Ieremias Viorel
Varadi Robert
Gr. 30432

Initiatives Platform
Analysis and Design Document

Revision History

Date	Version	Description	Author
04/Apr/2018	1.0	First Draft	Robert Varadi Ieremias Viorel
25/Apr/2018	2.0	First iteration	Robert Varadi Ieremias Viorel
20/May/2018	2.1	Final adjustments	Ieremias Viorel

Table of Contents

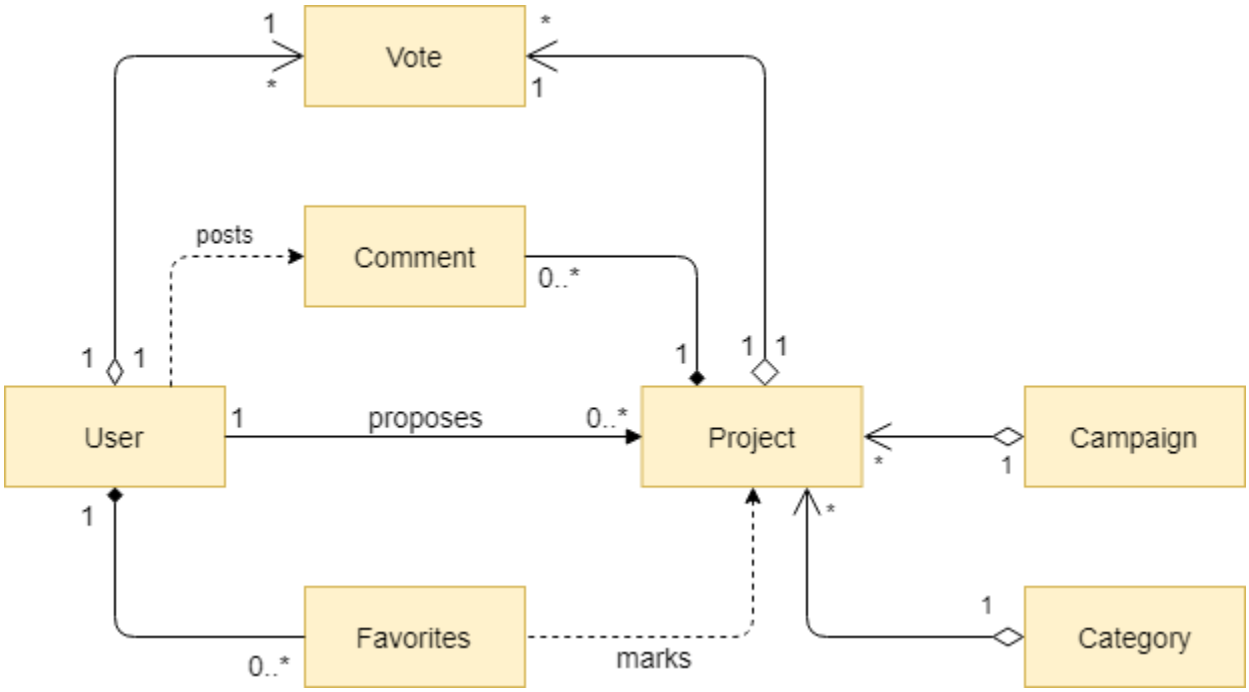
I. Project Specification

Implement a system where regular people can propose plans they would like to see implemented in their city. Other people, also users of the system, vote for the project they consider most useful. Projects are divided in some categories, on domains. Filtering can however be extended based on criteria related to budget or implementation time (short-term, medium-

term, long-term, etc.). Each person is able to vote for a limited number of projects and choosing the best project may take place in more than one round. The application also contains a comments section too, so people are able to discuss about the different projects. Each user has a "favorites" corner from where he/she can choose the projects he/she would like to revisit more times, in order to make the best decision.

II. Elaboration – Iteration 1.1

1. Domain Model



Domain Model Diagram

The main entities involved in the system are the User, the Project and the Vote. A User entity abstracts personal and account information of a real user of the system. A Project entity contains relevant information related to the initiatives and ideas proposed by the citizens. In order to provide a useful classification for the projects, each one has attached a Category attribute. To provide an interaction between the users and to facilitate debating on a subject, a Comment entity is added in the system. Finally, in order to enhance the experience of the user, there is a "Favorites" corner, where a user of the system can mark the projects he/she wants to review later, so a Favorite entity, which maps a User and a Project is necessary. The central purpose of the application is to determine the needs of most of the people so a voting mechanism is built around the Vote entity. A Vote instance represents the decision of a User to support an idea. The multiplicity of the relationships is OneToMany both in case of User-Vote and in case of Project-Vote, because a User has a number of votes he/she can exert, while a Project can be supported, through votes, by multiple users. A more accurate result is obtained if the projects are filtered and eliminated in stages so campaigns, with specific deadlines, are added in the system in order to achieve this purpose.

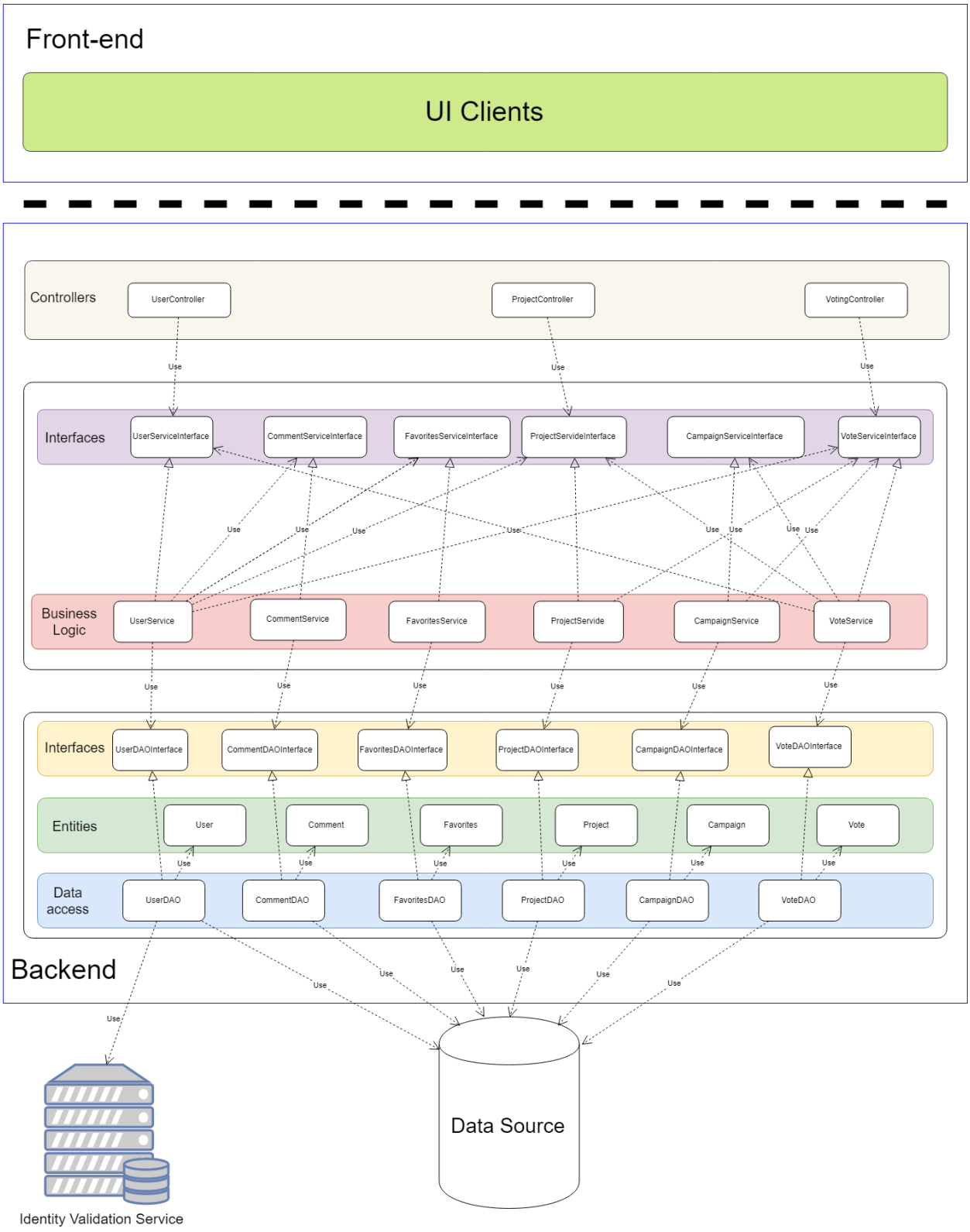
2. Architectural Design

2.1 Conceptual Architecture

The application is built as a decoupled system, composed of a set of services and repositories that provide functionality to the view, that is managed by a few actor-centered controllers. The core receives HTTP requests coming from the client, processes them and returns the rendered page which is displayed in the client. From this perspective, the application is a distributed client – server system.

The whole application is structured using the layers pattern, which has the advantage of isolating specific processing steps into independent components which can be tested, replaced, maintained and extended separately. The layers communicate with each other only through services which are exposed from one to another through interfaces which hide the specific implementation.

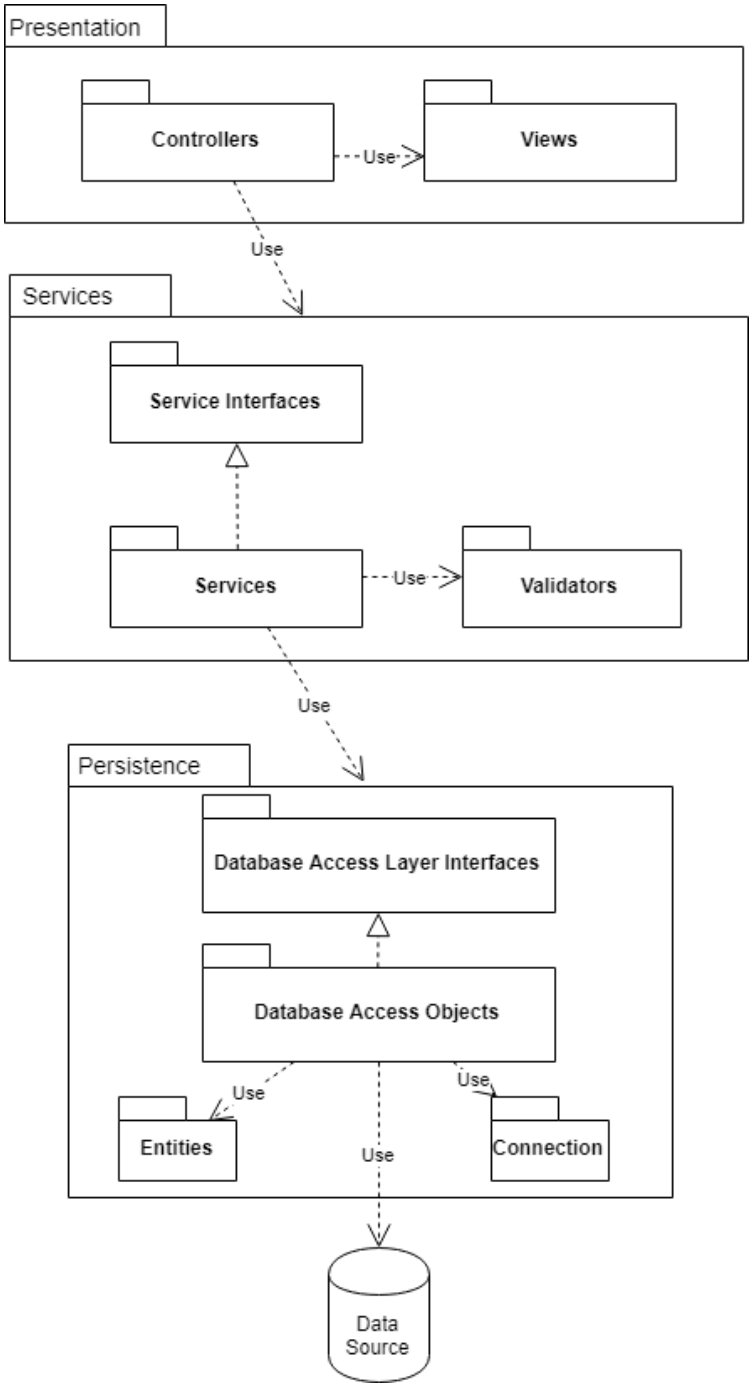
At the application level, we use MVC (Model-View-Controller). The Model component is represented by the persistence layer and the business logic layer, where the state of the application is stored and the corresponding processing logic is implemented. The View component is the graphical user interface itself, which is only responsible for the “look” of the application, namely what the user sees and interacts with. The Controller is responsible for picking-up the input from the user and transmitting it to the back-end for processing.



Architecture Diagram

2.2 Package Design

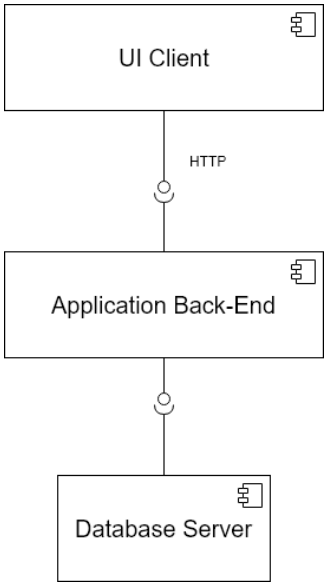
At package design level, each layer is assigned a package, which is responsible for the whole functionality of the layer. The coupling between the layers is low, because the only way a higher – level layer communicates with a lower – level one is through the interface exposed by the latter one.



Package Diagram

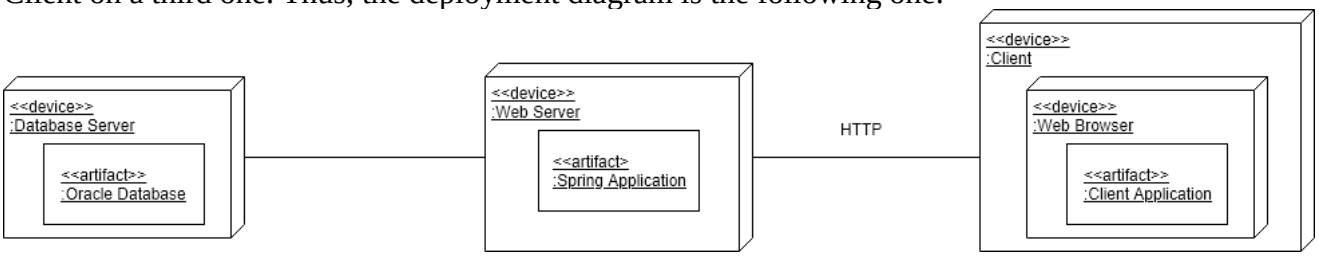
2.3 Component and Deployment Diagrams

The component diagram contains the layout of the system at physical component level, that is, files, libraries, executables. From this perspective, we can draw the following component diagram:



Component diagram

The deployment diagram contains the physical layout of the system grouped by tiers. We might have the Database Server on a tier, the Application Server on a different one and the UI Client on a third one. Thus, the deployment diagram is the following one:



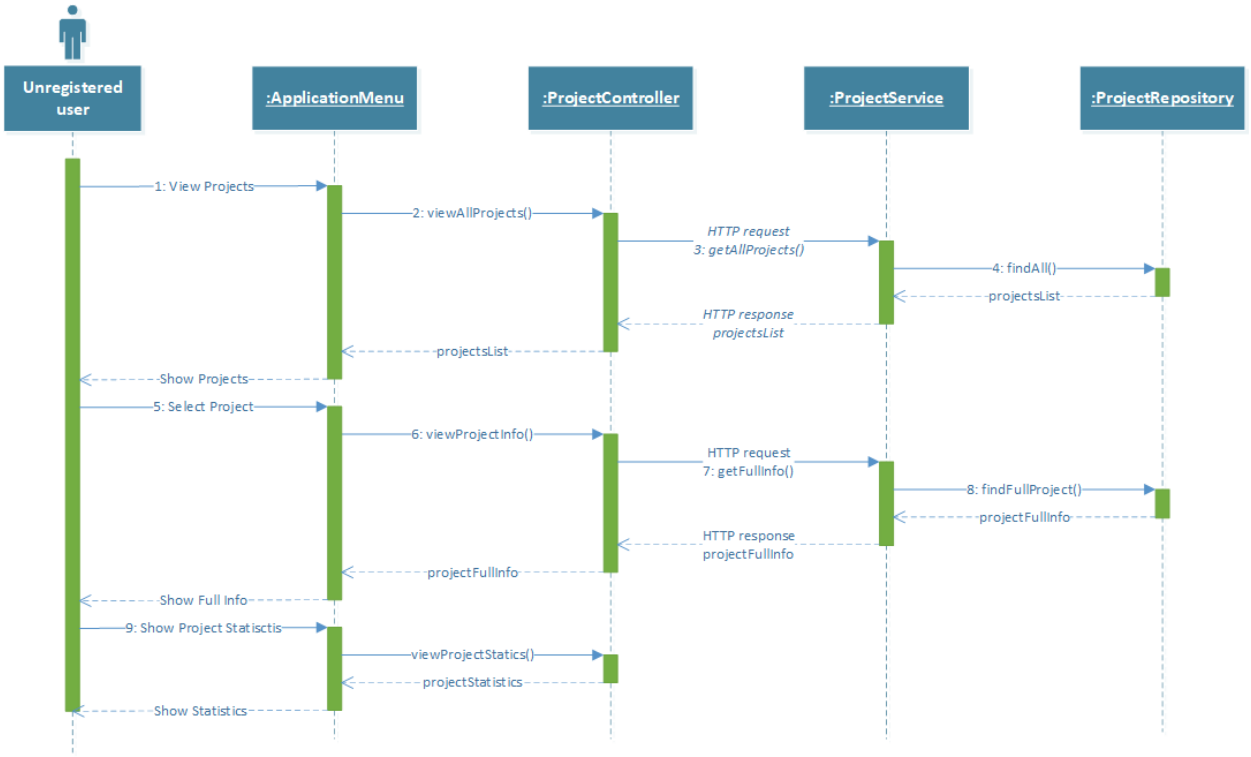
Deployment Diaagram

III. Elaboration – Iteration 1.2

1. Design Model

1.1 Dynamic Behavior

In this subsection, the dynamic behavior of our system is being described on two use cases that we considered relevant. The first use case is the one in which the unregistered user wants to see the projects, the other one being the one in which the content supervisor wants to block a user who is the author of a flagged project or comment. The first use case is described using a sequence diagram, while the second one is described with the help of the communication diagram.

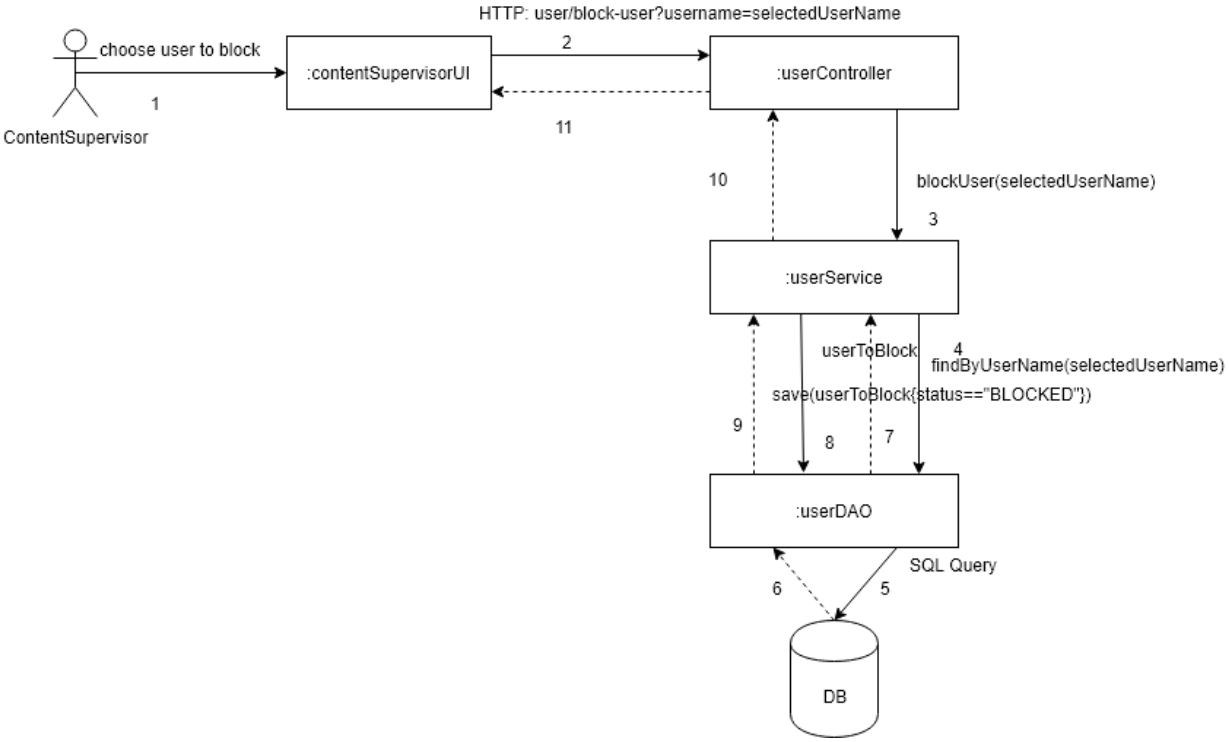


Sequence Diagram

The sequence diagram above presents the evolution of the system when the View Simple Project Statistics use case, of the actor Unregistered user is carried out. This use case coincides with the action of a casual user, that visits the platform without having an account or without logging in into the account and wants to see information about a specific project related to time of proposal, votes and views so far and category to which it belongs.

The user chooses from the Application Menu to see the projects listed so far, either form a specific category, or belonging to any category, based on a filter. A scrollable list is presented to him on the following page. Next the actor has the possibility to view more information about a project it selects. The page is filled with information available for that project. Finally, if besides the general information presented so far, the user wants to see detailed metrics, he uses a dedicated button on the current page that launches a screen with the required information.

The communication diagram models the interactions between objects or parts in terms of sequenced messages. For the “Block User” use case, the content supervisor chooses the user who will be blocked. At this time, the view controller sends an HTTP request to the userService object. This component parses the HTTP request, from which it takes the information needed to identify the user, for example his/her username. This information is processed by the UserService, which fetches the user to be blocked by asking the userDAO to query the database and return the user who has the given username. As soon as the userService has the User object associated to the user who has to be blocked, the service can modify the object’s status to “BLOCKED” and then ask the userDAO to save the modified user into the database.



Communication Diagram

1.2 Class Design

In this subsection we describe the design of the entities which take part in the system. These entities are the following ones:

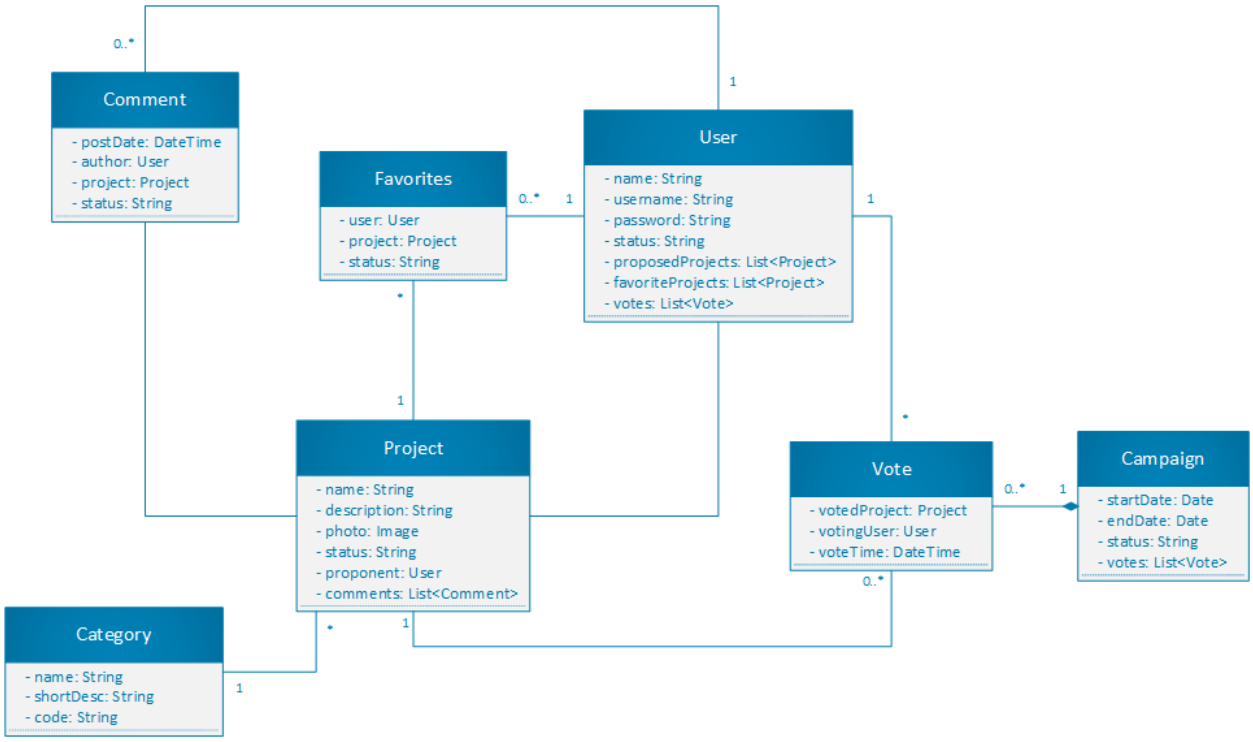
- User
- Project
- Comment
- Vote
- Category
- Campaign
- Favorites

The entities have the following relations between them:

- A User object has more associated comments since a user can post more comments in different parts of the application;
- A User has more Favorites objects because he/she can mark more projects present in the application as favorites;
- A User has more votes associated since he/she is able to vote for more projects;
- A Project may have more comments related to it, so there is a one to many association relationship;
- More projects may have the same category, so the relationship between projects and categories will be many-to-one.
- A project may have more votes so the proper relationship between Project and Vote will

be the one-to-many association relationship;

- During a campaign more users can vote and no vote should exist without a campaign so the relationship between Vote and Campaign is a many-to-one composition relationship.



Class Diagram

1.3. Design Patterns

Builder pattern is used in the case of the User entity. The system has 3 types of users: regular users of the application, content advisors and system admins. The attributes related to personal information, are captured in a UserInfo field that is necessary and thus common to all the users. The differentiation between the different types of users is made based on a ProfileInfo field, that stores information about the allowed and restricted functions for each type. An implementation based on the factory pattern makes it easier to create and handle users.

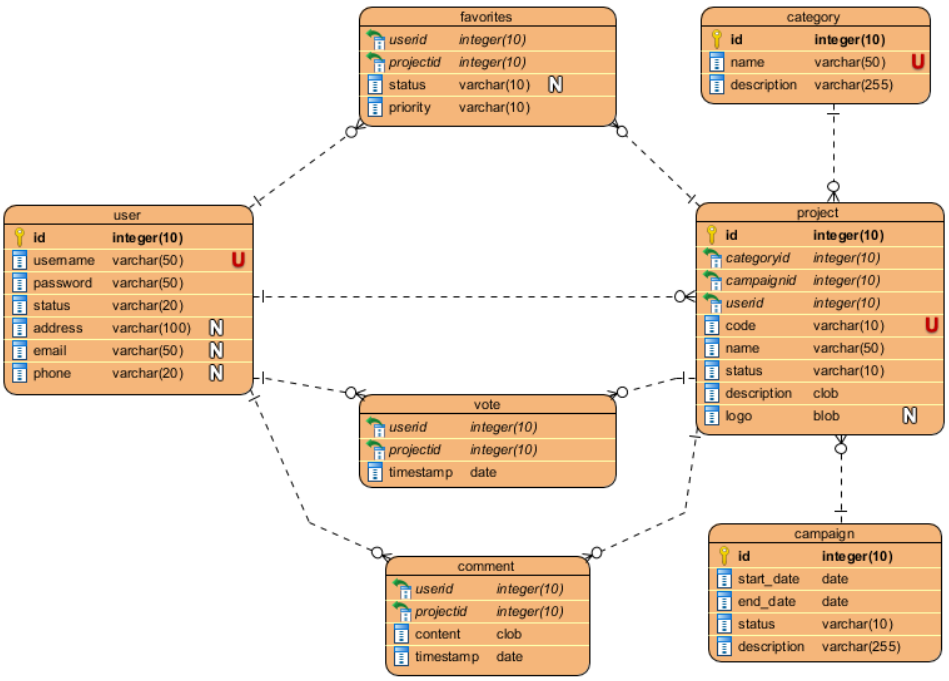
Prototype pattern is considered for handling large and costly objects. The entities that fit into this category are instances of the User class, and hold many fields, some of which are collections of references to other objects. Prototype objects that are cached and persisted only when changes occur are used with the purpose to decrease the effort to access the database and create new instances.

Filter pattern is necessary because the system is populated with many entities of the same class, mainly users, projects and votes. Complex filters have to be implemented on all these entities, so an approach based on the filter pattern allows splitting the functionality into small sized filters that can be composed into complex structures.

Singleton pattern is useful when creating and handling a unique service, which assures data integrity and consistency, despite concurrent calls being made to the service.

2. Data Model

The data model or database diagram is almost the same as the class diagram presented above, the difference is that each object will have a primary key or a composite key and these keys will be used to reference the objects, instead of pointers. The multiplicities of the relationships are the same. The red "U" letter on the right of the attribute means that the attribute should be unique, while the "N" letter means that the attribute might be null.



Data Model Diagram

3. Unit Testing

Unit test are performed on the repositories and services, both in sequential and concurrent conditions, to make sure they work as prescribed by their contracts. JUnit 5 is the main framework for testing.

IV. Construction and Transition

1. Future improvements

Further improvements include:

- Addition of a mobile application
- Addition of 2-step authentication

- Sending notifications to users when some events take place

V. Bibliography

- Fowler Martin, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002
- E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns. AddisonWesley, 1995. [GoF]
- David Patterson, Armando Fox, Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing, Alpha Ed.
- http://www.codeproject.com/KB/architecture/MVC_MVP_MVVM_design.aspx
- https://academy.realm.io/posts/mvc-vs-mvp-vs-mvvm-vs-mvi-mobilization-moskala/
- https://www.infoq.com/articles/no-more-mvc-frameworks?utm_source=infoq&utm_campaign=user_page&utm_medium=link