**QQ\_ProGem Management**

Content :

1. [Overview and concept](#progeam_overview)
2. [Features](#progeam_features)
3. [Technologies in use](#tech_in_use)
4. [High level System Design](#highlevel_sd_applicaiton)
5. [Authentication and authorization](#auth)
6. Project Management Section
   1. [Overview and concepts](#pm_overview)
   2. [High level design of the architecture](#pm_highlevel_sd)
   3. [Features](#pm_features)
   4. [Database design](#pm_db)
   5. [How do we ….?](#pm_how_do_i)
7. Social Network Section
   1. Overview and concepts
   2. High level design of the architecture
   3. Database design
   4. How do we ….?

**Overview**

ProGem is an feature rich Project Manager that makes it very easy to manage projects with a team or complex sub sections.

ProGem is a tool that is going to make managing projects easier and well structured, it allows users to divide projects into sub projects with their own goals, members and further sub projects.

It also comes with it’s own social networking platform where users can post blogs, and show their contributions towards tasks and projects.

Further interaction includes features like commenting, likes and friends

**Features**

1. Creating and managing projects with rich features (Features will be discussed in details in “Overview and Concept” section of “Project Management Section”)
2. Interact with other users through posts, comments, likes
3. Showcase your tasks and projects to others
4. Contribute towards public tasks and projects or do things privately
5. Make friends, interact with them and inspect their biography

**Technologies in use**

**Front-end :**

Flutter framework

**Back-end :**

We are making use of micro services architecture hence back-end is very flexible and loosely coupled allowing for usage of multiple technologies but as of now we are using the following technology :

Spring core (Dependency Injection and basic Spring framework)

Spring Boot (To make spring configuration easier)

Spring MVC (For RESTful API)

Spring Security (For authentication and authorization)

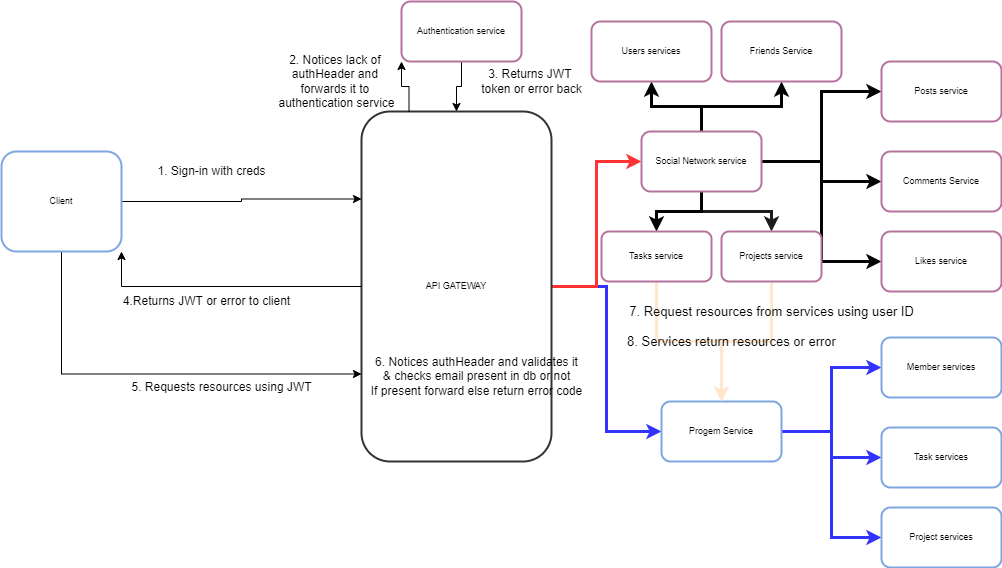
Spring Cloud (For Gateway and other features such as rate limiting)

As you can see we are making use of spring technologies as of now but planning to expand to other technologies as time goes on.

**High level System Design of the application architecture**

Front-end is going to connect to an API Gateway which is going to handle authentication using JWT. Once gateway is done authenticating, it is going to send the jwt token to the client. The client then going to use this token for every request. When Gateway finds that the token is valid it is going to forward the request to other micro services internally and return the client with the response.

I have still yet to decide if Gate should parse the userID from the jwt token it receives and then send it to micro services or If I should send the token to micro services who is then going to parse it. I personally do not think that there is going to be a difference so as of now I am planning to work with Parsing JWT at the API Gateway itself. This will save a little bit of time because parsing in each service will need it’s own JWT implementation which will result in duplication in code and make it hard to refactor.



**Authentication and Authorization**

Authentication : Verifying is the client is who they claim to be

Authorization : Checking if the client is allowed to access the item it is asking for

We are going to go stateless with authorization.

JWT is a good choice for Stateless authorization. The way JWT works is by generating a token based on secret key stored in server. You can store any payload you want in the token, in our case we are going to store userID as payload.

The token is going to get stored in client and it can’t be manipulated because it’s payload and signature is going to be unique and is going to result in invalid token if anyone tried to manipulate it.

The API Gateway is going to be responsible for parsing this token and using it to get the intended userID and forwarding it to the micro services.

If AuthHeader contains “Guest” as token we are going to treat the user as a guest and forward it

We also have to manage api limiting and blocking IPs

**Project Management Section**

**Overview and concepts**

Projects are the main component. They are going to consist of members and leaders. The leaders can assign tasks and members can get the task done.

It is very difficult to structure a complex and/or large project hence why I introduced the feature of breaking down project into sub projects. These sub projects are going to have its own set of task, leaders, members, rules, etc.

A project is going to consist of :-

1. Sub projects : Branches of project that deserves it’s own section. Example :- A web application can a back-end sub project which is going to define the back-end section of the web application. The project can also have a front-end sub project which is going to define the front-end section of the web application. Sub projects can have further sub projects and this is going to form a tree structure.
2. Tasks : Goals that are give to members OR sub-projects by it’s parent project
3. Members : Members are users who are going to do tasks.
4. Leaders : Leaders are users who have special rights and belong to a project or sub projects

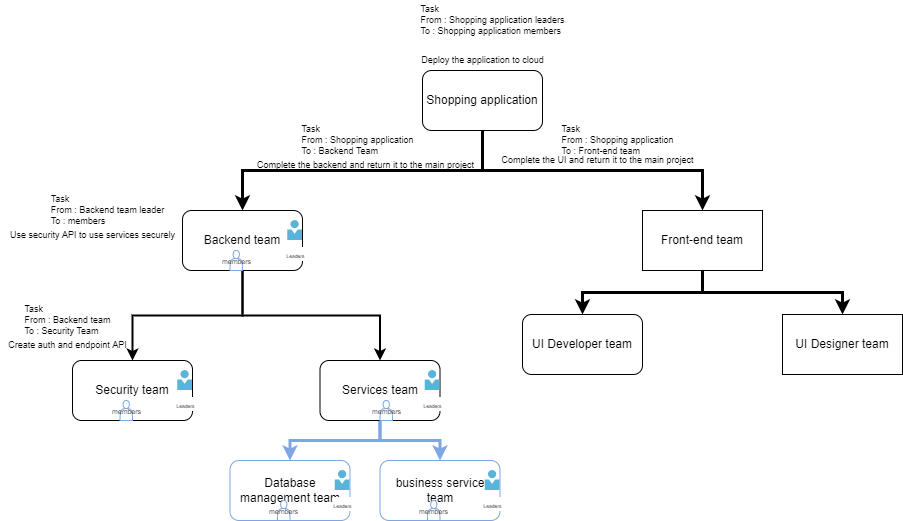
The root project is going to be an admin and will have full access to the project.

They can create new sub projects with specific access control such as creating sub projects underneath them and so on. We term these rules as **Project Rules.**

The same logic applies for tasks as well as leaders. They have rules set to them and are called **Task Rules** and **Leader rules** respectively.

**High Level Design of ProGem’s Project Manager architecture**

Image of ProGem’s Project structure

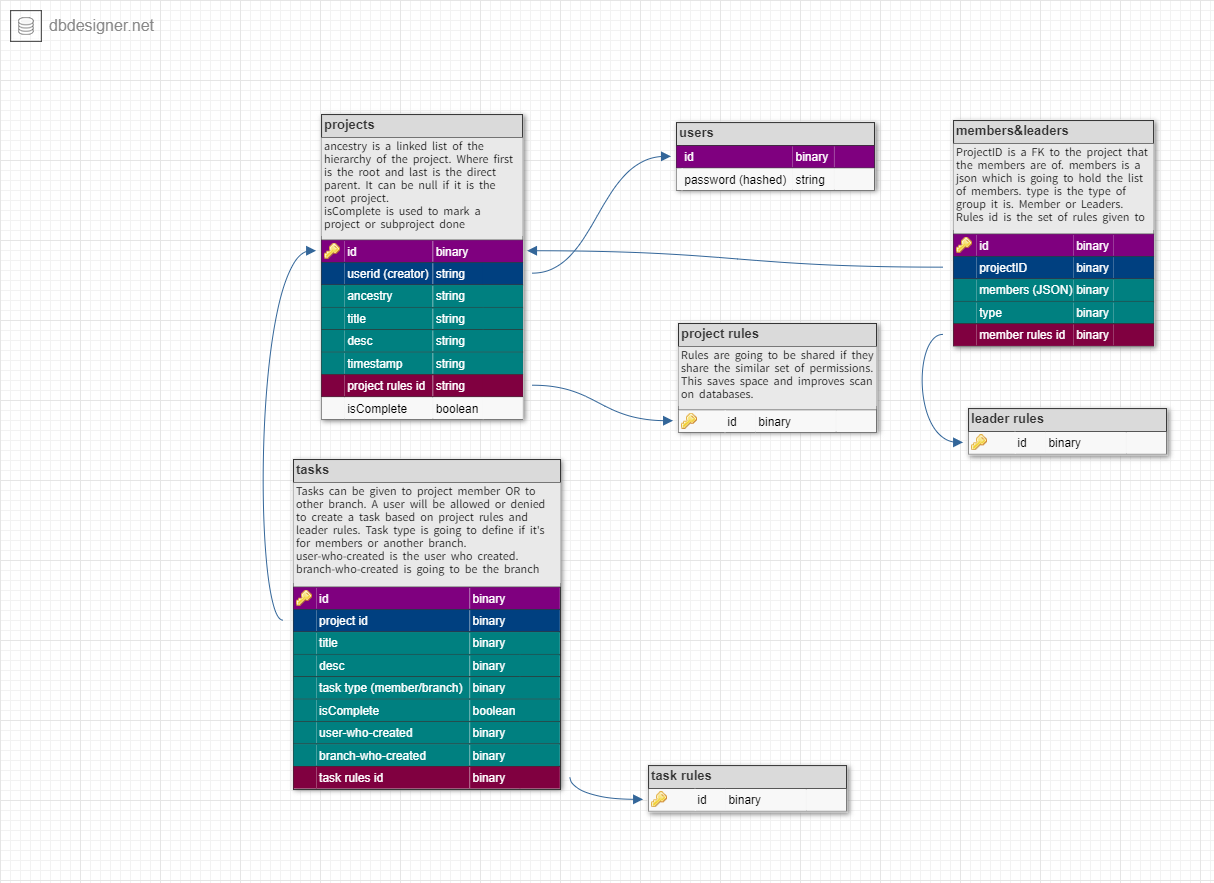


We can see how a project is going to have multiple branch and tasks given to it’s members and children projects.

**Features**

1. **Creating Main project** with set of leaders and admin.
2. **Visibility of project** can be public, friends or participants only.
3. **Creating sub projects** with it’s own set of **project rules.**
4. **Adding leaders and members** with it’s own set of **leader rules**. Such as visibility, allowed to create task for children, etc.
5. **Assigning tasks to children or members** as a leader (If allowed)
   1. Tasks can be made to be verified by project leaders and/or parent project leaders Or have no verification at all. This is going to determine if a task needs to be verified before it can be marked as completed.
   2. Tasks will also have it’s own set of rules that are going to be defined by the task producer
   3. Just like project tasks will have its own visibility setting
   4. All these customization will be defined in task rules for the task
   5. Tasks that are public are open to public contribution and will need to be verified by the leader of the project and such tasks are going to appear as contribution on the user’s profile who solved it
6. Authentication and authorization for allowing users to view and create project as per their scope
   1. Guest users will be able to see only those projects, task and members that are public
   2. Authenticated users will be able to see posts of their friends if the one who posted has set the visibility to friend
   3. Private projects are only going to be visible to only it’s members

**Database Design of ProGem’s Project Manager**

****

Example Record

Users

|  |
| --- |
| userID |
| kukuborom@gmail.com |

Projects

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | userID | Ancestry | Title | Desc | TimeStamp | Project Rules ID | Is Complete |
| 46 | [kukuborom@gmail.com](mailto:kukuborom@gmail.com) | Null | RootProject | This is the root | 2.4.23 | 65 | false |
| 87 | Null (was not root) | 46-> | secondChild | This is child | 2.4.23 | 65 | False |
| 78 | Null | 46->87 | thirdChild | Grandchild | 2.4.23 | 53 | false |

Project-Rules

|  |  |
| --- | --- |
| Id | rules |
| 65 | {  Scope : private,  createChildren : true,  } |
| 53 | {  Scope : public,  createChildren : false  } |

Users : This table is going to be the primary table where users are stored.

Projects : This table is going to have a value added every time we create a new project or sub project and the rules we set are going to be added in **project rules** table.

To determine if a project is complete we have a column known as isComplete.

Project rules : When we set rules for a project the server is going to check if there is already a rule with such settings, if there is we are going to set it as the rule ID for the project in **project table.**

Tasks : Tasks are given to either members or children. We also keep a record of which user created it as well as which branch created it. RulesId is going to link the rules with the task.

Task rules : Same as Project rules table but for tasks.

Members and leaders : This table is going to hold a json list of userIds. The type is going to define if it is a leader or a member

The rulesID is what is going to be used to determine it’s access control.

Member rules : Same as Project rules table but for members and leaders.

**How do we ….?**

1. Create a new project?
   1. Server expects a payload which will contain userID, Project title, Desc, and optional RULE (No rule for Root project), ancestry (NULL for root project)
   2. Server then adds this record to projects table to the database and adds the userID who created as the leader of root project with full access
2. Create new sub project
   1. Server expects a payload with userID, project title, Rules, ancestry of parent
      1. For example lets take the following data is the ancestry of it’s parent whose id is “child1”

Root->foo

So foo is the parent of “child1”

* 1. Server will then add a new data to the ancestry as shown Root->foo->child1
  2. A new record will be added in project table with it’s ancestry as Root->foo->child1
     1. This shows the hierarchy of the project where the parent of the newly added sub project is “child1” which is what we assumed the parent’s projectID to be.

1. Delete a root/sub project:
   1. Server expects a userID and projectID
   2. Lets projectID is “foo”
   3. Verify if userID is leader and has rights
   4. Get all project who has ancestry with value “foo”
   5. Loop through the result and delete the appropriate task and member&leader
   6. Once done delete the project id and the task, member&leader related to it.
   7. Scan the project table to see if project rule is in use anywhere, if not delete the rule
2. Create new task
   1. Server expects a userID and projectID
   2. Checks if userID is leader and has rights
   3. Adds the task in table
   4. Adds rules in task rules table or assigns one to the task depending on if it found a similar rule or not
3. Delete task
   1. Server expects a userID and taskID
   2. Gets projectId based on taskID
   3. Checks if the userID is leader and has right to do so
   4. Deletes the task
   5. Stores the ruleID
   6. Scan task table to see if anyone is using the ruleID, if no one found delete the rule from task ruleTable
4. Mark task as complete
5. Assign leader
6. Assign members
7. Delete Leader
8. Delete member