier to choose a front-end framework. While looking around I came to these frameworks;

1. Vue – Promoting single-file components it seems to be a framework that promises using html CSS and JavaScript bundled together rather than having multiple files. It is a newer framework that is picking up in popularity and is gaining a stronger community. But being

less mature does mean that it doesn’t have as much community support as the other frameworks.

2. React – Most probably the best-known JavaScript framework and it seems also the favoured framework in the industry. The community is huge and just looking around react is always the first recommended framework for web apps it seems. It makes use of .jsx which is a syntax allowing the infusion of JavaScript into HTML.
3. Angular – A mature JavaScript framework with a well sized community which places it at the 2nd spot of most used js framework. Compared to react it seems people describe it as a steeper learning curve and some complexity that can cause first timer a lot of confusion. Angular seems to offer more built-in functionality and compared to react it is less flexible in the use of native JavaScript.

This was difficult to research as most js frameworks have a ton in common and in the end, I need to nitpick things. I ended up being most convinced by react as it seemed to be the most user friendly and its interaction with JavaScript also made it more attractive for me as I also want to learn and improve in js.

C4 modelling

As the C4 diagrams are mostly self-explanatory I will just give a short summary of each diagram and the goal on its level

C1

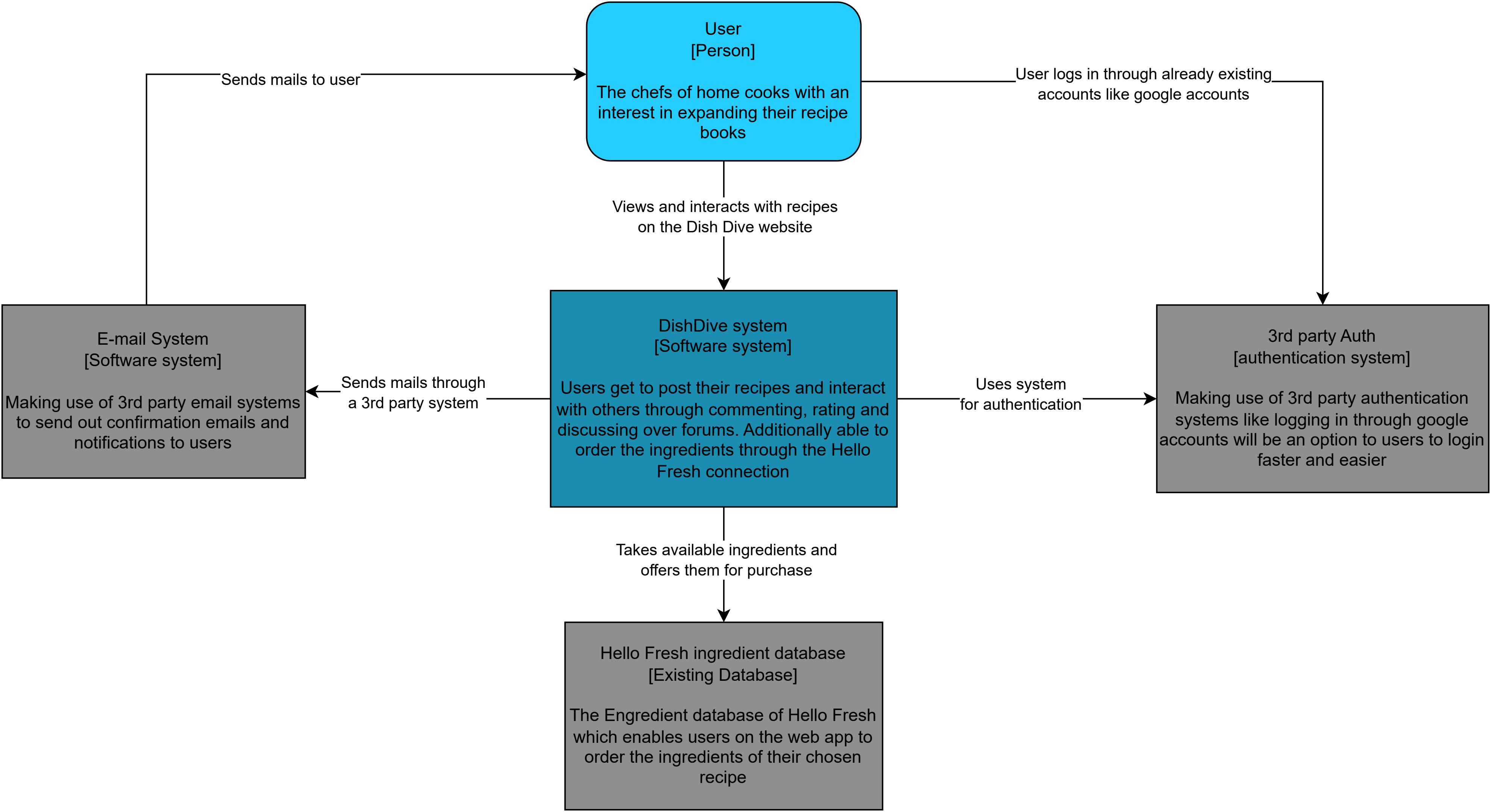
It starts with the user. The user interacts with the Dishdive system. In turn the Dishdive system makes use of external system to fulfil some functionality. The email system is there to email the user confirmation emails and optional notifications, the 3rd party authentication is there for a smoother account creation and login flow and lastly the Hello Fresh Database provides functionality for the buying of ingredients for recipes.

C2

Zooming in on the Dishdive system we see the layering I will be using. The user interacts with the Web application which requests data from the backend application through REST APIs. From here the backend now uses the residual systems to fulfil the needed functionality. Requesting data for our web application will be done to out own database, while the email, authentication and ingredient functionality will be gotten through the 3rd party systems.

C3

Zooming in on the backend application we now see a more detailed setup of the API and service modules. Each flow is made to ensure single responsibility and make sure there is no overlapping while also making sure the routing doesn’t go out of control. My decision on making these controllers and services rests on the thought that the more you split up responsibility the more difficult it becomes to have a good overview of the application. I could for example split the account controller up further into a ‘login controller’ and ‘user controller’ but in my opinion systems shouldn’t be verbose and provide enough grouping of responsibilities to have a balance between “it’s easy to follow and keep track of" and “it makes sense”.



User
[Person]

The chefs of home cooks with an interest in expanding their recipe books

Sends mails to user

User logs in through already existing accounts like google accounts

Views and interacts with recipes on the Dish Dive website

E-mail System
[Software system]

Making use of 3rd party email systems to send out confirmation emails and notifications to users

Sends mails through a 3rd party system

DishDive system
[Software system]

Users get to post their recipes and interact with others through commenting, rating and discussing over forums. Additionally able to order the ingredients through the Hello Fresh connection

Uses system for authentication

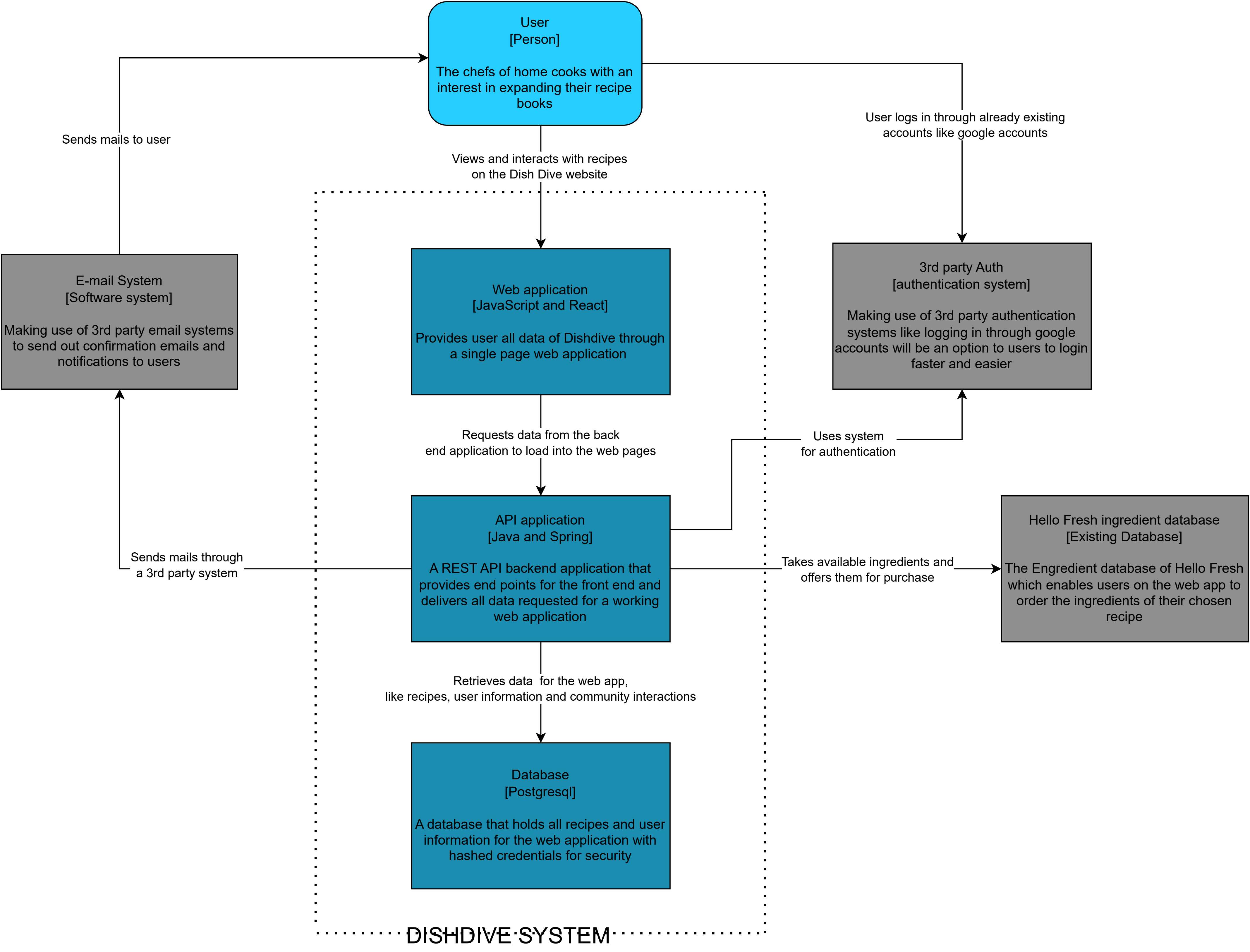
3rd party Auth
[authentication system]

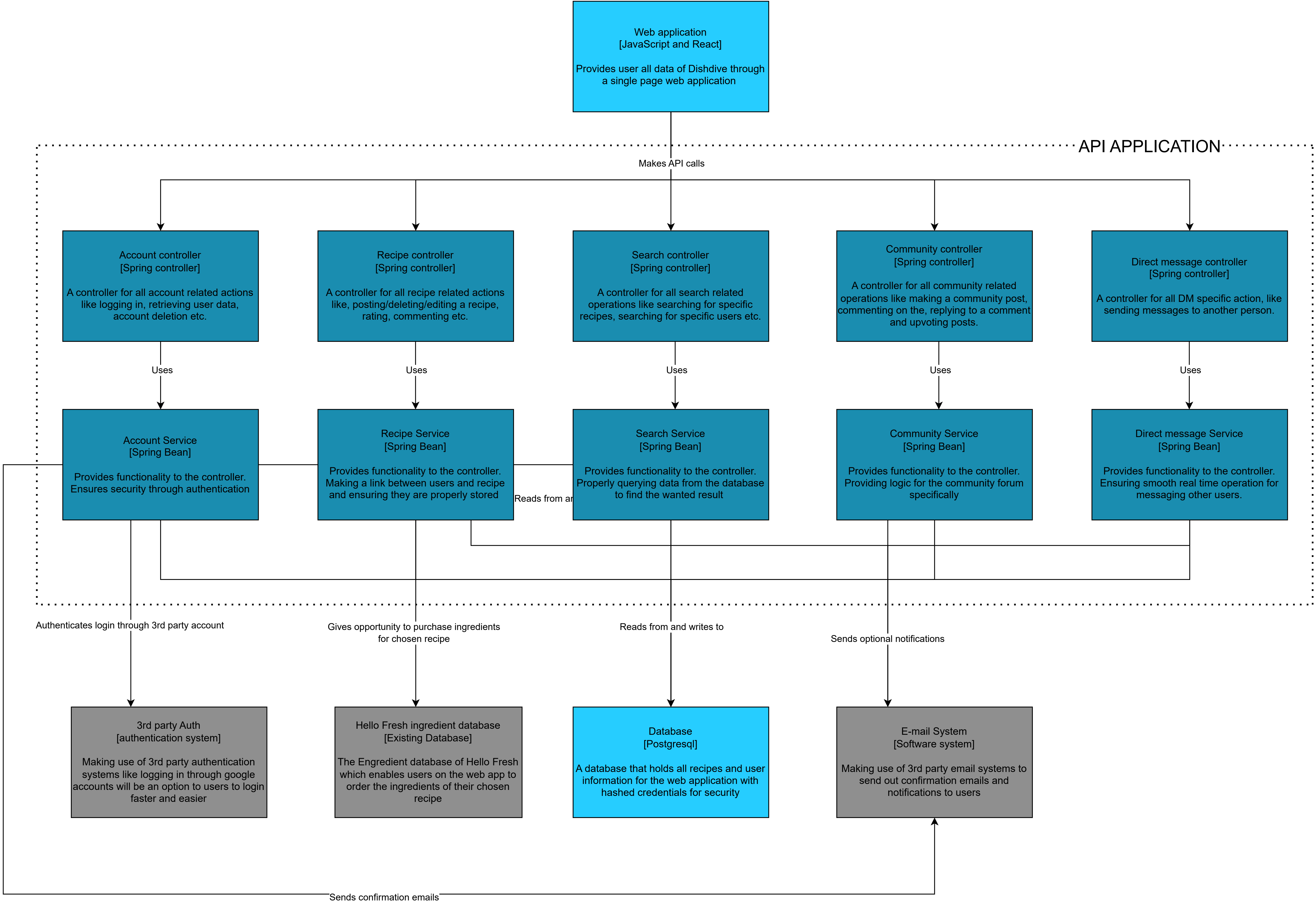
Making use of 3rd party authentication systems like logging in through google accounts will be an option to users to login faster and easier

Takes available ingredients and offers them for purchase

Hello Fresh ingredient database
[Existing Database]

The Engredient database of Hello Fresh which enables users on the web app to order the ingredients of their chosen recipe


User
[Person]
The chefs of home cooks with an interest in expanding their recipe books
Sends mails to user
User logs in through already existing accounts like google accounts
Views and interacts with recipes on the Dish Dive website
E-mail System
[Software system]
Making use of 3rd party email systems to send out confirmation emails and notifications to users
Web application
[JavaScript and React]
Provides user all data of Dishdive through a single page web application
3rd party Auth
[authentication system]
Making use of 3rd party authentication systems like logging in through google accounts will be an option to users to login faster and easier
Requests data from the back end application to load into the web pages
Uses system for authentication
API application
[Java and Spring]
A REST API backend application that provides end points for the front end and delivers all data requested for a working web application
Sends mails through a 3rd party system
Takes available ingredients and offers them for purchase
Hello Fresh ingredient database
[Existing Database]
The Engredient database of Hello Fresh which enables users on the web app to order the ingredients of their chosen recipe
Retrieves data for the web app, like recipes, user information and community interactions
Database
[Postgresql]
A database that holds all recipes and user information for the web application with hashed credentials for security
DISHDIVE SYSTEM



Web application
[JavaScript and React]

Provides user all data of Dishdive through a single page web application

API APPLICATION

Account controller
[Spring controller]

A controller for all account related actions like logging in, retrieving user data, account deletion etc.

Recipe controller
[Spring controller]

A controller for all recipe related actions like, posting/deleting/editing a recipe, rating, commenting etc.

Search controller
[Spring controller]

A controller for all search related operations like searching for specific recipes, searching for specific users etc.

Community controller
[Spring controller]

A controller for all community related operations like making a community post, commenting on the, replying to a comment and upvoting posts.

Direct message controller
[Spring controller]

A controller for all DM specific action, like sending messages to another person.

Account Service
[Spring Bean]

Provides functionality to the controller. Ensures security through authentication

Recipe Service
[Spring Bean]

Provides functionality to the controller. Making a link between users and recipe and ensuring they are properly stored

Search Service
[Spring Bean]

Provides functionality to the controller. Properly querying data from the database to find the wanted result

Community Service
[Spring Bean]

Provides functionality to the controller. Providing logic for the community forum specifically

Direct message Service
[Spring Bean]

Provides functionality to the controller. Ensuring smooth real time operation for messaging other users.

3rd party Auth
[authentication system]

Making use of 3rd party authentication systems like logging in through google accounts will be an option to users to login faster and easier

Hello Fresh ingredient database
[Existing Database]

The Engredient database of Hello Fresh which enables users on the web app to order the ingredients of their chosen recipe

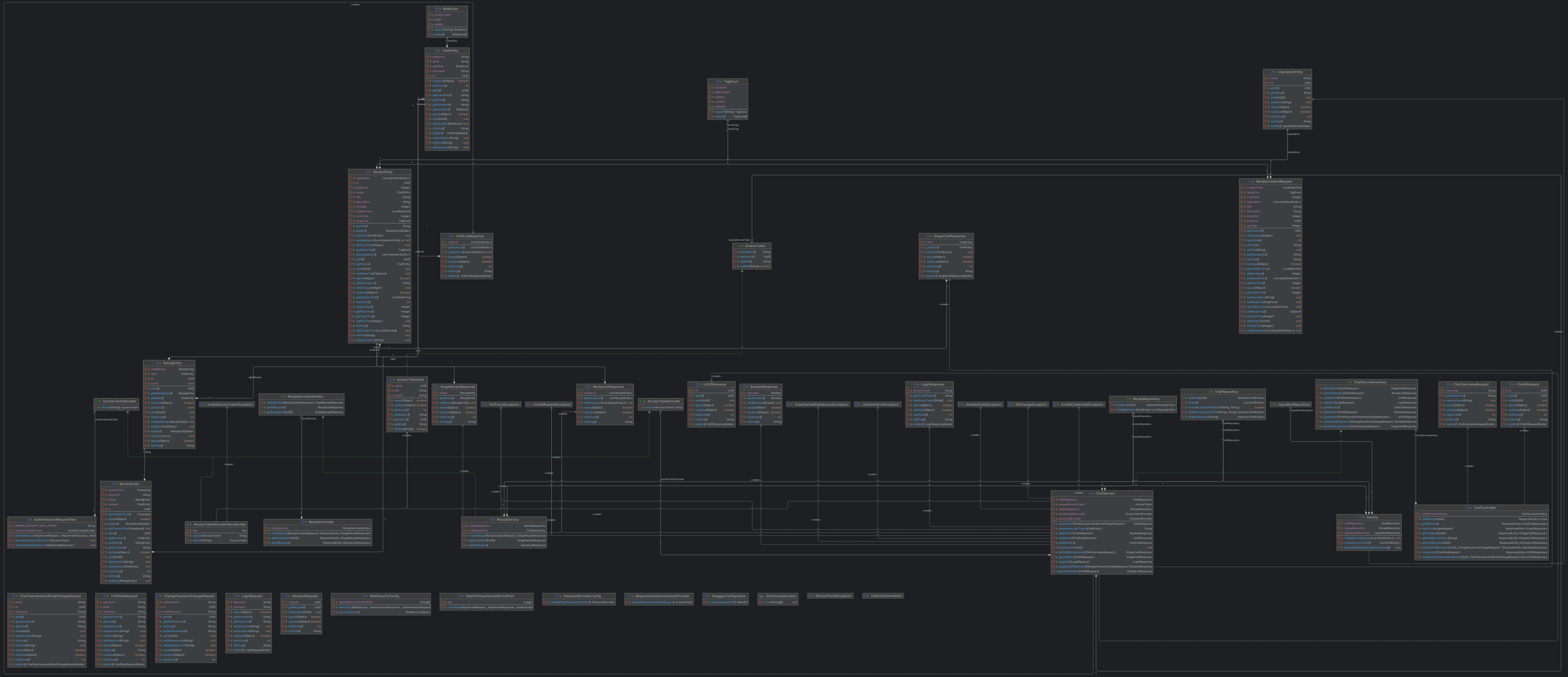
Database
[Postgresql]

A database that holds all recipes and user information for the web application with hashed credentials for security

E-mail System
[Software system]

Making use of 3rd party email systems to send out confirmation emails and notifications to users

Makes API calls
Uses
Uses
Uses
Uses
Uses
Reads from ar
Authenticates login through 3rd party account
Gives opportunity to purchase ingredients for chosen recipe
Reads from and writes to
Sends optional notifications
Sends confirmation emails


«create»
RoleEnum
SUPER_CHEF
CHEF
ADMIN
valueOf(String) RoleEnum
values() RoleEnum[]
userRole
ChefEntity
password String
email String
userRole RoleEnum
username String
id UUID
canEqual(Object) boolean
hashCode() int
getId() UUID
getUsername() String
getEmail() String
getPassword() String
getUserRole() RoleEnum
equals(Object) boolean
setId(UUID) void
setUserRole(RoleEnum) void
toString() String
builder() ChefEntityBuilder
setUsername(String) void
setEmail(String) void
setPassword(String) void
poster
reviewer
TagEnum
DESSERT
BREAKFAST
SNACK
LUNCH
DINNER
valueOf(String) TagEnum
values() TagEnum[]
recipeTag
IngredientEntity
name String
id UUID
getId() UUID
getName() String
setId(UUID) void
setName(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
builder() IngredientEntityBuilder
ingredients
RecipeEntity
ingredients List<IngredientEntity>
id UUID
prepTime Integer
poster ChefEntity
title String
description String
servings Integer
creationTime LocalDateTime
cookTime Integer
recipeTag TagEnum
getTitle() String
builder() RecipeEntityBuilder
setPoster(ChefEntity) void
setIngredients(List<IngredientEntity>) void
setCookTime(Integer) void
getRecipeTag() TagEnum
getIngredients() List<IngredientEntity>
getId() UUID
getPoster() ChefEntity
setId(UUID) void
setRecipeTag(TagEnum) void
equals(Object) boolean
getDescription() String
setServings(Integer) void
canEqual(Object) boolean
getCreationTime() LocalDateTime
hashCode() int
getServings() Integer
getPrepTime() Integer
getCookTime() Integer
setPrepTime(Integer) void
toString() String
setCreationTime(LocalDateTime) void
setTitle(String) void
setDescription(String) void
chefList
ChefListResponse
chefList List<ChefEntity>
getChefList() List<ChefEntity>
setChefList(List<ChefEntity>) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
builder() ChefListResponseBuilder
requestAccessToken
AccessToken
getSubject() String
getUserId() UUID
getRole() String
hasRole(String) boolean
SingleChefResponse
chef ChefEntity
getChef() ChefEntity
setChef(ChefEntity) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
builder() SingleChefResponseBuilder
RecipeCreationRequest
creationTime LocalDateTime
recipeTag TagEnum
cookTime Integer
ingredients List<IngredientEntity>
title String
description String
prepTime Integer
posterId UUID
servings Integer
getPosterId() UUID
setServings(Integer) void
hashCode() int
getTitle() String
setTitle(String) void
getDescription() String
toString() String
canEqual(Object) boolean
getCreationTime() LocalDateTime
getServings() Integer
getIngredients() List<IngredientEntity>
getPrepTime() Integer
equals(Object) boolean
getCookTime() Integer
setDescription(String) void
setRecipeTag(TagEnum) void
setCreationTime(LocalDateTime) void
getRecipeTag() TagEnum
setCookTime(Integer) void
setPosterId(UUID) void
setPrepTime(Integer) void
setIngredients(List<IngredientEntity>) void
recipe
recipeList
chef
rater
ratedRecipe
RatingEntity
ratedRecipe RecipeEntity
rater ChefEntity
id UUID
score short
getId() UUID
getRatedRecipe() RecipeEntity
getRater() ChefEntity
canEqual(Object) boolean
getScore() short
setId(UUID) void
hashCode() int
setRatedRecipe(RecipeEntity) void
setRater(ChefEntity) void
builder() RatingEntityBuilder
setScore(short) void
equals(Object) boolean
toString() String
rating
AccessTokenDecoder
decode(String) AccessToken
accessTokenDecoder
InvalidAccessTokenException
RecipeServiceInterface
createRecipe(RecipeCreationRequest) SingleRecipeResponse
getAllRecipes() RecipeListResponse
getRecipeById(UUID) SingleRecipeResponse
recipeService
AccessTokenImpl
userId UUID
role String
subject String
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
getSubject() String
getUserId() UUID
getRole() String
hasRole(String) boolean
SingleRecipeResponse
recipe RecipeEntity
getRecipe() RecipeEntity
setRecipe(RecipeEntity) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
NotFoundException
InvalidRequestException
RecipeListResponse
recipeList List<RecipeEntity>
getRecipeList() List<RecipeEntity>
setRecipeList(List<RecipeEntity>) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
AccessTokenEncoder
encode(AccessToken) String
accessTokenEncoder
UUIDResponse
id UUID
getId() UUID
setId(UUID) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
builder() UUIDResponseBuilder
BooleanResponse
aBoolean Boolean
getABoolean() Boolean
setABoolean(Boolean) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
UnauthorizedDataAccessException
UnknownErrorException
LoginResponse
accessToken String
getAccessToken() String
setAccessToken(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
builder() LoginResponseBuilder
AlreadyExistsException
NoChangeException
InvalidCredentialsException
RecipeRepository
findById(UUID) Optional<RecipeEntity>
findAllByPoster(ChefEntity) List<RecipeEntity>
recipeRepository
ChefRepository
findById(UUID) Optional<ChefEntity>
findAll() List<ChefEntity>
existsByUsernameOrEmail(String, String) boolean
findByUsernameOrEmail(String, String) Optional<ChefEntity>
findByUsername(String) Optional<ChefEntity>
chefRepository
IngredientRepository
ingredientRepository
ChefServiceInterface
getChefById(ChefIdRequest) SingleChefResponse
getChefRecipes(ChefIdRequest) RecipeListResponse
createChef(ChefDataRequest) UUIDResponse
loginChef(LoginRequest) LoginResponse
getAllChefs() ChefListResponse
deleteChef(ChefIdRequest) BooleanResponse
updateChef(ChefUsernameAndEmailChangeRequest) UUIDResponse
updateChefPassword(ChangePasswordChangeRequest) BooleanResponse
getChefByUsername(ChefUsernameRequest) SingleChefResponse
chefServiceInterface
ChefUsernameRequest
username String
getUsername() String
setUsername(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
builder() ChefUsernameRequestBuilder
ChefIdRequest
id UUID
getId() UUID
setId(UUID) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
builder() ChefIdRequestBuilder
AuthenticationRequestFilter
SPRING_SECURITY_ROLE_PREFIX String
accessTokenDecoder AccessTokenDecoder
doFilterInternal(HttpServletRequest , HttpServletResponse , Filter
setupSpringSecurityContext(AccessToken) void
sendAuthenticationError(HttpServletResponse) void
ReviewEntity
creationTime Timestamp
comment String
rating RatingEntity
reviewer ChefEntity
id UUID
getCreationTime() Timestamp
equals(Object) boolean
builder() ReviewEntityBuilder
setCreationTime(Timestamp) void
getId() UUID
getReviewer() ChefEntity
getRating() RatingEntity
getComment() String
canEqual(Object) boolean
setId(UUID) void
setComment(String) void
setReviewer(ChefEntity) void
hashCode() int
toString() String
setRating(RatingEntity) void
AccessTokenEncoderDecoderImpl
key Key
encode(AccessToken) String
decode(String) AccessToken
RecipeController
recipeService RecipeServiceInterface
createRecipe(RecipeCreationRequest) ResponseEntity <SingleRecipeResponse>
getRecipesById(UUID) ResponseEntity <SingleRecipeResponse>
getAllRecipes() ResponseEntity <RecipeListResponse>
RecipeService
recipeRepository RecipeRepository
chefRepository ChefRepository
createRecipe(RecipeCreationRequest) SingleRecipeResponse
getRecipeById(UUID) SingleRecipeResponse
getAllRecipes() RecipeListResponse
ChefService
chefRepository ChefRepository
requestAccessToken AccessToken
recipeRepository RecipeRepository
accessTokenEncoder AccessTokenEncoder
passwordEncoder PasswordEncoder
updateChef(ChefUsernameAndEmailChangeRequest) UUIDResponse
generateAccessToken(ChefEntity) String
deleteChef(ChefIdRequest) BooleanResponse
createChef(ChefDataRequest) UUIDResponse
getAllChefs() ChefListResponse
isAuthorized(UUID) void
getChefByUsername(ChefUsernameRequest) SingleChefResponse
getChefById(ChefIdRequest) SingleChefResponse
loginChef(LoginRequest) LoginResponse
updateChefPassword(ChangePasswordChangeRequest) BooleanResponse
getChefRecipes(ChefIdRequest) RecipeListResponse
Dummy
chefRepository ChefRepository
recipeRepository RecipeRepository
ingredientRepository IngredientRepository
createDummyRecipes(List<ChefEntity>) void
createDummyChefs() List<ChefEntity>
populateDatabaseInitialDummyData() void
ChefController
chefServiceInterface ChefServiceInterface
deleteChef(UUID) ResponseEntity <Void>
getAllChefs() ResponseEntity <ChefListResponse>
loginChef(LoginRequest) ResponseEntity <LoginResponse>
getChefById(UUID) ResponseEntity <SingleChefResponse>
getChefByUsername(String) ResponseEntity <SingleChefResponse>
getChefRecipes(UUID) ResponseEntity <RecipeListResponse>
updateChefPassword(UUID, ChangePasswordChangeRequest) ResponseEntity <BooleanResponse>
createChef(ChefDataRequest) ResponseEntity <UUIDResponse>
updateChefUsernameAndEmail(UUID, ChefUsernameAndEmailChangeRequest) tity <UUIDResponse>
ChefUsernameAndEmailChangeRequest
email String
id UUID
username String
getId() UUID
getUsername() String
getEmail() String
setId(UUID) void
setUsername(String) void
toString() String
setEmail(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
builder() ChefUsernameAndEmailChangeRequestBuilder
ChefDataRequest
password String
email String
username String
getUsername() String
getEmail() String
getPassword() String
setUsername(String) void
setEmail(String) void
setPassword(String) void
equals(Object) boolean
toString() String
canEqual(Object) boolean
hashCode() int
builder() ChefDataRequestBuilder
ChangePasswordChangeRequest
oldPassword String
id UUID
newPassword String
getId() UUID
getOldPassword() String
toString() String
getNewPassword() String
setId(UUID) void
setOldPassword(String) void
setNewPassword(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
LoginRequest
password String
username String
equals(Object) boolean
getUsername() String
getPassword() String
setUsername(String) void
setPassword(String) void
canEqual(Object) boolean
hashCode() int
toString() String
builder() LoginRequestBuilder
RecipeIdRequest
recipeId UUID
getRecipeId() UUID
setRecipeId(UUID) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
WebSecurityConfig
SWAGGER_UI_RESOURCES String[]
filterChain(HttpSecurity , AuthenticationEntryPoint , AuthenticationRequest
corsConfigurer() WebMvcConfigurer
Http401UnauthorizedEntryPoint
log Logger
commence(HttpServletRequest , HttpServletResponse , Authentication
PasswordEncoderConfig
createBCryptPasswordEncoder() PasswordEncoder
RequestAuthenticatedUserProvider
getAuthenticatedUserInRequest () AccessToken
SwaggerConfiguration
customizeOpenAPI() OpenAPI
DishdiveApplication
main(String[]) void
NoUserFoundException
ValidUuidAnnotation

CI Flow Diagram

On the next page you will find the CI diagram of my CI/CD pipeline. You will notice that I split up the phases each having their own little 'tasks' which are then colored.

The colors represent the following;

- Green: Local internal which mean within the project environment, think of project files etc.
- Red: Local external which means it is locally on the running machine but is outside of the project environment, think of sonar running locally on port 9000.
- Purple: Remote which means it is completely removed from the running machine and relies on external systems, think of docker images hosted on docker-hub.

Pre

With the pre-phase I want to emphasize on what is done before the pipeline can properly start building the project. Here tasks are done like the connection of the runner, pulling the right environment versions from docker and setting the gradle home variable.

Build

In the build stage we go step by step through how the pipeline goes about building the project and caching the needed items for further steps. Caching improves performance and makes sure same data is used where needed.

Test

The test phase is all about building the project but with an emphasis on the tests, this is done through the test property in the build.gradle file.

Test: Sonar

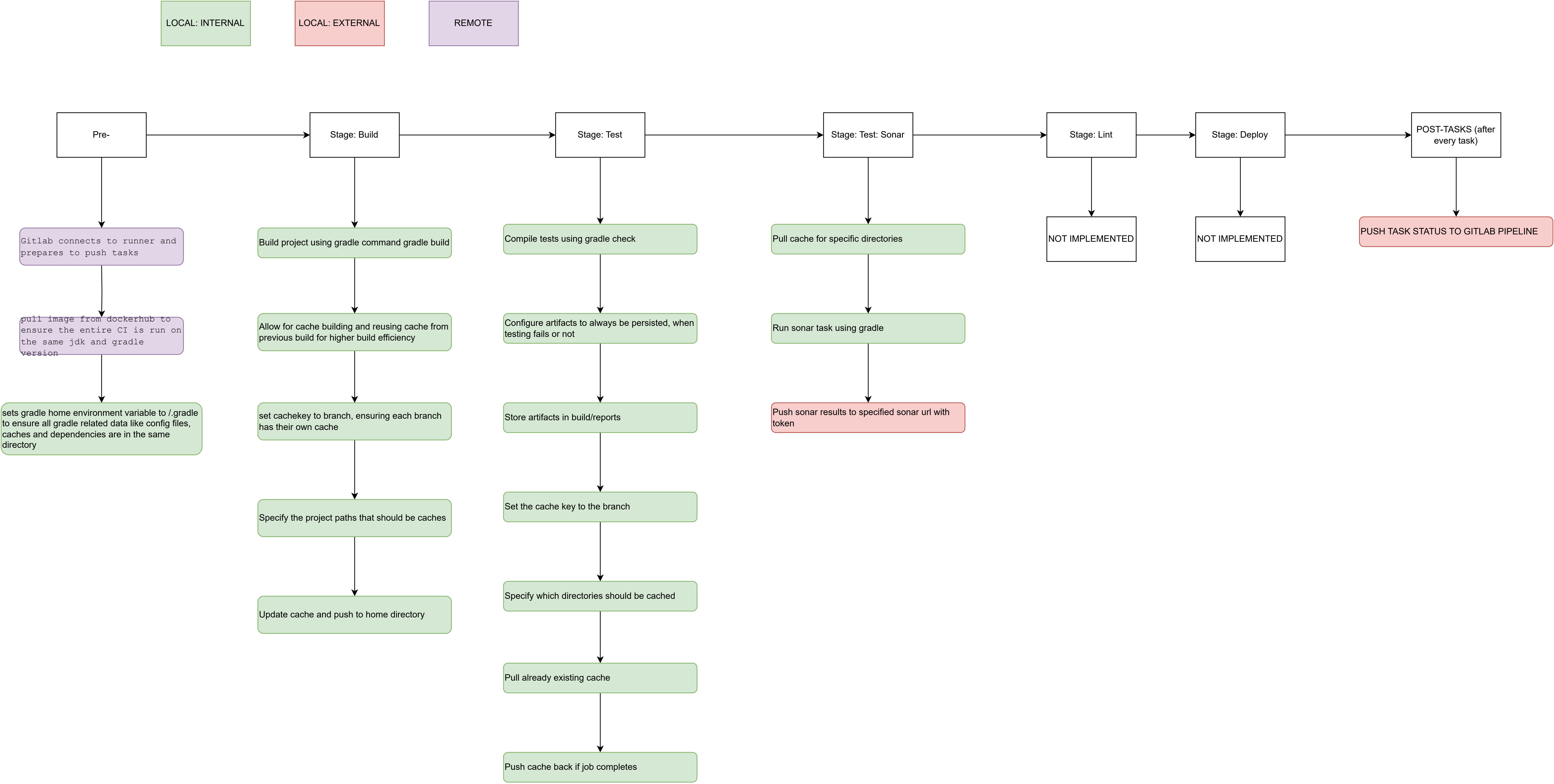
Now that the tests have been run it is time for sonar to compile the test data and push it to the locally running sonar container where then a report is shown of this stage.

Lint & Deploy

These stages have not been implemented yet. Linter is not that necessary as sonar acts as an advanced linter.

Post

This is put at the end of the pipeline, but it really happens during and after every single task, which is also written in the diagram. As each task is being taken care of by the runner it will continuously report back to gitlab where you can see the status of each task as the pipeline is going.



🟩 LOCAL: INTERNAL

🟥 LOCAL: EXTERNAL

🟪 REMOTE

Pre-

- Gitlab connects to runner and prepares to push tasks
- pull image from dockerhub to ensure the entire CI is run on the same jdk and gradle version
- sets gradle home environment variable to /.gradle to ensure all gradle related data like config files, caches and dependencies are in the same directory

Stage: Build

- Build project using gradle command gradle build
- Allow for cache building and reusing cache from previous build for higher build efficiency
- set cachekey to branch, ensuring each branch has their own cache
- Specify the project paths that should be caches
- Update cache and push to home directory

Stage: Test

- Compile tests using gradle check
- Configure artifacts to always be persisted, when testing fails or not
- Store artifacts in build/reports
- Set the cache key to the branch
- Specify which directories should be cached
- Pull already existing cache
- Push cache back if job completes

Stage: Test: Sonar

- Pull cache for specific directories
- Run sonar task using gradle
- Push sonar results to specified sonar url with token

Stage: Lint

- NOT IMPLEMENTED

Stage: Deploy

- NOT IMPLEMENTED

POST-TASKS (after every task)

- PUSH TASK STATUS TO GITLAB PIPELINE