FoodGram

Analysis and Design Document

Student: Jakab Gyöngyi Anikó

**Group:30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 23/04/2019 | 1.0 | First Version | Jakab Gyongyi Aniko |
| 07/05/2019 | 2.0 | Iteration 1.2 | Jakab Gyongyi Aniko |
| 03/06/2019 | 3.0 | Iteration 2 + correcting iteration 1.2 | Jakab Gyongyi Aniko |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 6

2.3 Component and Deployment Diagrams 7

III. Elaboration – Iteration 1.2 7

1. Design Model 7

1.1 Dynamic Behavior 7

1.2 Class Design 8

2. Data Model 9

3. Unit Testing 10

IV. Elaboration – Iteration 2 11

1. Architectural Design Refinement 11

2. Design Model Refinement 12

V. Construction and Transition 13

1. System Testing 13

2. Future improvements 13

VI. Bibliography 14

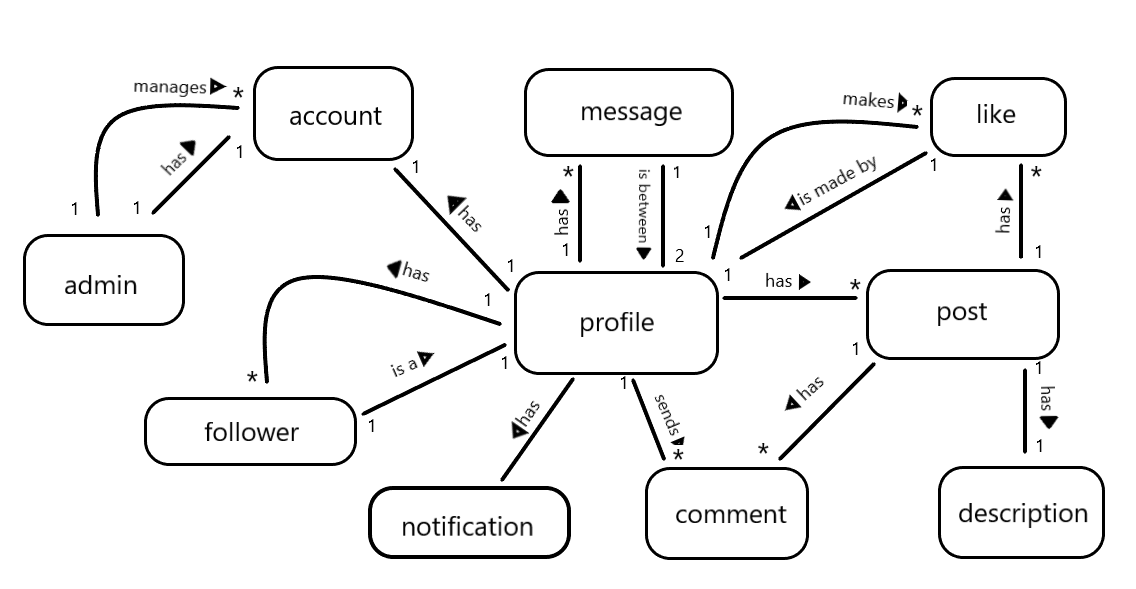
# Project Specification

Foodgram is a web application for a community of people that love making and sharing food recipes. This application will make possible for them to share their taste with others or even communicate with their followers. Every post of an individual will contain an image of a specific food or recipe and a short description of how it is made. Description of the post may be updated in the future, and the user’s information / bio as well.

# Elaboration – Iteration 1.1

# Domain Model

A domain model is a conceptual model of the domain that incorporates both behavior and data. In our case domain elements and the relations between them are described in the following diagram. Elements like username and password are supposed to be incapsulated in the concept called account, and images are part of the concept labeled as post.



# Architectural Design

## Conceptual Architecture

Layered architectural pattern can be used to structure applications that can be decomposed on several sub-application or sub-tasks, each of them representing a different level of abstraction. Each layer provides services for layer above, in this manner, the logic of structuring has a linear flow. As we discussed on previous projects, the layer architectural pattern has 3 main layers:

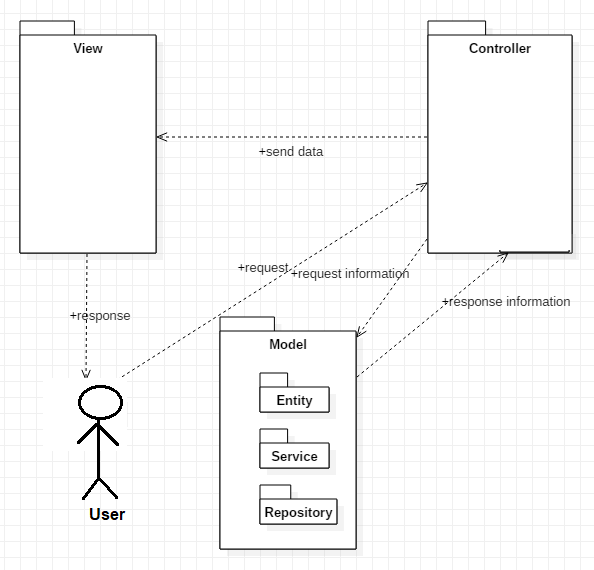
1. Presentation Layer
2. Business Logic Layer or Domain Logic Layer
3. Data Access Layer

Presentation Layer has the highest abstraction level in this system, because everything that requires must be previously implemented and tested to work properly. The presentation Layer is often implemented using another architectural pattern like MVC, MVVM, MP or other derivates of the MVC pattern. The Business Logic Layer consist of classes and interfaces that represent the domain logic and functionality of the application, basically if the application would have no database connection or GUI this would be the only existing layer. Finally, the Data Access Layer which in our case represent only entities and data access object, but in other cases may also contain services from other applications or web services.

Model-View-Controller architectural pattern follows an elementary idea – we must separate the responsibilities in any application on the following basis:

* **Model:** Entities of the application that are represented in GUI.
* View: Presents the data to the user whenever asked for. View represents the visualization of the data that model contains.
* Controller: Entertains user requests and fetch necessary resources. Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

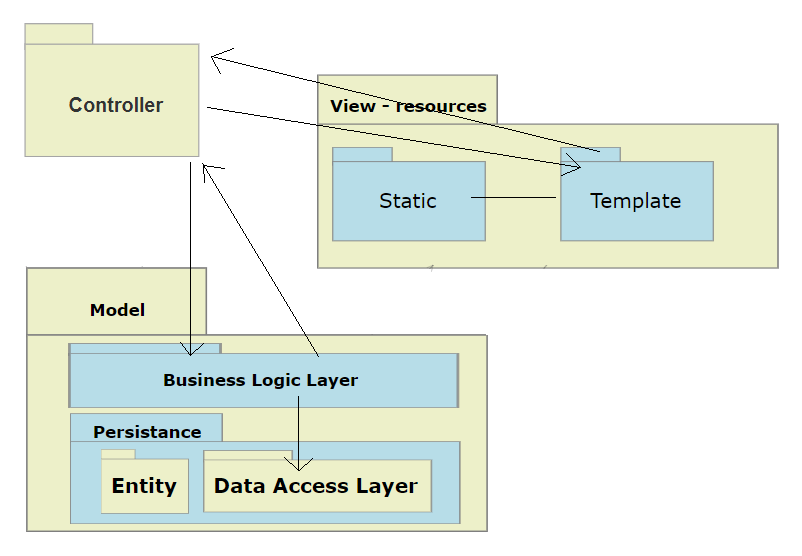
In our implementation the MVC pattern is suited for the Spring Framework, meaning that the concept of model encapsulates both data access and business logic layer, views are represented by HTML pages and the controllers are the connection objects between the main logic and the views.



The motivation for using the specified architectural patterns is to logically structure the application so that intermediar implementation steps and further developments are easily made. In a Java project using Spring framework the first step would be to implement and test the elements of the data access layer, which in our case is represented by the Persistence Layer or Package. After this step the database of the project is automatically generated and the so called, implementation and coding may start. Step by step the Services, which represent the domain logic are added, so the HTML pages the view and the controllers, the mediator between domain logic and presentation are constructed.

## Package Design

The Foodgram application in a Java application constructed with the support of Spring Framework. It is constructed in a Spring web application style using Thymeleaf template engine. This means the MVC architectural pattern is the main pattern on which the application is based. The controller package will contain the controllers for login and signup, administrator and user operation managers. The information from the users side is collected into and displayed on view element, which in out case are stored under resources directory, and has two main packages, static and templates. The templates package contains the HTML views that are used for data display and collection, and the static package contains javascript and css elements, to manage and style the data. The model package is the projects core. It contains services (domain logic), entities and data access object (repositories).



## Component and Deployment Diagrams

A UML component diagram is a physical representation of class diagram. It is used in modeling physical aspects of object-oriented systems that are used for visualizing, specifying and documenting component-based systems, also for constructing executable systems through forward and reverse engineering.

Deployment diagrams show their physical configurations of software and hardware. Deployment Diagram show the structure of the run-time system, capture the hardware that will be used to implement the system and the links between different items of hardware.

# 

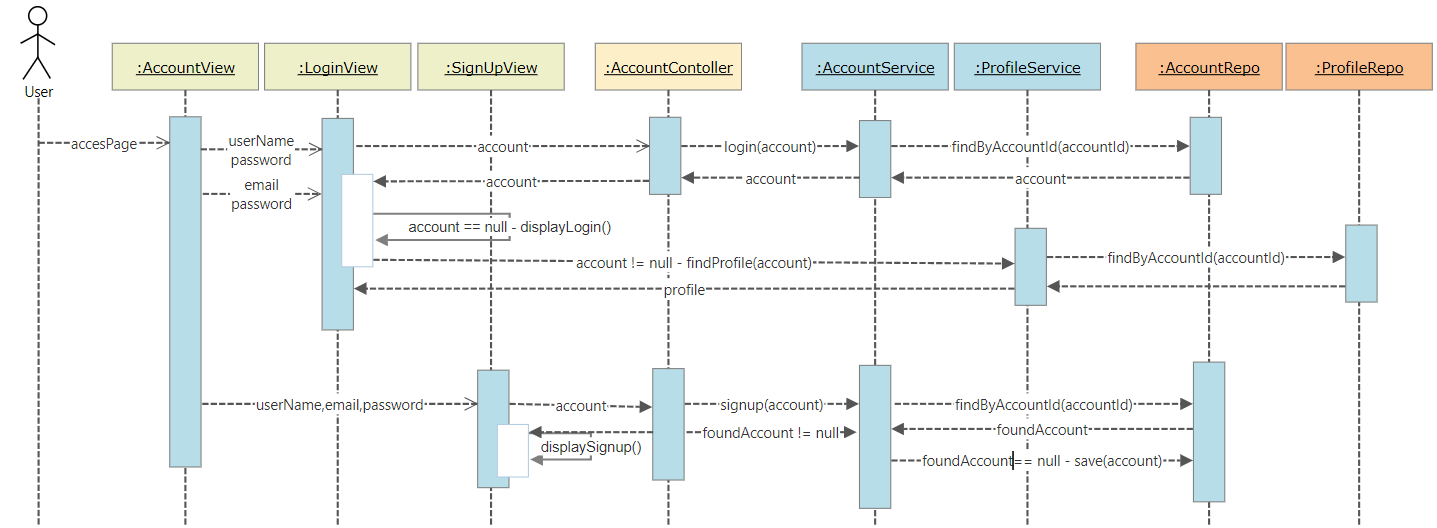
# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

Communication diagram is a kind of UML interaction diagram which shows interactions between objects using sequenced messages in a free-form arrangement. Sequence diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of the collaboration. Sequence diagrams are time focused and they show the order in which the interactions are made (vertical axis represent an object time line).

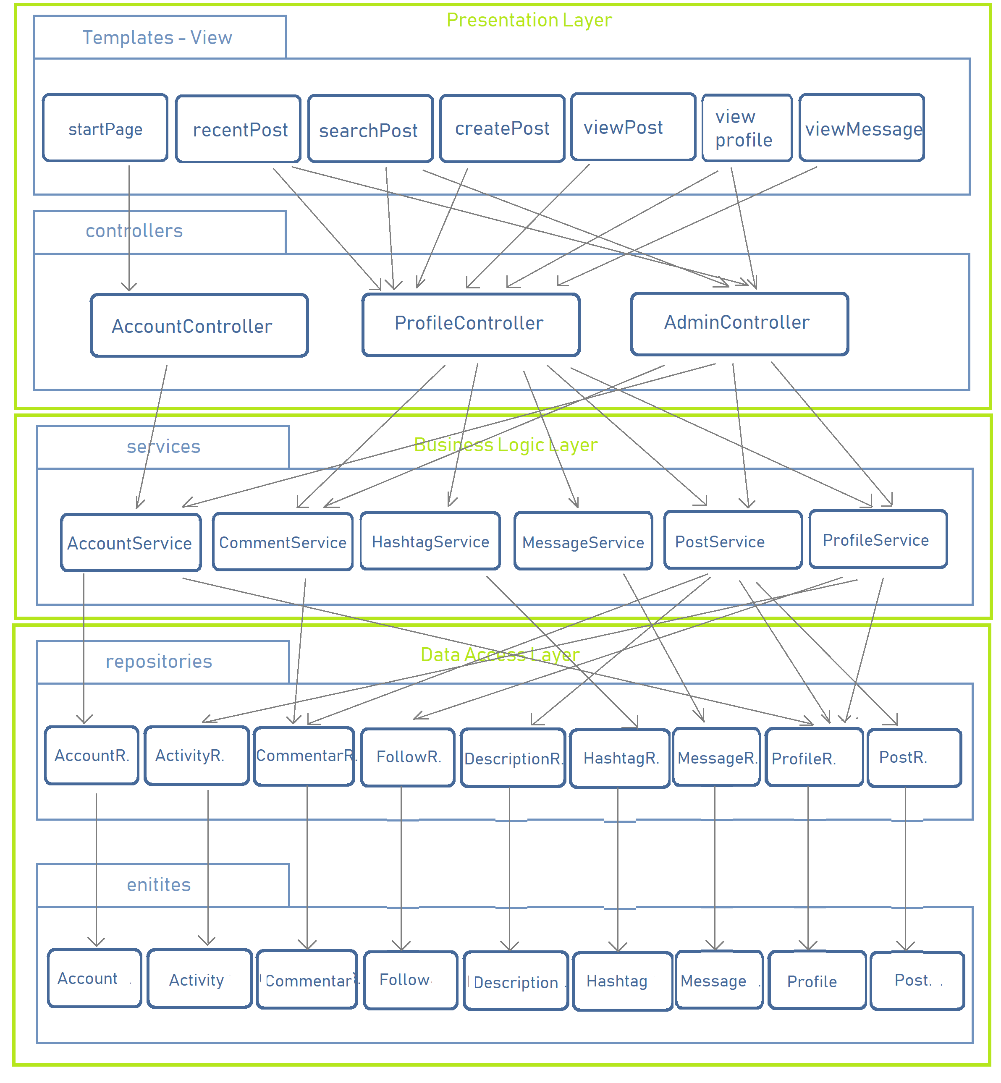
In the following section it will be presented the account management diagram, meaning login and signup operations and a presentation of communication between service – business logic and data access objects.



## 

## Class Design

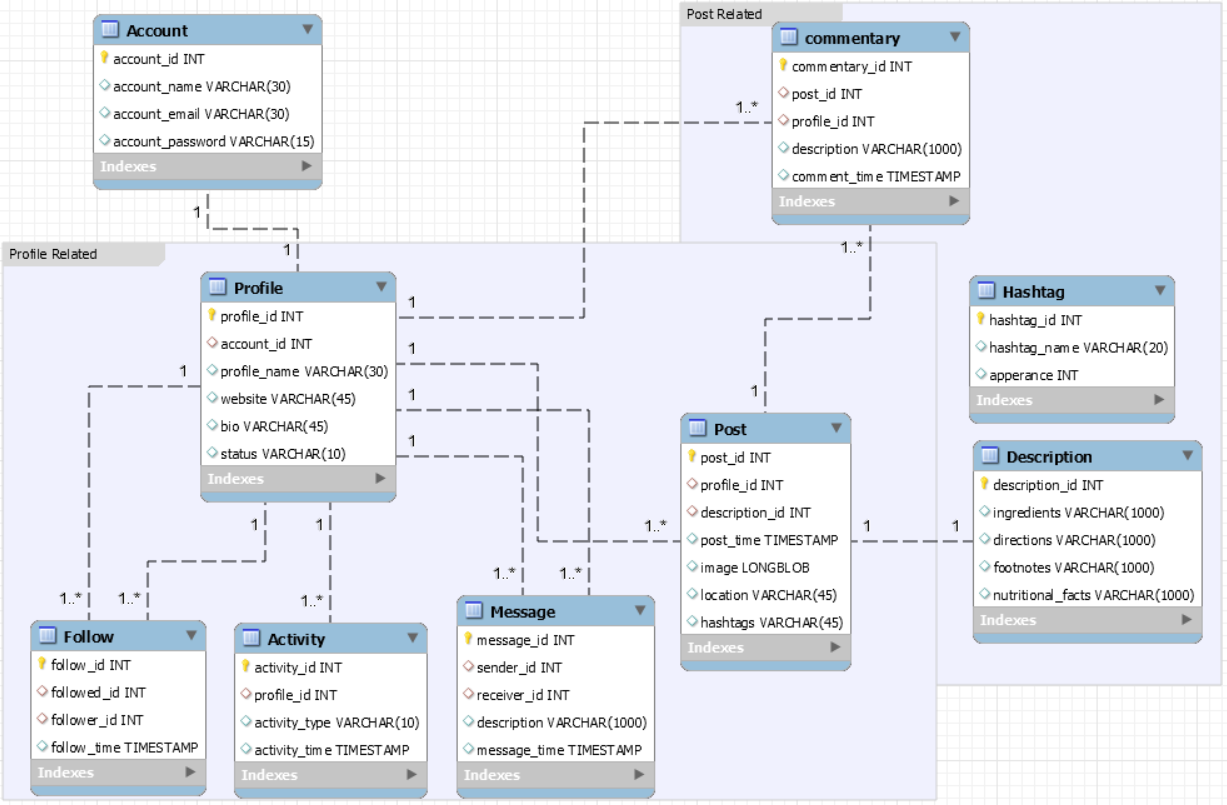
Design of this project is structured using Layers and Model-View-Controller architectural patterns. Considering this, it is needed a separation on layers of classes and their relations.



# Data Model

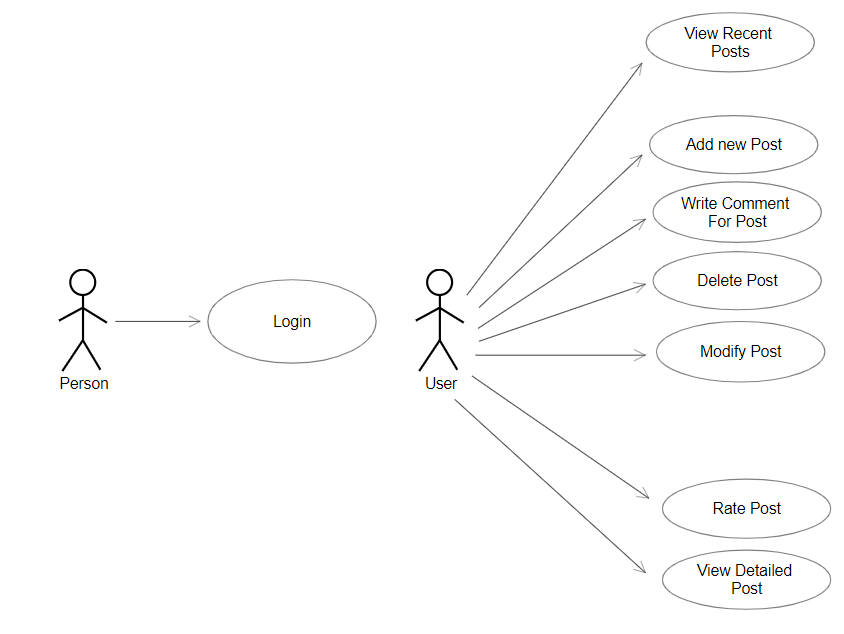
The Data Model represent the elements of the persistence layer, the so called entities and the relations between them. These concepts were extracted based on the domain model of the project, in a way that only main domain elements were selected to be entities, and the number of relations, or their types might be changed from the original idea.

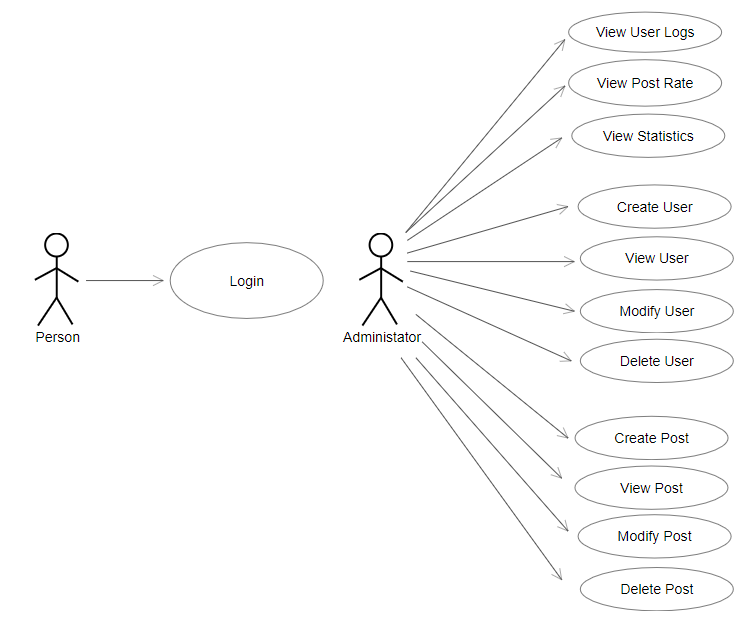
The Relational Database Diagram of the System using UML representation



# Unit Testing

Testing in case of this project is made using Junit 4, which is integrated in Spring. In the main testing class we injected all repositories and tested them individually for each main operation: save, delete and find by different attribute names. This testing unit is also including some use case testing, for example login and signup operation. After all repositories are tested and services are implemented alongside views and their associated controllers, use case tested could me made though the user interface. In some cases the user interface implementation has not been made with best practice style, in these cases unit testing is made on the service elements too, to prove the correct functionality of the system.

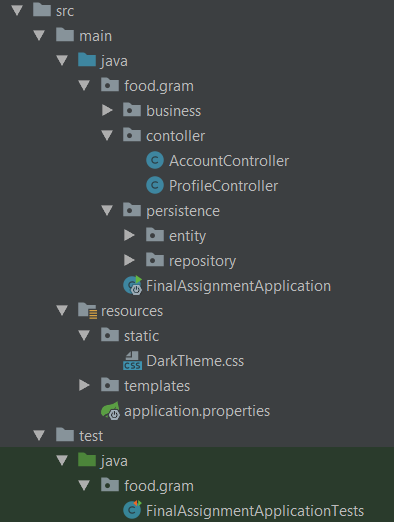
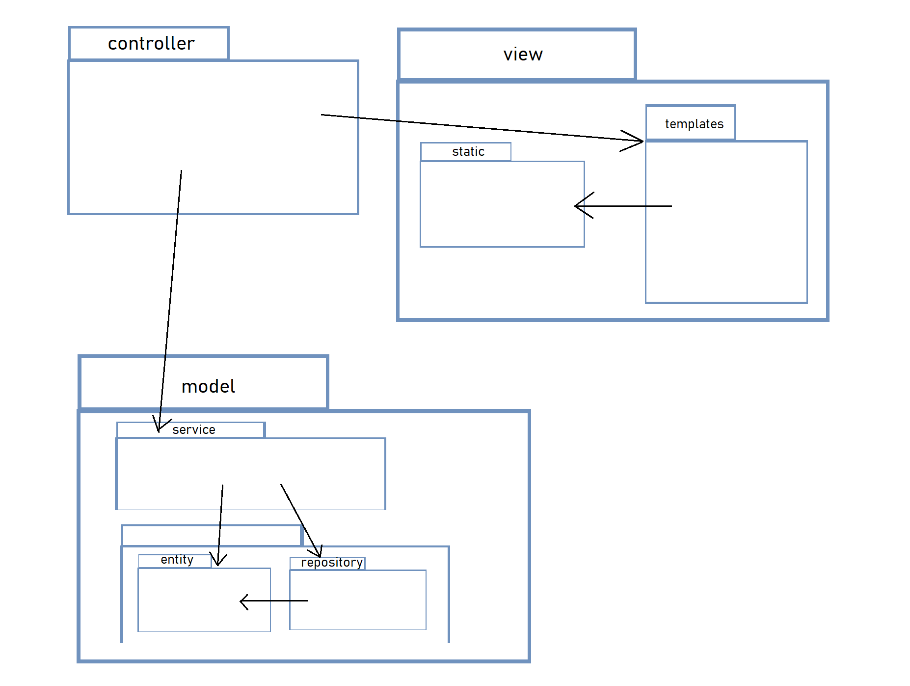




# Elaboration – Iteration 2

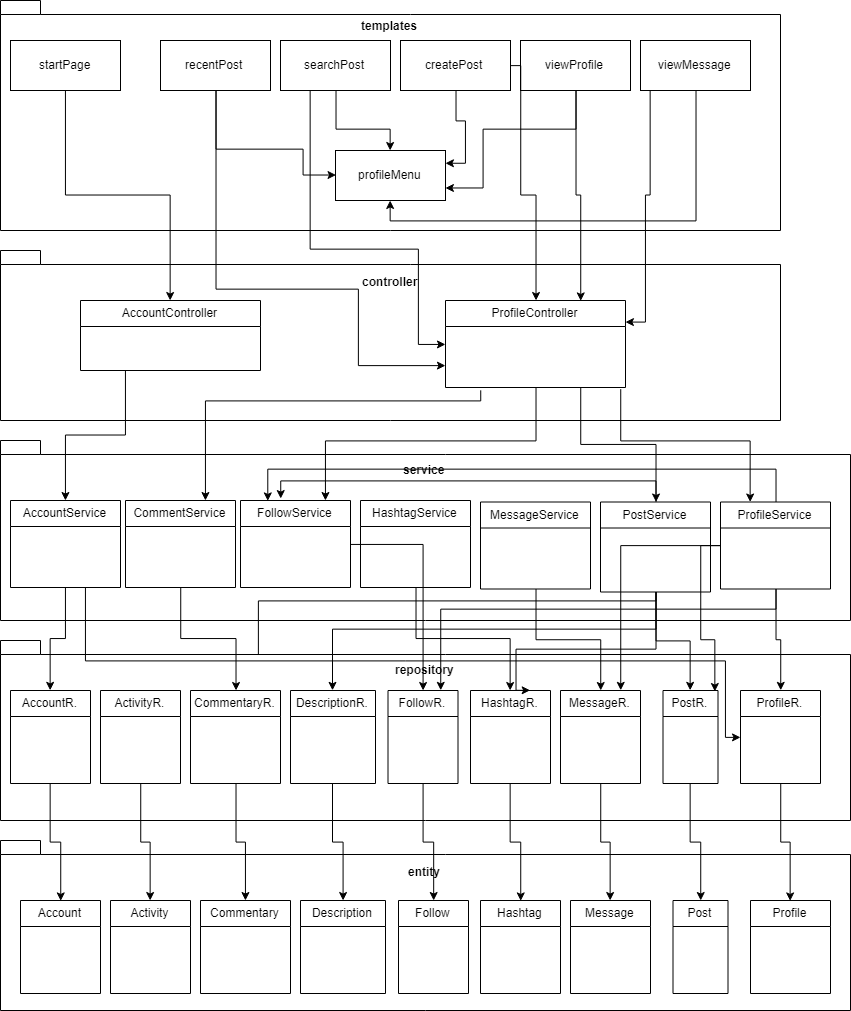
# Architectural Design Refinement

As always, some elements of the project change after the whole implementation of the application is finished. Below the refined package design diagram if resented. Observe that package relations do not change after implementation, but element of specific packages does, for example there are some services, that are implemented using another service, in order to reduce code duplication and the maintain the SOLID principles, end eliminate cyclic dependencies between elements.



# Design Model Refinement

As I mentioned in the previous section, in order to keep the SOLID principles intact in case of this project, there where some changes that I’ve made in the structure of specific packages. For example I’ve made changes in the service package of the application, because there were some services, that could be reused to access their methods and functionality.



# Construction and Transition

# System Testing

Integration Testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

During development, I integrated every unit or module when it was finished and tested it can be integrated with the others. After I finished the project, I integrated all component and test them manually by doing a full flow of the application. Basically, Integration testing was done after unit testing, and it revealed that every case scenario works properly and fits the description provided in the Use Case Models document.

The following usual and particular cases have been tested:

* User login with username and password OR email address and password
* User sign up with username, email address and password (the account is created based on these data, and the profile is generated automatically using the username) with minimal validation on the “front – end” side
* Creation of new post for a specific profile
* Update or delete of previously create post that belongs to the currently online profile
* Recent post visualization for a specific profile
* Search post visualization for a specific profile
* Profile visualization for a specific profile alongside the possibility of account and profile data update
* Message sub-system testing - conversation search and send message for another profile, start new conversation after profile follow

# Future improvements

Future improvement could be the following ones:

* The application should support real account authentication using Spring Security, so that specific operations could be made only when the user in online
* The application should implement a notification system, so the users login, logout actions are tracked, and also visualize the recent notifications like if someone likes your post, commented on it
* The application should support Spring Mail, so that account related update, like password or username change actions are made securely via email address check
* The application should support multiple image upload in case of a post
* The application should support intelligent recommendation system, so in the search section of the application, a user could search after multiple object types like, hashtags or elements of a word, location, not just profiles and posts of a specific profile

# Bibliography

<https://o7planning.org/en/11545/spring-boot-and-thymeleaf-tutorial>

<https://o7planning.org/en/10683/create-a-shopping-cart-web-application-with-spring-boot-hibernate>

<https://stackoverflow.com/questions/20193138/very-simple-spring-mvc-button-click>

<https://www.baeldung.com/java-year-month-day>

<https://medium.com/@RogelioOrts/layered-architecture-spring-boot-af7dc071d2b5>

<https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html>

<https://thoughts-on-java.org/ultimate-guide-association-mappings-jpa-hibernate/>