Analysis Me

**Authors**   
Leeza Sinha  
Charu Thaore  
Gongpu Zhu  
Gaurav Shegokar

# Table of Contents

# 

[**1. Introduction**](#_tpfsji12iqcj) **4**

[**2. Motivation**](#_8wxjzh2tii5m) **4**

[**3. Related work**](#_hcwxftwbhzo2) **5**

[**4. System design (Gaurav and Gongpu)**](#_30j0zll) **7**

[4+1 Views:](#_z61v1i4zie79) 7

[Use Case View](#_xlckwswqdjle) 7

[Process View](#_w69l4t39en89) 10

[Logical View](#_r9ptc5ule0oj) 11

[Deployment View](#_uzqyikt38fzu) 12

[Development View](#_cdh7gw5u9m86) 13

[**5. System implementation**](#_rv03fc2rspso) **14**

[Memento](#_cc9vjpapr0hl) 14

[Flyweight](#_6vffdgqrxfjv) 14

[Builder](#_8peqa8t4edd3) 15

[Template](#_4k1s8371umm2) 15

[Observer](#_2juk6fyhxtbz) 16

[Facade](#_s7zp5jqachlh) 17

[Mediator](#_uw1zjsy80pxq) 17

[Command](#_xrckp1hroq9u) 18

[Filter](#_jeeexx8km0qx) 19

[Singleton](#_b58vznpsvkfd) 19

[Factory](#_kgfzo3xsb0gj) 20

[Visitor](#_ohpab8kiwe4) 21

[State](#_tlwvpvvnzrg3) 22

[Strategy](#_cnvmr06dbdk0) 23

[Proxy](#_x4k6pyliygnk) 24

[Adapter](#_non27jk4lyx8) 24

[Prototype](#_4gkzq48psajc) 25

[Decorator](#_gn5jzjerwfdc) 26

[Iterator](#_ph3vmvn9p4xy) 27

[Interpreter](#_qwfr6ys6g7sk) 27

[**6. Tutorial**](#_m4uq8t540zjg) **28**

[How to use the Application](#_hyvrojt4qnlj) 28

[1. Download and Install required softwares](#_x0is3wvo3doi) 28

[2. Download/Setup & StartUp Application](#_jaqcyp5r9eca) 30

[3. FrontEnd project set-up/startup](#_b5uzijrzf1o9) 32

[4. Server (Backend) project setup/start](#_3i05a9t7vvoi) 34

[Application Demo/Execution steps](#_lir09gtx3wpx) 41

[**7. Self analysis**](#_vmrx6ph97b4e) **49**

[Better Collaboration - Comprehensive API documentation](#_159sddp2mat0) 49

[Clean and Elegant UI](#_gyc9zeydzotg) 50

[In-Depth Understanding of Design Patterns](#_bjqb49b5ta1n) 50

[Ebean ORM made our life easy](#_jpndmswg9aeg) 50

[**8. Iteration workflow and contribution**](#_uf6d58bkw9ti) **51**

[Iteration Workflow for each Sprint](#_wi68kztm1wsz) 51

[Contribution](#_2xotijul3pte) 52

[**9. Conclusions and recommended future work**](#_d5i48skrj23i) **53**

[Conclusion](#_o9bcfnlyjvs) 53

[Future Work](#_s7zi93nrk17m) 53

# 

# 

# 

# 1. Introduction

Forums are a powerful tool to connect people who are not even remotely related to each other, work together. What starts off as a simple discussion around new ideas, can often evolve into valuable knowledge which could be used to the benefit of the whole community. For instance, early discussions may end up guiding the creation of a new product and/or services. Ask.Me is one such online forum which encourages discussions. It improves collaboration as well as helps in seeking assistance for a problem.

Ask.Me supports a lot of important forum features which are described as follows. Registration/signup of a user, all the usernames in Ask.Me are unique, we have implemented security mechanisms to make sure our user’s data remains protected. Users can create their own channels and every post a user creates is associated with a channel, this is a peculiar feature which we have provided. It makes searching related posts very easy. We have hashtags functionality as well, which lets users categorize posts as per their convenience. A post can be searched based on channel or hashtags. Users can view posts/comments which he/she has posted and they are sorted based on the timestamp. We have a service level subscription based model implemented in our application. Hence the prices for users and admin varies. Users can pay using paypal or credit card.

We have implemented twenty design patterns in total, namely Memento, Flyweight, Builder, Template, Observer, Facade, Mediator, Command, Filter, Singleton, Factory, Visitor, State, Strategy, Proxy, Adapter, Prototype, Decorator, Iterator, Interpreter. Detailed explanation about where and how we have implemented them is given in the later sections.

# 2. Motivation

The primary motivation behind this project was to understand how and where to implement design patterns. [Anti-patterns](http://en.wikipedia.org/wiki/Anti-pattern) are certain patterns in software development that are considered bad programming practices. As opposed to [design patterns](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)) which are common approaches to common problems which have been formalized and are generally considered a good development practice, anti-patterns are the opposite and are undesirable. For example, in object-oriented programming, the idea is to separate the software into small pieces called objects. An anti-pattern in object-oriented programming is a [God object](http://en.wikipedia.org/wiki/God_object) which performs a lot of functions which would be better separated into different objects.

Through this project we wanted to learn how to not create anti-patterns and understand the scenarios where implementing a design pattern would be actually beneficial. Misuse of patterns implementation leads to unnecessary complex design and ultimately causes performance implications, we wanted to learn how and when to avoid such situations.

Design and architectural styles are very crucial in achieving non-functional requirements. Our motto was to learn how they help in achieving these requirements in the product. We have applied patterns to make our application extensible, maintainable, available and modular.

# 3. Related work

Our application uses Angular framework for frontend, Play framework for backend, MySQL database for storing data. Following is the description of each of the technology which we have used to make our project successful.

* Angular



[An](https://angularjs.org/)gular is built with the [Model-View-Controller concept](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) in mind, though authors of the framework often called it “Model-View-\*” or even “Model-View-Whatever”.

The framework, written in pure JavaScript, is intended to decouple an application’s logic from DOM manipulation, and aimes at dynamic page updates. It isn’t very intrusive i.e we could have only a part of the page controlled by AngularJS. This framework introduces many powerful features allowing the developer to create rich, single-page applications quite easily.

* Play Framework



Play Framework is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) [web application framework](https://en.wikipedia.org/wiki/Web_application_framework) which follows the [model–view–controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) (MVC) [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern_(computer_science)). It is written in [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) and usable from other programming languages that are compiled to [JVM](https://en.wikipedia.org/wiki/JVM) [Bytecode](https://en.wikipedia.org/wiki/Bytecode), e.g. [Java](https://en.wikipedia.org/wiki/Java_(programming_language)).

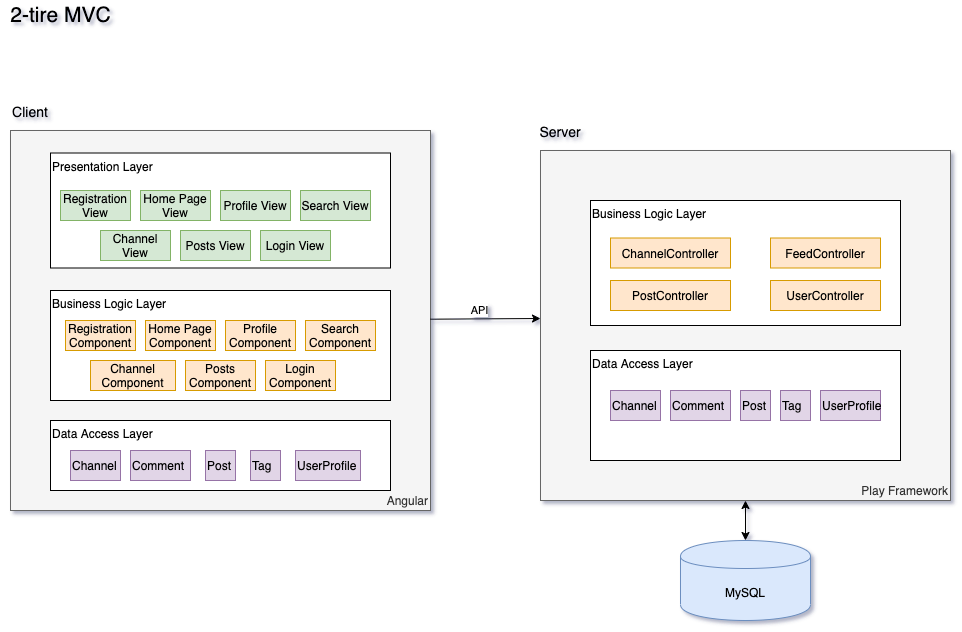
Support for the [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) programming language has been available since version 1.1 of the framework. In version 2.0, the framework core was rewritten in Scala. Build and deployment was migrated to [SBT](https://en.wikipedia.org/wiki/SBT_(software)), and [templates](https://en.wikipedia.org/wiki/Web_template) use Scala instead of [Apache Groovy](https://en.wikipedia.org/wiki/Groovy_(programming_language)).

* MySQL



MySQL is a relational database management system based on SQL – Structured Query Language. The application is used for a wide range of purposes, including data warehousing, e-commerce, and logging applications. The most common use for mySQL however, is for the purpose of a web database.

# 4. System design (Gaurav and Gongpu)



Since we are using the Play framework as the backbone of the server side and Angular as the render engine on the frontend, the system follows the 2-tier MVC. Both the frontend and the backend have the data access layer containing all models and the business layer including all controllers. On top of that, the frontend also has a presentation layer which serves as a view layer.

RESTful APIs act as the only bridge between the frontend and backend. Frontend talks to the backend through RESTful APIs.

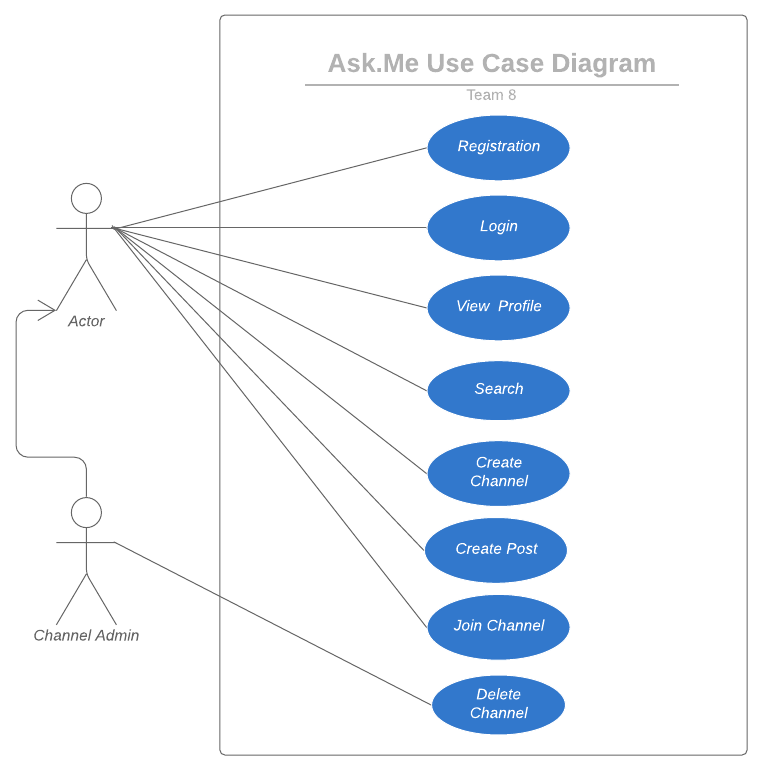
And on the backend, we use ebean to handle database operations.

The 2-tier MVC design achieves a clean separation of concern. And the backend might be reused to a different frontend.

## 

## 4+1 Views:

### Use Case View



The above use case diagram shows scenarios supported by the system. We will discuss such use cases.

**1. Registration**The user should be able to register himself on the system.

**2. Login**The registered user should be able to login to the system by entering credentials used while registration use case. Once logged in the user could see the main page where the posts on the subscribed channels or hashtags could be seen.

**3. View Profile**Once the user is logged in, he could view his profile. That includes the users details - username, first name, last name. Along with it, he can see what all channels he has subscribed to and created channels. Also the user could see the posts created by him on channels.

**4. Search**The logged-in user could search the information on the following levels

1. Post
2. Hashtag

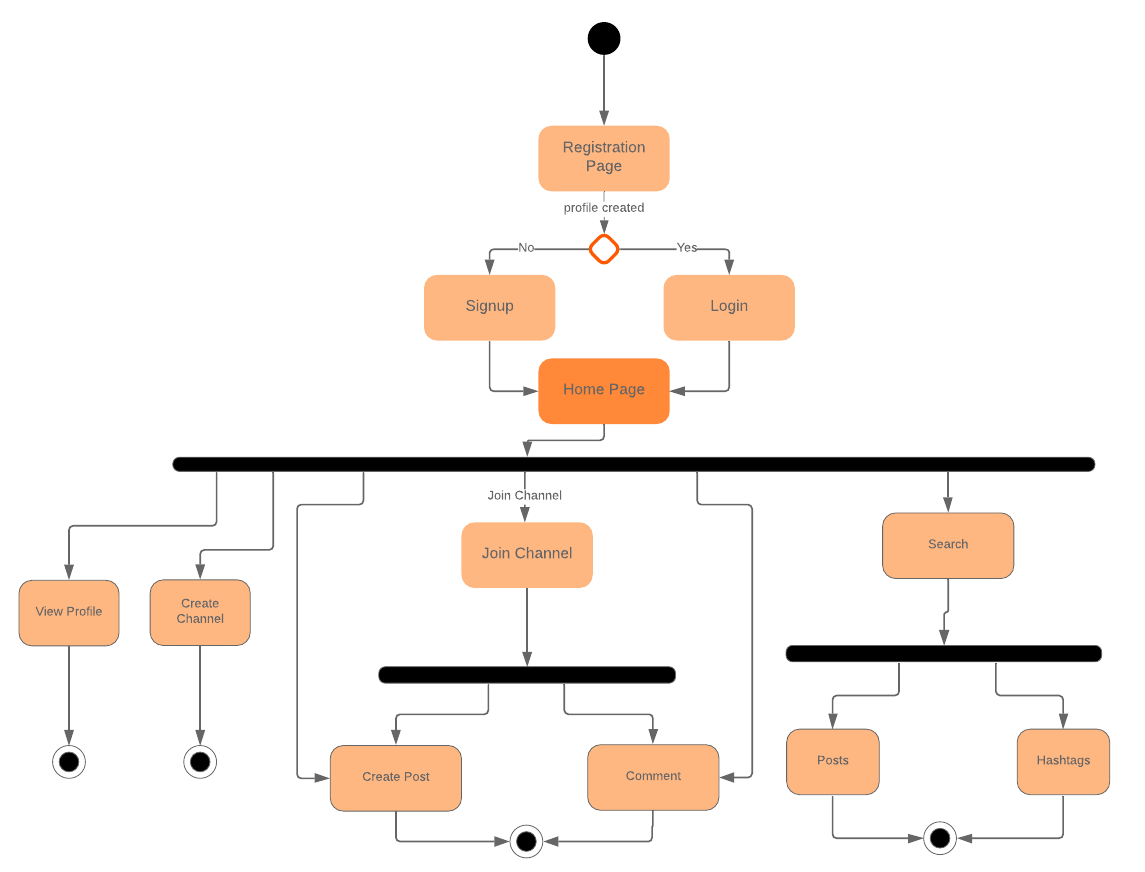
**5. Channel Creation**The user could create a channel according to his needs by providing the required information as the channel description, channel name, etc

**6. Join Channel**The user should be able to join the channel. Once the channel is joined the user can see the channel posts in the feed as well as create a post in the channel.

**7. Delete Channel**Only Channel admin (user who has created the channel can delete a channel)

### 

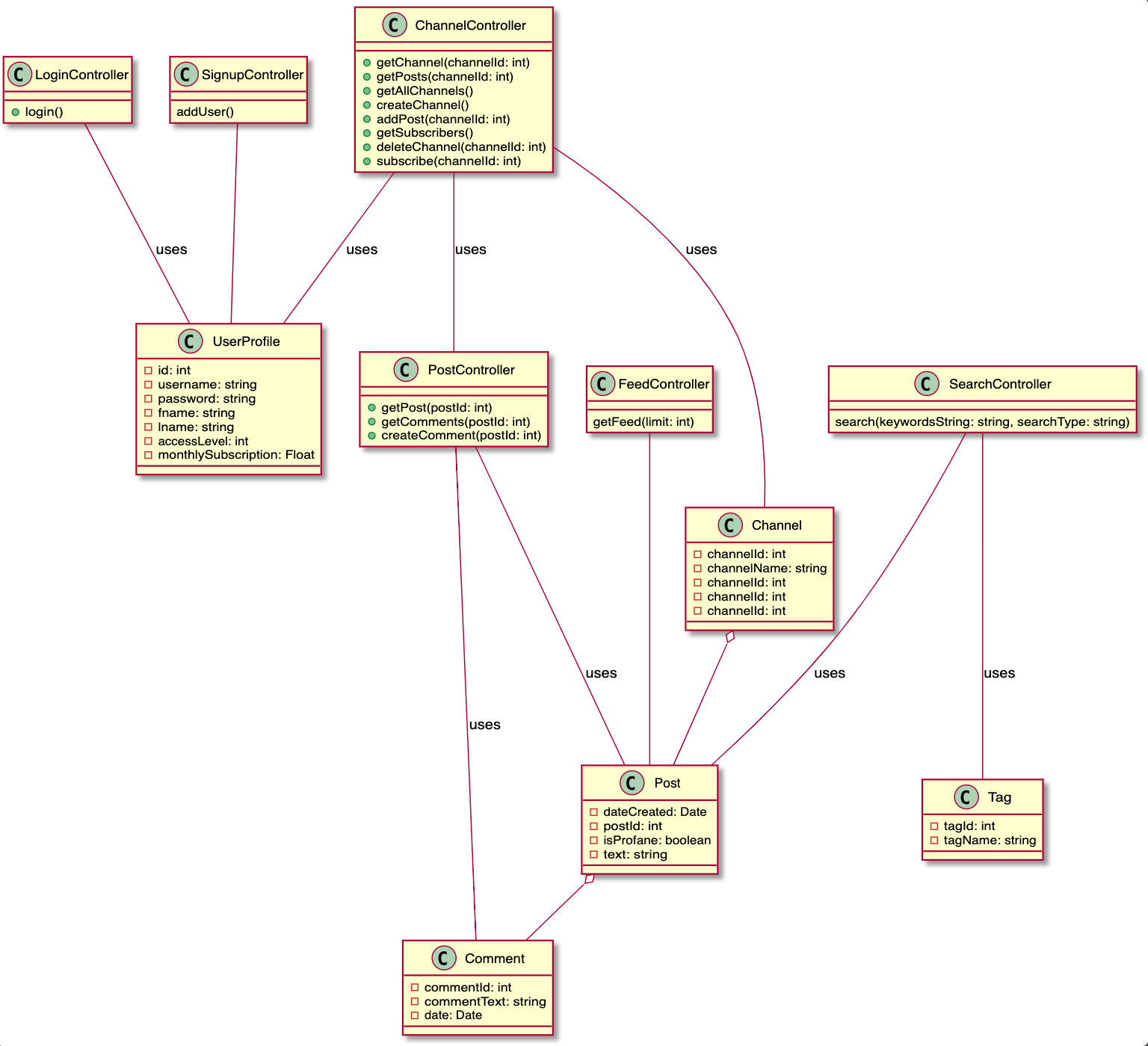
### Process View



The process view diagram is used to display the functionalities of various activities and flows in business processes and software systems. That is the dynamic view of the software system. As evident from the diagram, the user has to register or login to use the application. Once logged in, the users have several functionalities available as shown in the process view such as Join Channel, View Profile, Create Channel, Search, Create Post, add Comment. The descriptions about these functionalities are listed in the Use Case View.

Process views are important from the architectural point of view. It shows the dynamic behavior of the system and helps to understand the control flow among the subsystems. Without the process view in place, it would become difficult to understand the behavior or the process flow. They are important to make architectural decisions in terms of scaling, distributing, organizing the software system. The process view displays the represented features and the respective process flows after going through a particular option or process.

### Logical View



This is the logical view of the system. The class diagram describes the structure of the system by showing system classes their attributes, operations, and relationships among objects. The diagram describes various controller classes and their interactions with different model classes. Methods are displayed for the controller classes and attributes are mentioned in the respective model classes. One important thing to notice here is there is an aggregation relationship between the Comment - Post Class and Post - Channel Class.

### Deployment View

# 

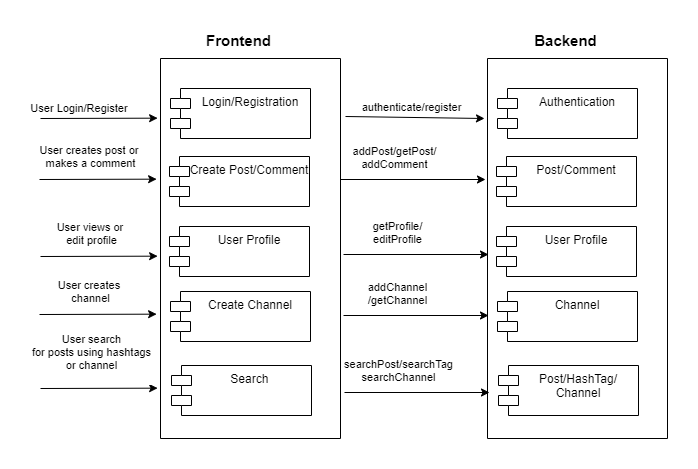
The Deployment view details on the infrastructure and network component of the application. Frontend and Backend communicate using the HTTP protocol Backend Communicates with the Database Server to facilitate the user requests. Web Client can access the application using HTTP requests on the frontend. We have integrated the Social Media - Twitter for the hashtag.

This view describes the environment within which the system is executed; it shows the system from the operator’s point of view. The frontend and backend subsystems are to be deployed as separate encapsulated packages, preferably on separate VM machines for enhanced performance and optimal usage of the available resources.

### 

### 

### Development View



The Development View illustrates the system from a programmer's perspective.

We have two separate Subsystems in this project namely: Frontend and Backend. The diagram shows different logical components present in frontend and backend systems. It includes view components such as login, registration, profile, channel, search. These components interact with backend models in order to display results or set values. Backend subsystems consist of components such as Post-Comment module, Authentication module, etc, which interact with the database to perform CRUD functions and facilitate the client requests.

Since the target number of users we are expecting is in millions, we wanted to keep our project as modular as possible. In the future, we will deploy each component as shown above, as a microservice. For now to avoid impact of changes in front end on backend services and vice versa, we are developing front-end and backend separately. This approach will also help us to assign project work accordingly.

# 

# 5. System implementation

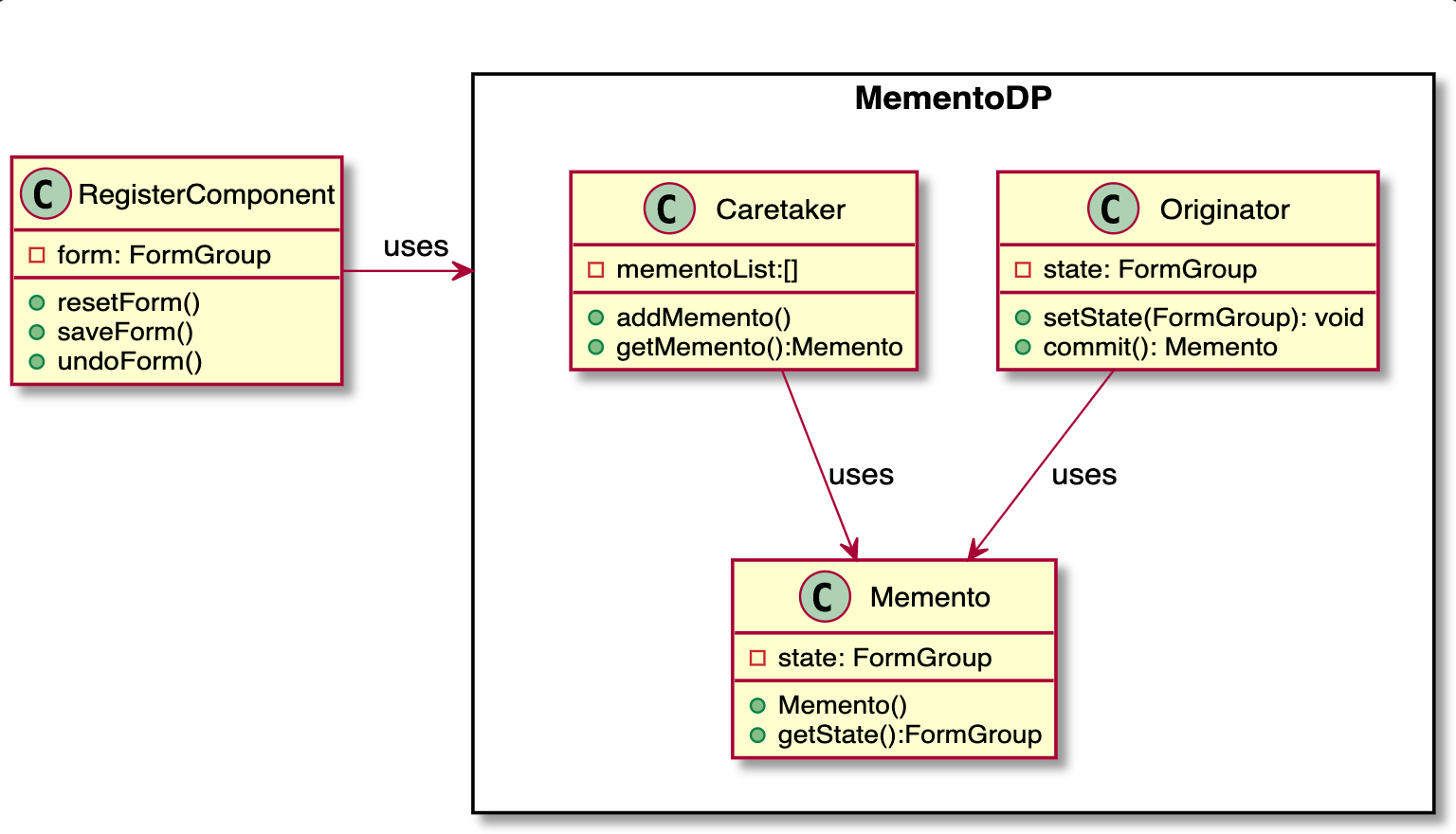
In the past two sprints, we as a team totally implemented 20 design patterns, including memento, flyweight, builder, template, etc.

## Memento

One feature is that we need to give the user an option to reset the signup form or to save the signup form so that he can undo the last saved state.

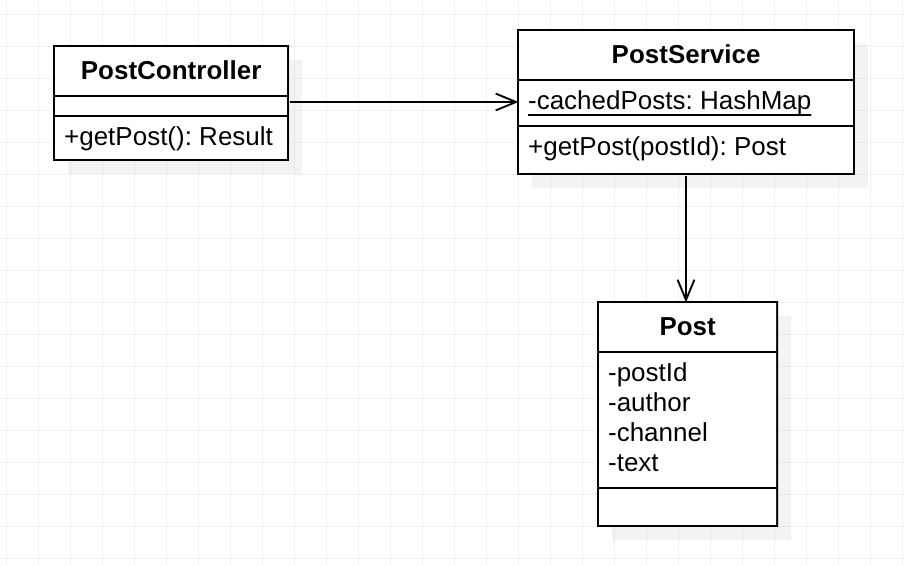
We make use of the Memento pattern. First the state of Signup Form is saved so that the user can reset the form. With it, we can also provide options for the user to save the form.

The initial state of the form with some default values is stored using the Memento Design Pattern. Whenever the user clicks on the reset button the initial state is restored.



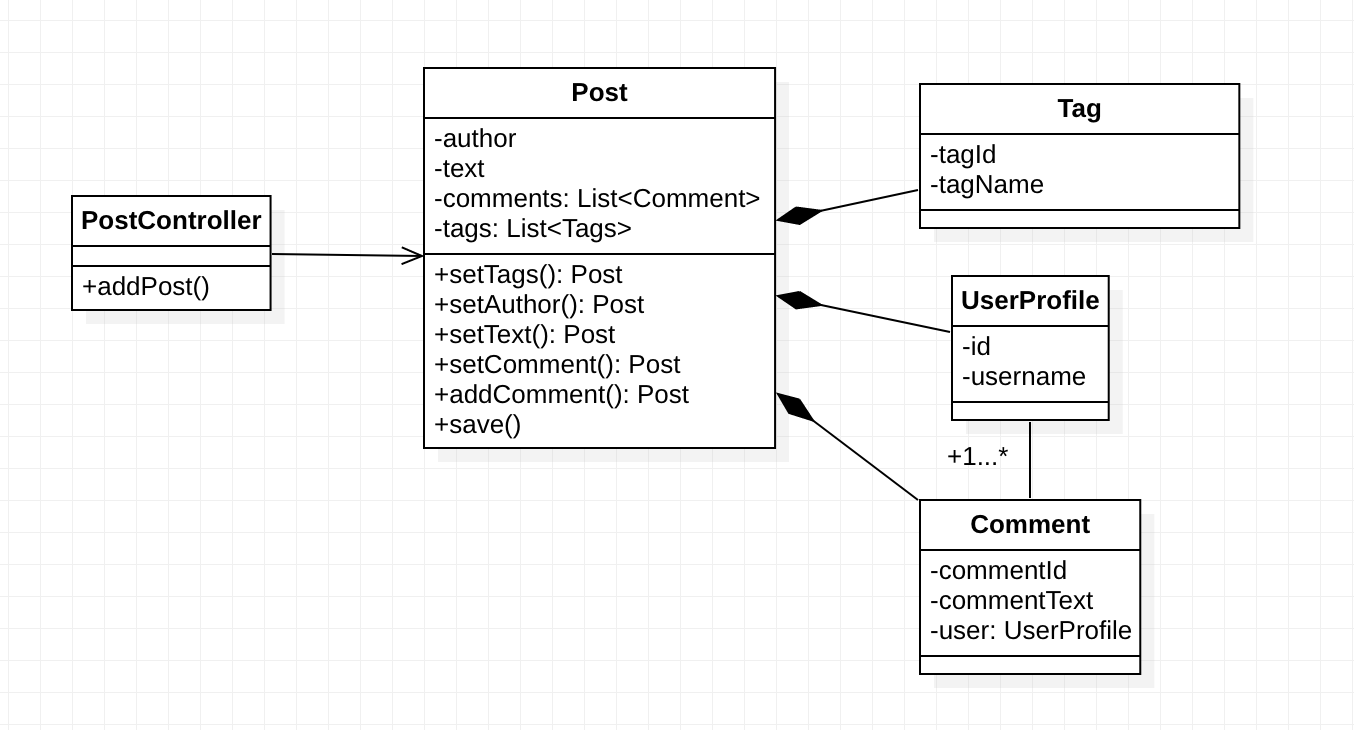
## Flyweight

To reduce the IO footprint of database access, we use the flyweight pattern to cache posts in memory. To be specific, PostService caches posts in a HashMap keyed with their Id. When retrieving a post, we first check if it is cached in the hash map, otherwise we will fire a query in the database.



## Builder

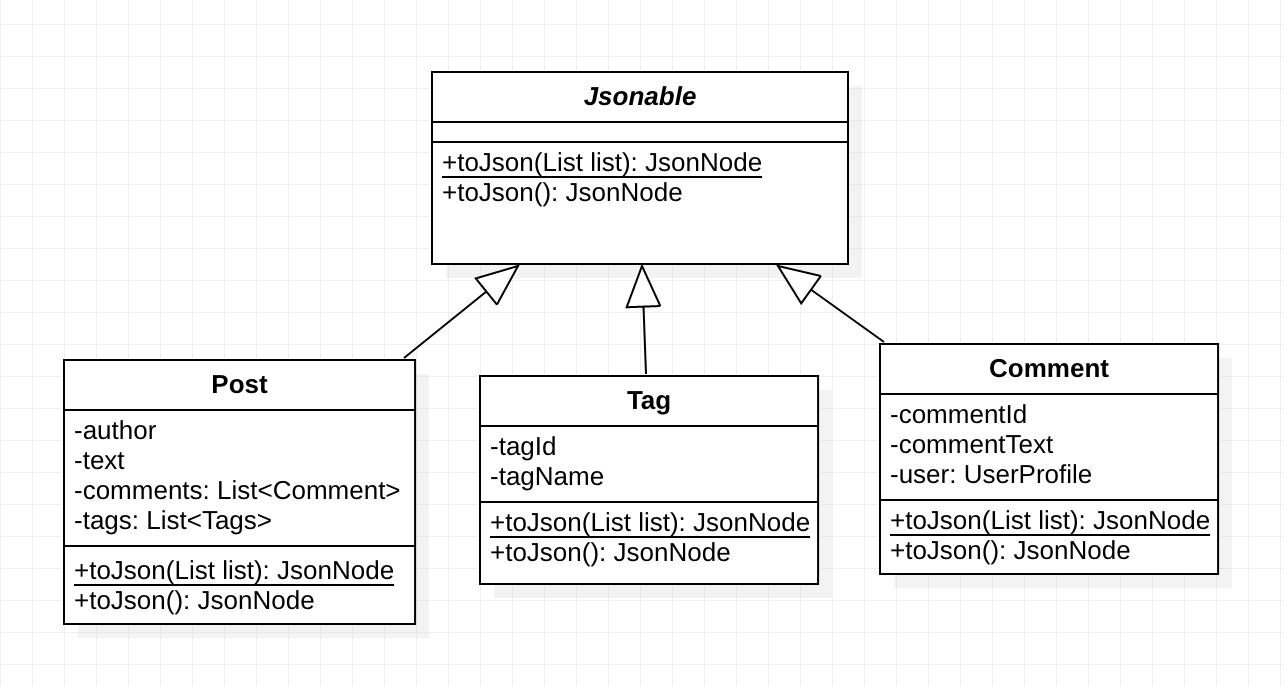
When creating a post on the server side, we need to set its text content, author info, tags, and comments, and save it to the database in the end. In this case, it is a perfect fit for the builder pattern. Also, we make all the setters in the Post model return the identity so that we can chain the operations until saved.



## Template

On the backend, models should be serialized into JSON format to send as responses to the frontend. This is a common behavior across the entire backend. So we choose the Template Pattern to encapsulate such a behavior into the Jsonable interface.

The interface defines a toJson method, which converts the instance into JsonNode and should be implemented by the concrete classes, and a toJson static method, which converts a list of instances into JsonNode. Currently, Post, Tag, and Comment models all conform to this interface.



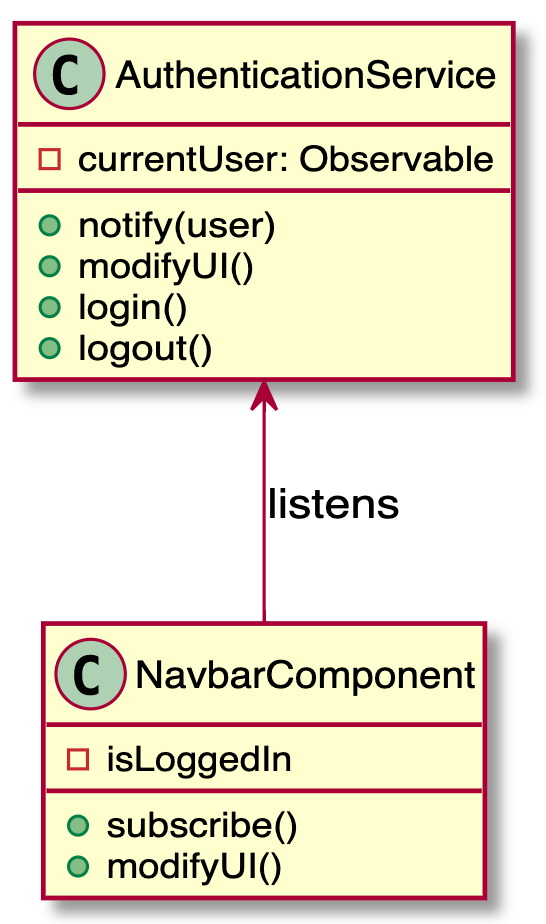
## Observer

We have to display the functionalities of the application when the user logs in or logs out. To facilitate that, reload the webpage again in order to reflect available functionality in the navigation bar.

Instead of a refreshing mechanism, we can make use of the Observer pattern. Whenever the user logs in or logs out it generates an event and notifies its subscribers. Upon getting the event from the notifier the Navigation Bar Component changes its visible functionalities.

The code is implemented in the frontend. Whenever the user logs in or logs out the event is triggered to all its subscribers.

We have the AuthenticationService class which has an Observable object. NavbarComponent class subscribes to the Observable object from the AuthenticationService class. When the user logs in or logs out, it notifies the NavbarComponent and it changes the visible functionalities accordingly in the UI without refreshing the page.



## Facade

We use the Facade pattern when we need to have a limited but straightforward interface to a complex subsystem. Here, we want the users to use the application without knowing how the internal subsystems communicate and bring out the necessary results in the UI. To achieve this we are using the Navigation Bar in the UI. Using the navigation bar, the user can access the functionalities implemented by the system.

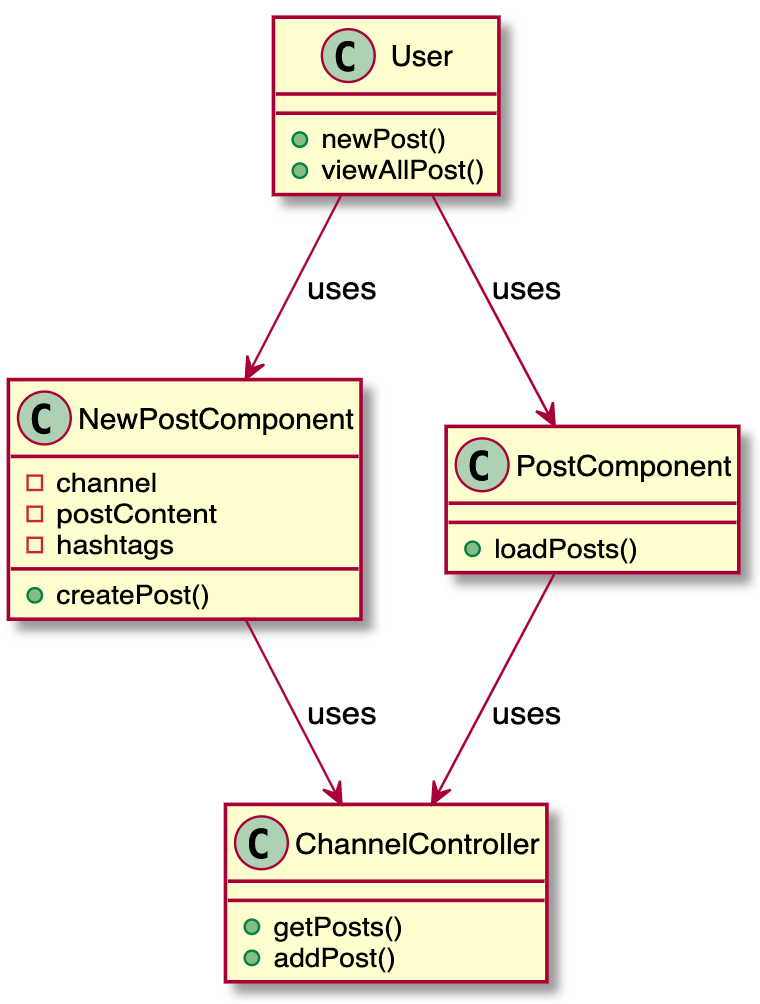
The design pattern is implemented in the frontend, it collaborates the complex subsystems in the backend to facilitate the expected behavior. Using the navigation bar, the user can access the functionalities implemented by the system such as view profile, create post, search, login, logout, etc.

## 

## Mediator

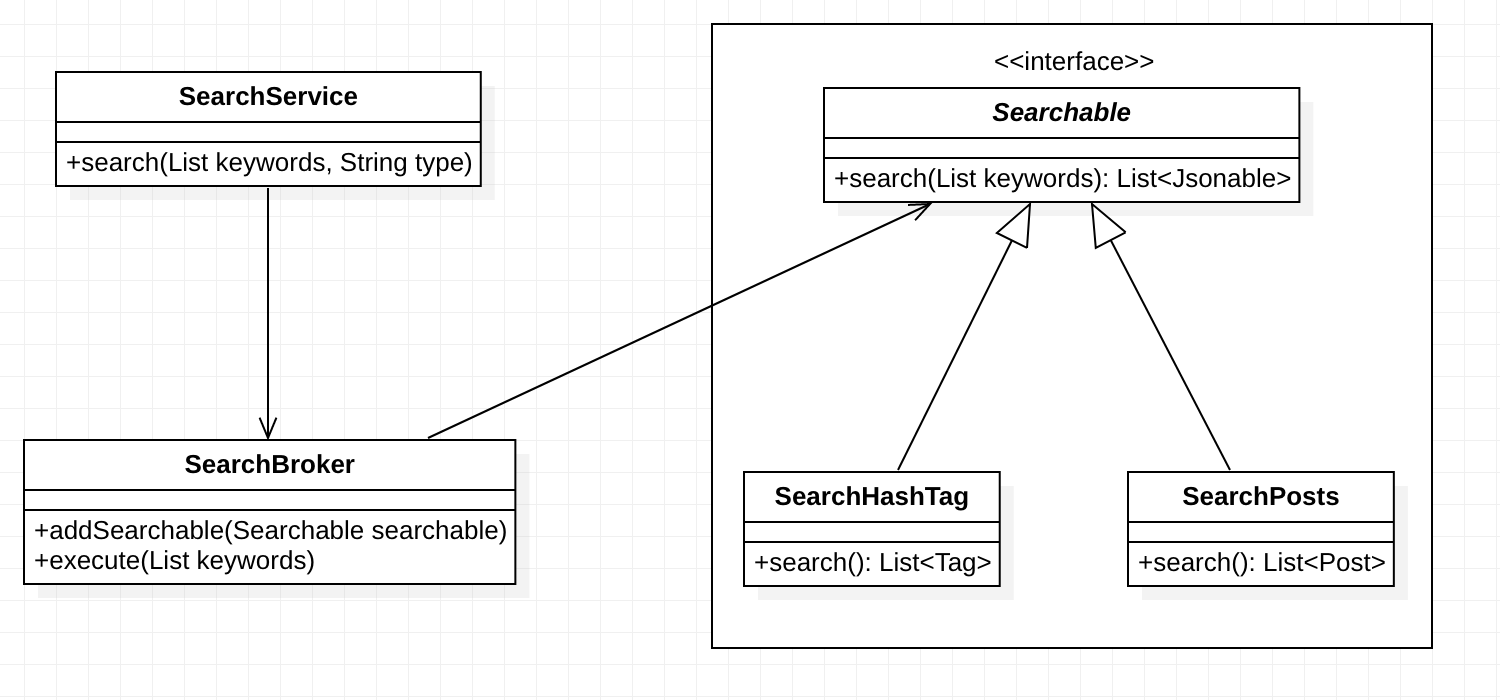
We have to display a post created by different users to all users in the channel. Here, we can make use of the mediator pattern. Multiple users can create posts on a channel and it is the responsibility of the channel to show the posts to all users. Instead of communicating posts to all the users, users will only communicate with the channel and the channel will then display the posts to all the users. As a result, the channel will act as a mediator between the users.

The client will create a post in the channel using the interface of ChannelController for a Channel with method addPost. It will take input as the username, post contents, channel where the post will be posted. For viewing the posts the client uses the interface of ChannelController specifically the method getPosts to get all the posts added by all users.



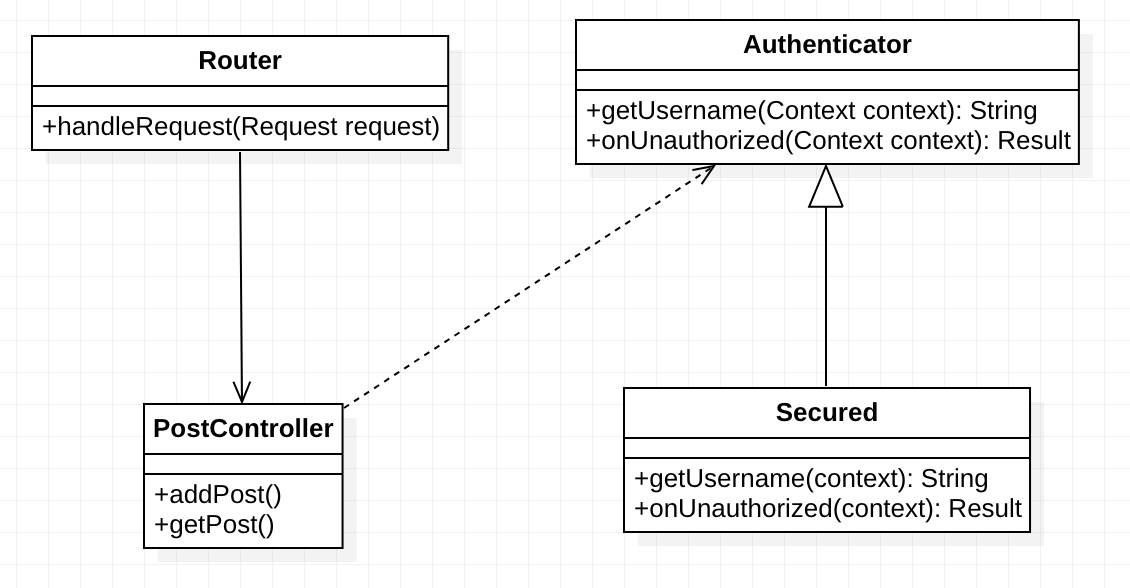
## Command

In the forum, we support three kinds of searches, including searching by posts, or by hashtags, or both. We use the command pattern to implement this requirement. Each search request serves as a command containing keywords and search type from the client side. Then we add SearchHashTag or SearchPosts or both to the SearchBroker based on the search type that the user expects. When the searchables are resolved and added, SearchBroker will execute the search in sequence and return the result.



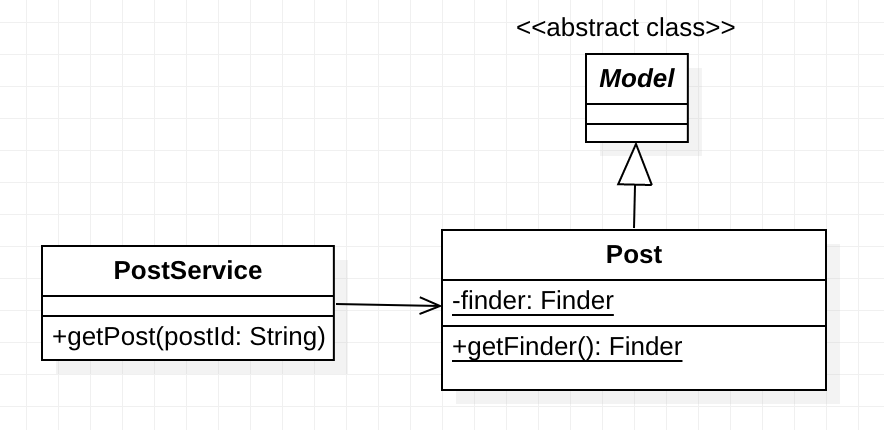
## Filter

We use the filter pattern to filter out unauthorized requests and only pass those valid requests to corresponding controllers. To be specific, we build a Secured middleware, which inherits from Play’s Authenticator class, as a filter using if a valid user id appears in the request header as the criteria. Basically, every incoming request to the server will go through the Secured middleware/filter first, and only those authenticated requests will be pipelined into controllers.



## Singleton

Each model in the backend has a static Finder instance to do queries in the database through Ebean. We make these Finder instances singleton in each model. For example, in the Post model, we expose a getFinder method to return the private static finder instance. In the PostService class, when getting a post by its id, it will invoke the getFinder method to build and execute a query.



## Factory

While creating the user, the client shouldn’t need to know which class to call in order to create an admin user or a regular user. Also, if a requirement arises to create a third type of user, no code changes should be required at the client side.

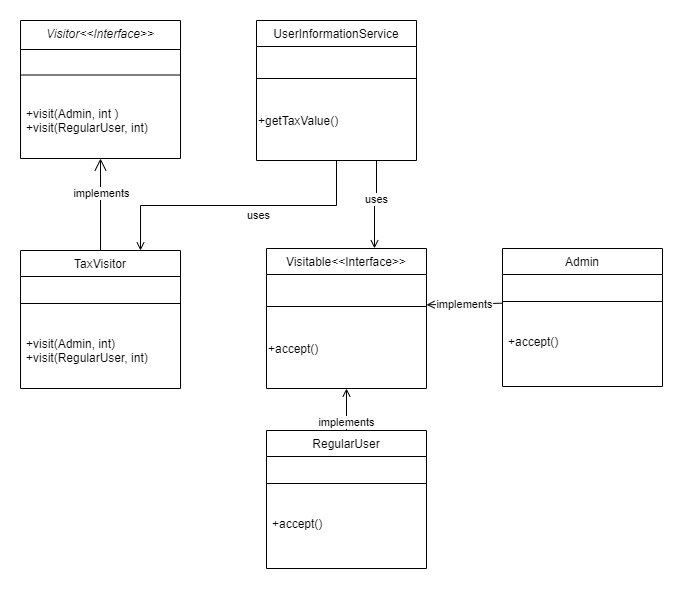
We implemented the factory design pattern which allows the subclass to decide the type of object to create. Client doesn’t need to know about it. Also, for adding another type of user, just userFactory code needs to be modified.

We have implemented an interface called User. Admin class and Regular user class inherits it. We have created a UserFactory class that decides which type of user needs to be created at runtime. Whenever a request is received by SignupController(client) to create a new User, it asks the SignupService to create a user. SignupService calls the UserFactory to get the user object depending on the type requested by the controller.

# 

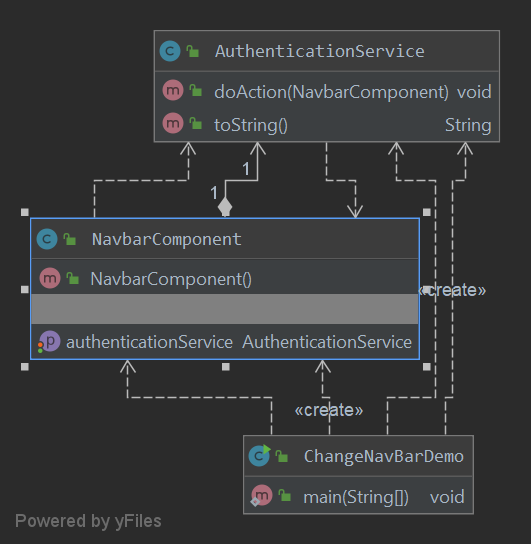
## Visitor

The visitor pattern is used when similar action(but different algo) is required for different types of objects. In our current implementation of the payment system, action was to calculate tax on seperate objects i.e the tax calculation algorithm varies as the object varies. Instead of adding tax calculation methods in these objects we made use of the visitor pattern which allow us to calculate tax once the object accepts the visitor.



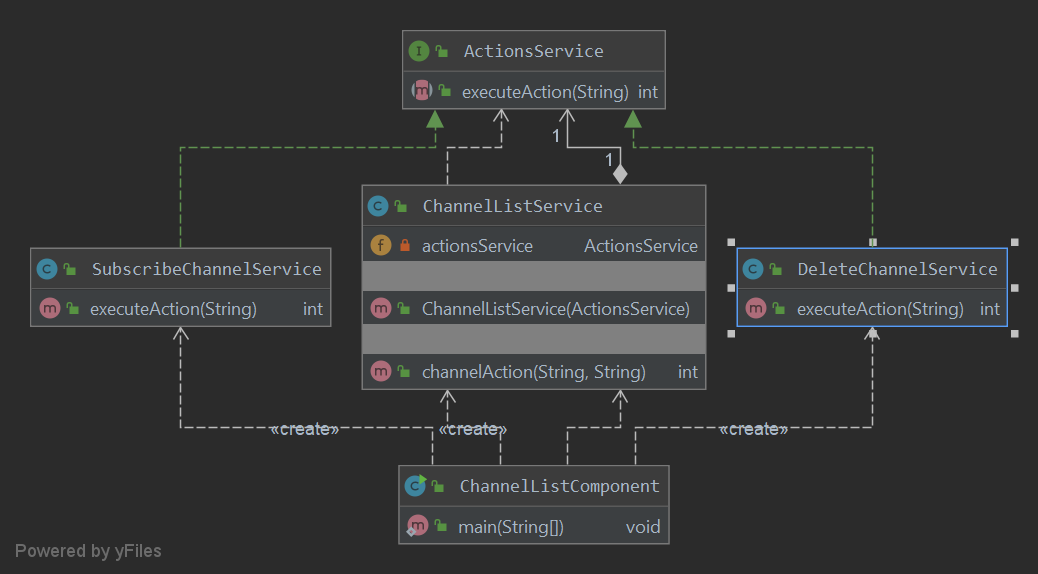
## State

The State of the currentUser authentication is maintained in the AuthenticationService whereas the context object NavbarComponent keeps a reference to the AuthenticationService object. The behaviour of the Context object (NavbarComponent) changes as per the change in the logged in user state in the AuthenticationService i.e the Navigation Bar will change depending upon the fact that user is logged in or not.



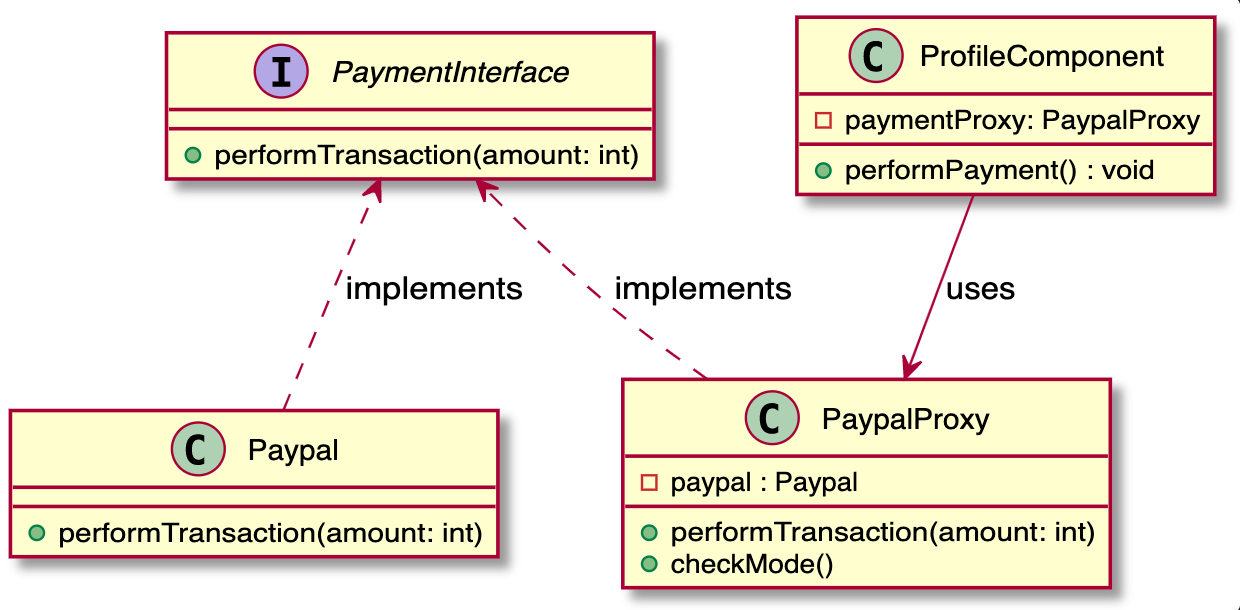
## Strategy

The Channel page allows the user to select either subscribe or delete channel options. Based upon the user selection action, the SubscribeChannelSerice or the DeleteChannelService is called appropriately.



## Proxy

Proxy pattern perfectly fits in the payment processing. We create one more handle for the PayPal that does the mode checking as well, if the mode is development, the payment is not done but if it is in production mode, payment is delegated to Paypal service through Proxy.



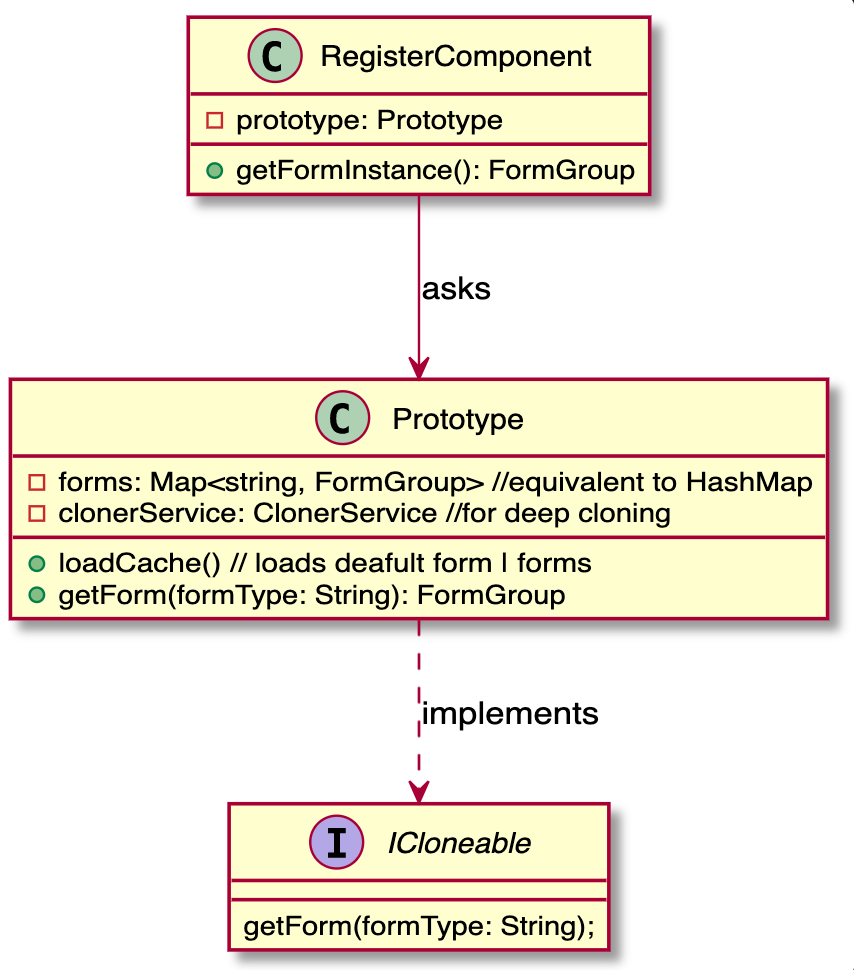
## Adapter

We get data from the backend for the user’s keyword search. API call is made and the data is received in JSON format. In the Json data, we have date strings, but the frontend expects dates in the Date object format. So to convert the incompatible interface of SearchService’s search method. We are making use of the adapter pattern. We use the SearchResultsAdapter class to use the legacy code and then convert the data in compatible format as expected by frontend.



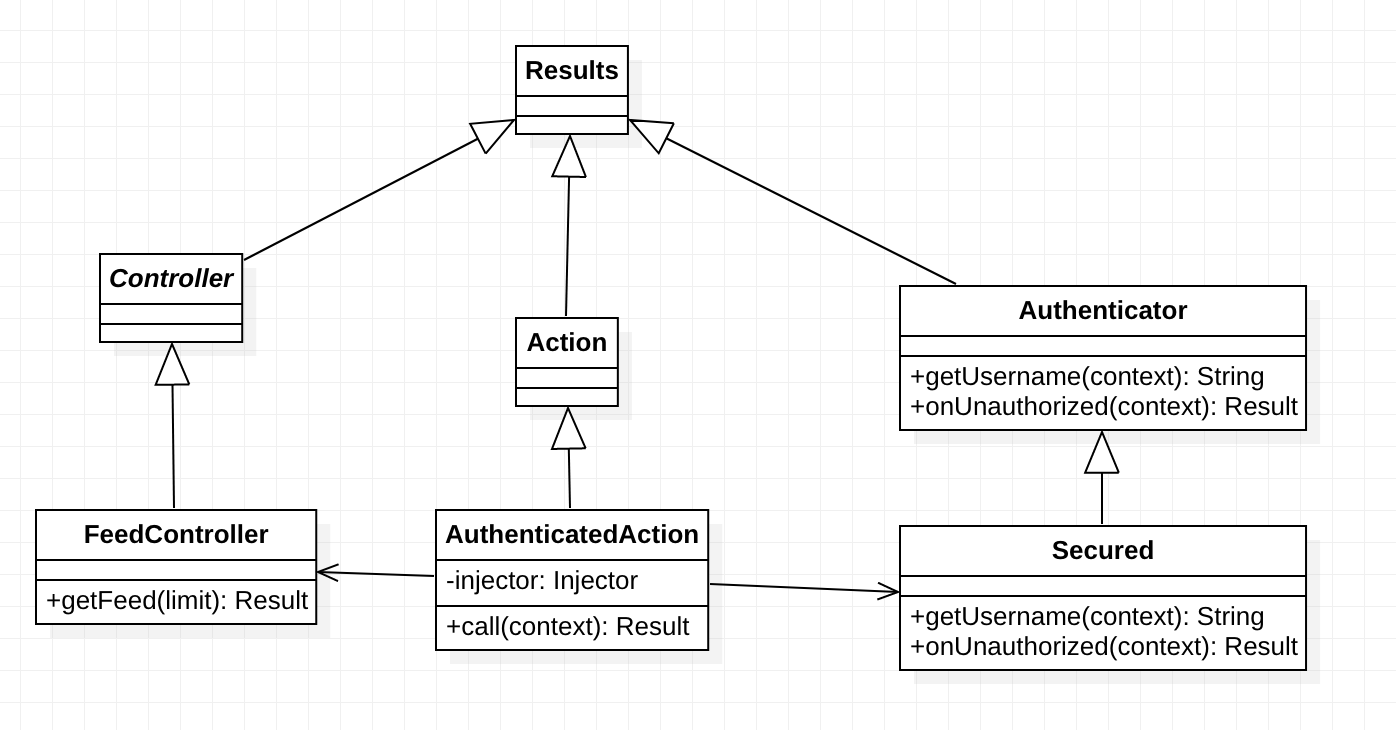
## Prototype

Consider the registration form creation as a costly process. We can avoid the process of creation of forms from scratch by simply cloning the created form whenever is required. we can create different types of forms and store it in Map and return those forms stored in map whenever queried.

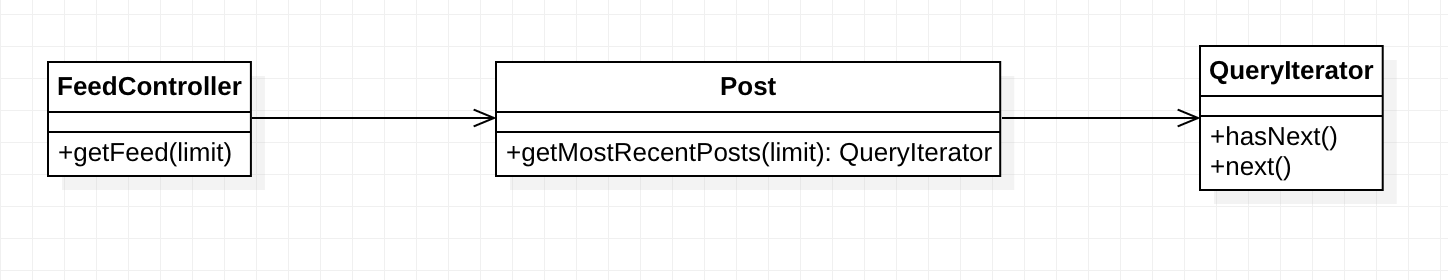


## Decorator

Decorator pattern is used to enhance the original function without changing its code. Many APIs in the system require that the user who initiates the request is already logged in. Here we use the decorator pattern to enhance the behavior of each controller at runtime by injecting the Secured class.

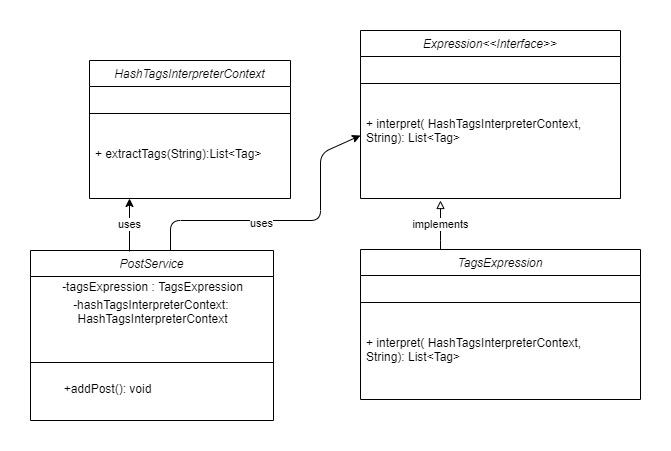
Iterator

## Iterator

Iterator pattern is used to sequentially access the elements of a collection. In the Post model, we expose the getMostRecentPosts(limit) API that returns a QueryIterator which is a built-in iterator class in Play and Ebean. When the FeedController tries to retrieve the most recent posts, it will loop through the iterator and visit elements until reaching the limit

## Interpreter

Interpreter pattern is used to define a grammatical representation for a language and provides an interpreter to deal with this grammar. In our scenario we are using it to extract hashtags from a given string. PostService(Client) invokes an object of expression to interpret a given string. If the string has ‘#’ then it calls HashTagsInterpreterContext(engine) to parse the string and get tags.



# 

# 6. Tutorial

In this section provides information about the project setup and how to use Ask.Me platform by providing step by step information.

## How to use the Application

Here in this section we will discuss how to set up the project to start using the application.

### 1. Download and Install required softwares

### Java



Download and install JDK 1.8 for the OS type

<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>

### Intellij IDEA



Download and install Intellij IDEA Ultimate Edition 2019.3 and above

<https://www.jetbrains.com/idea/download/#section=windows>

### Node.js

Download and install NodeJS appropriately for the OS

<https://nodejs.org/en/download/>

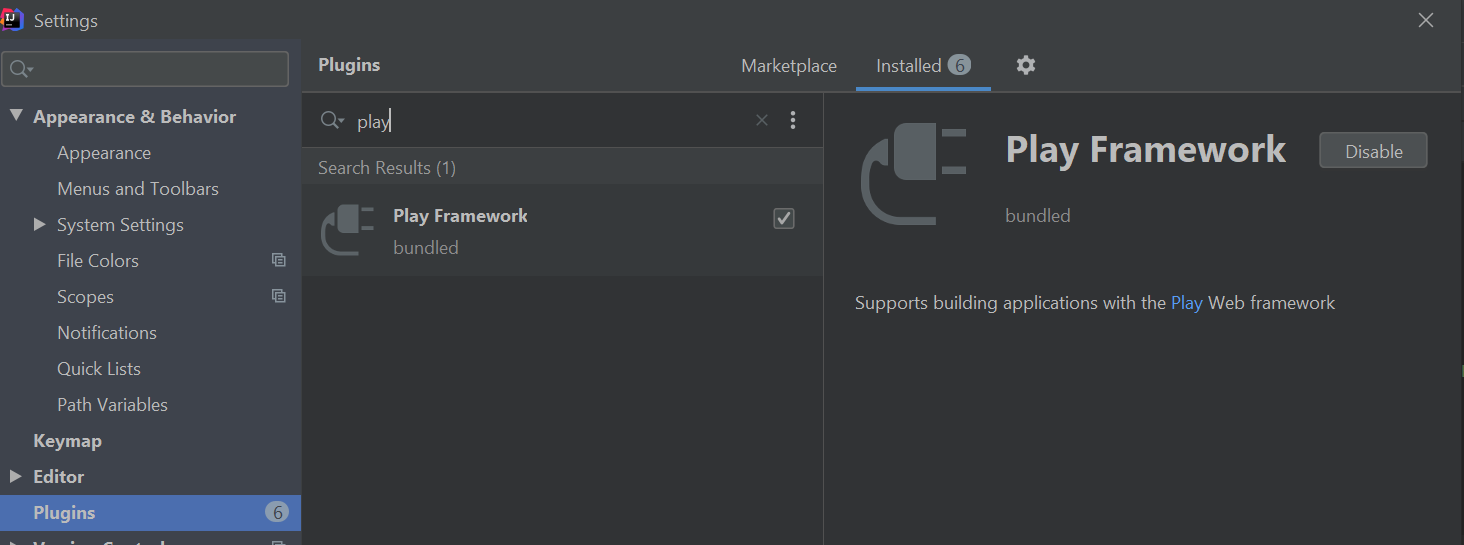
### NPM

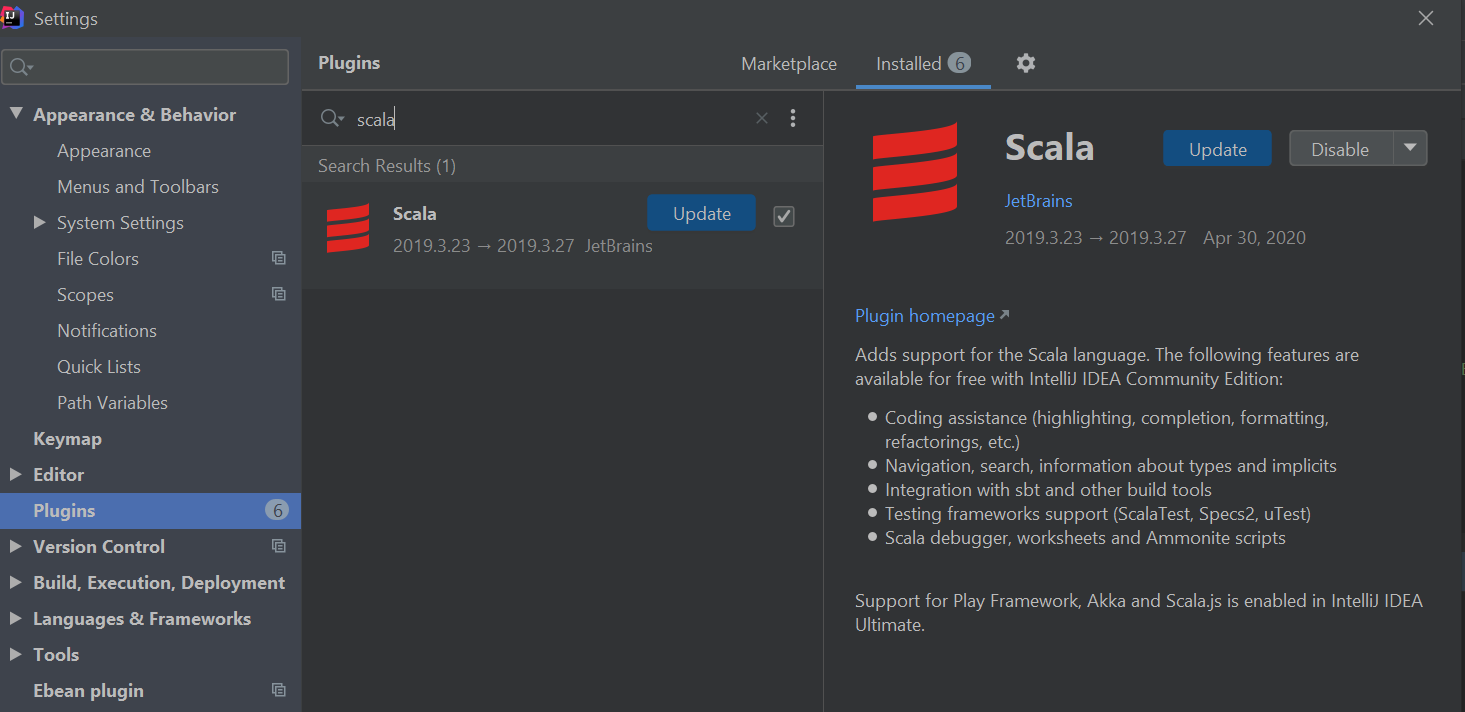
Download and install NPM appropriately for the OS

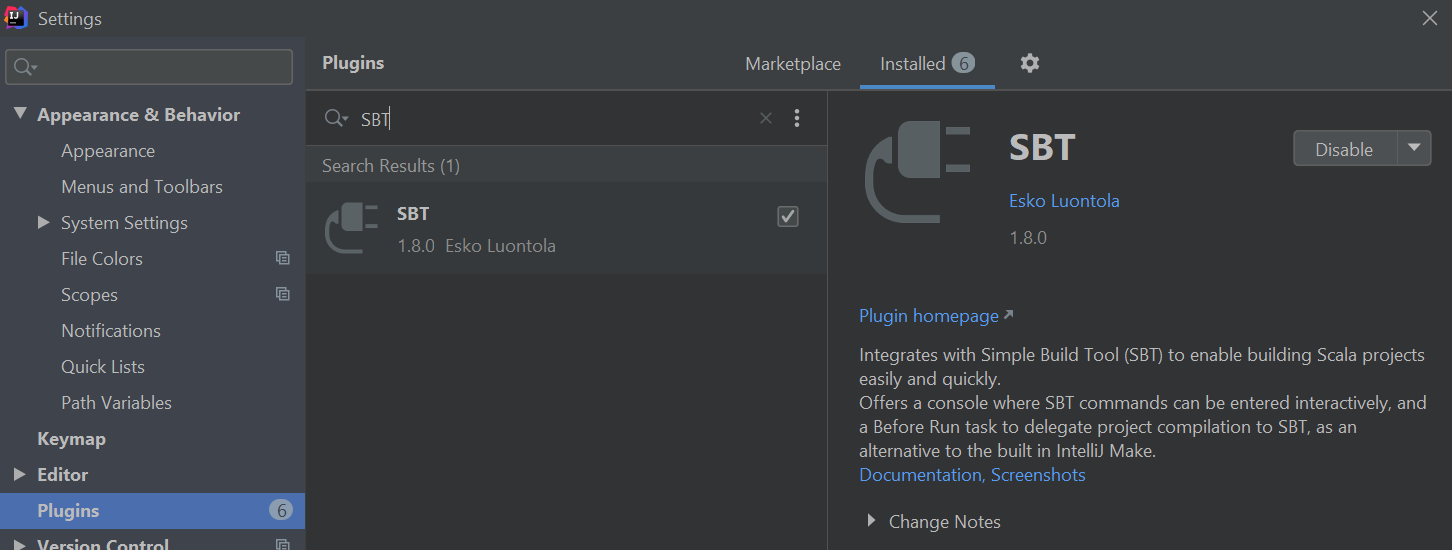
<https://www.npmjs.com/get-npm>

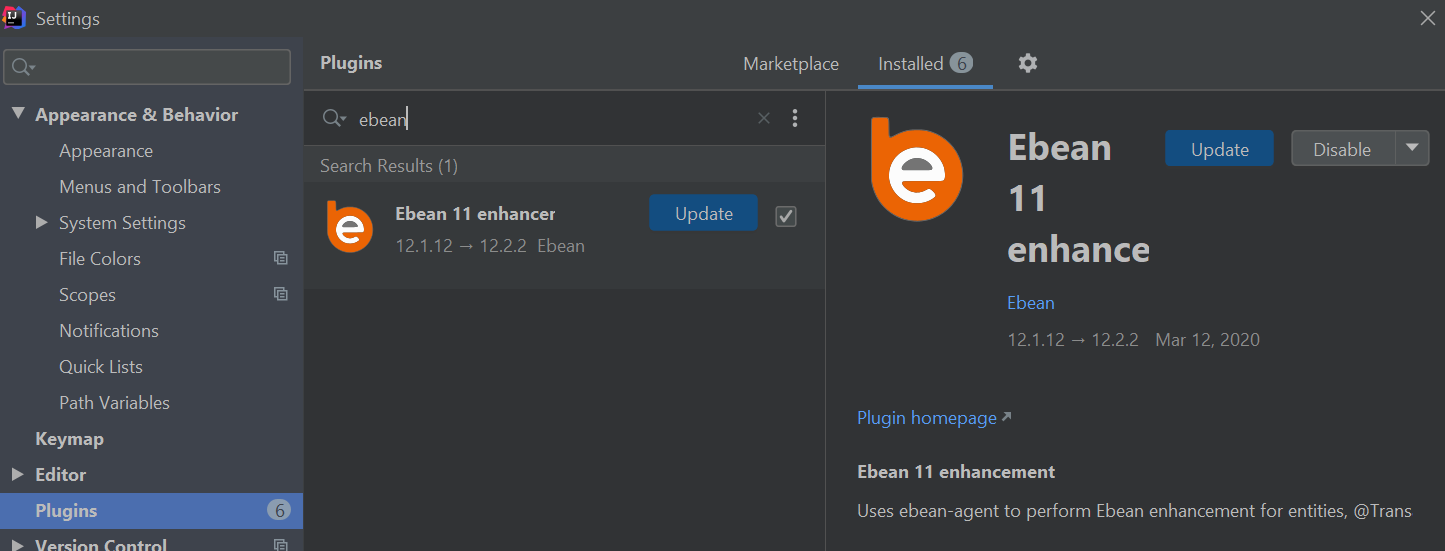
### Plugins & SDK

Open Intellij IDEA, ***File->Settings*** and ensure that the following plugins are installed.

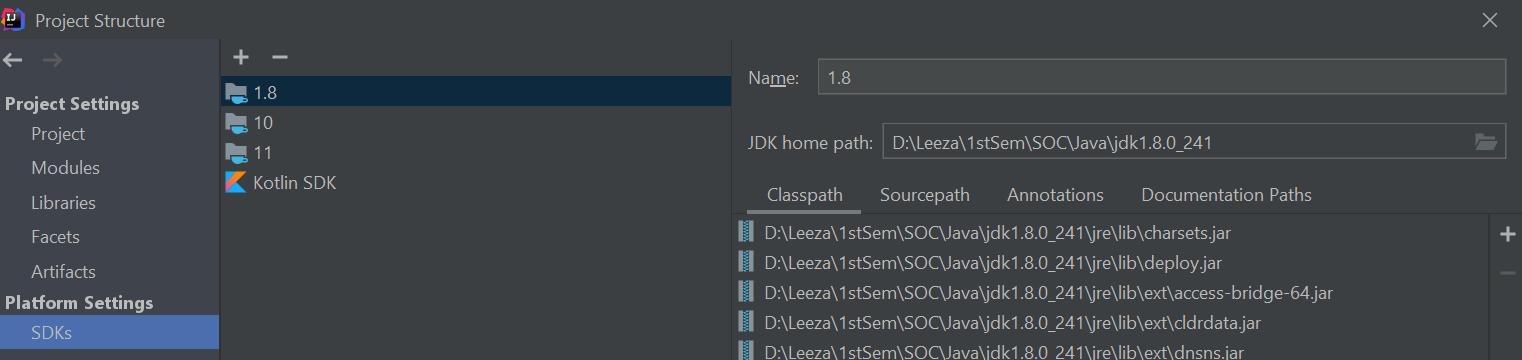








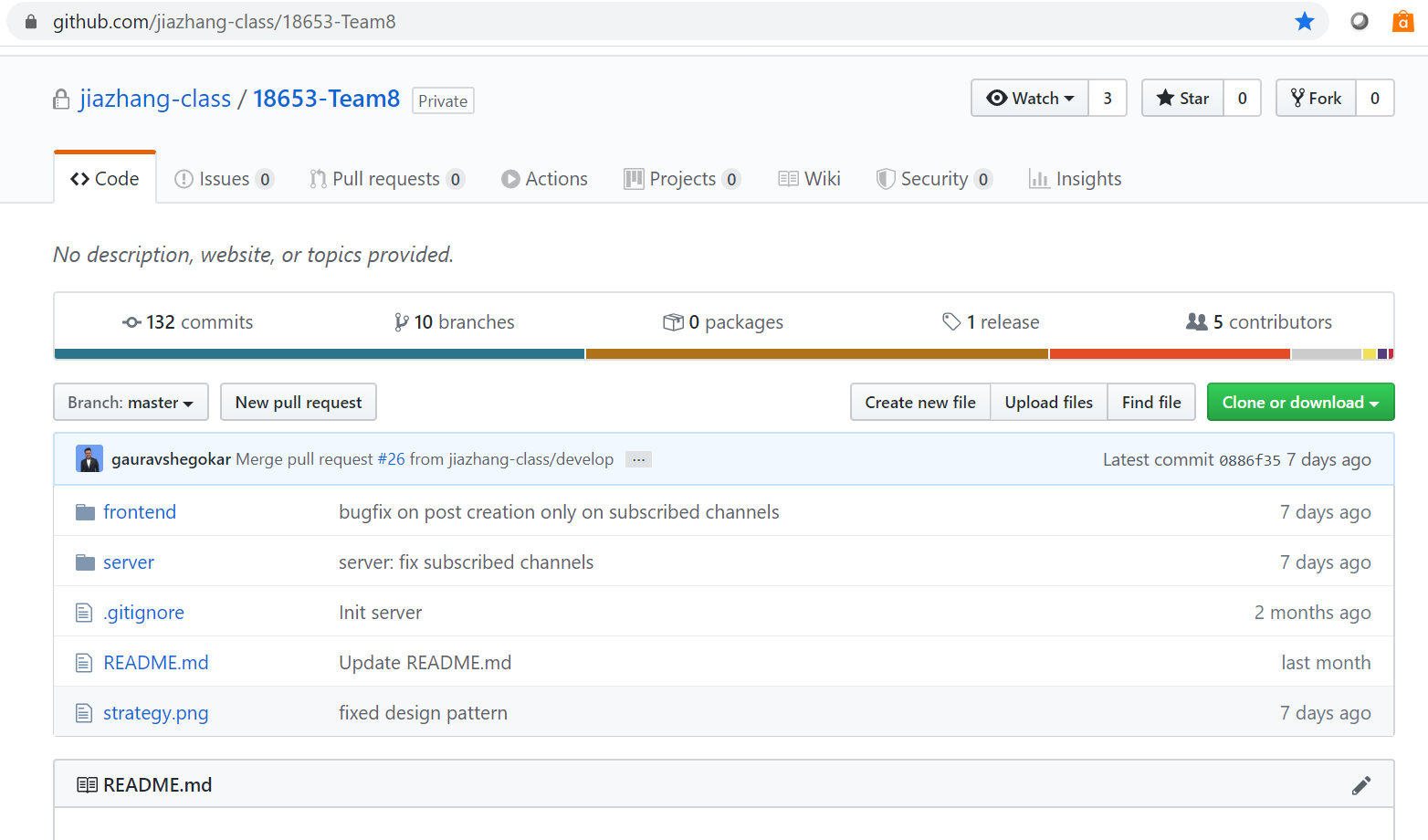
Ensure that sdk is appropriately set to 1.8, ***File->Project Structure***



### 2. Download/Setup & StartUp Application

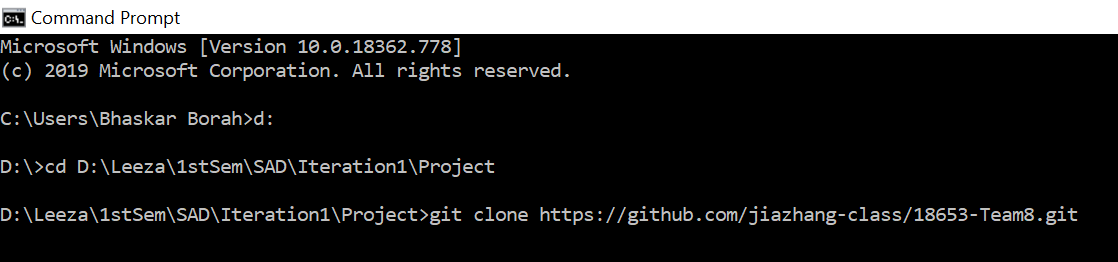
* + The website codebase is available at the following github location:

[https://github.com/jiazhang-class/18653-Team8](https://github.com/jiazhang-class/18653-Team8.git)

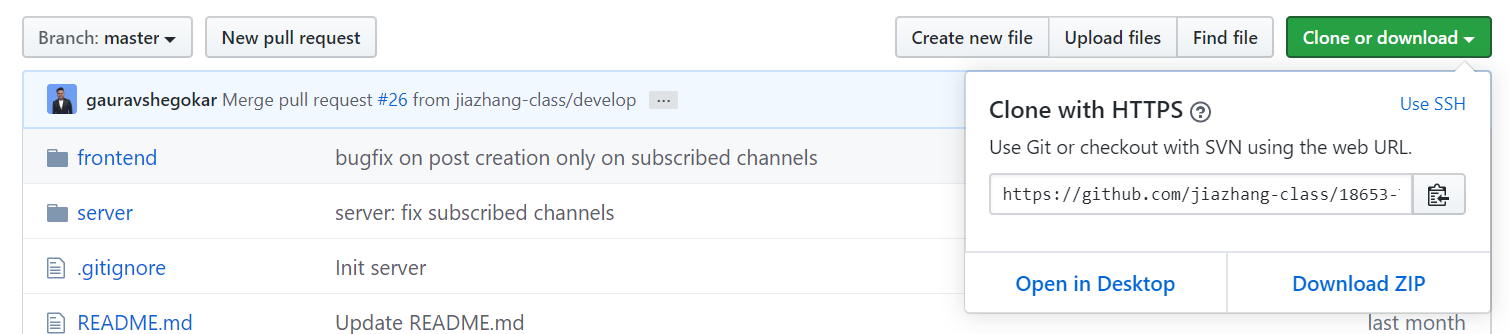


* + The codebase can be cloned via the git command line tool

**git clone** <https://github.com/jiazhang-class/18653-Team8.git>

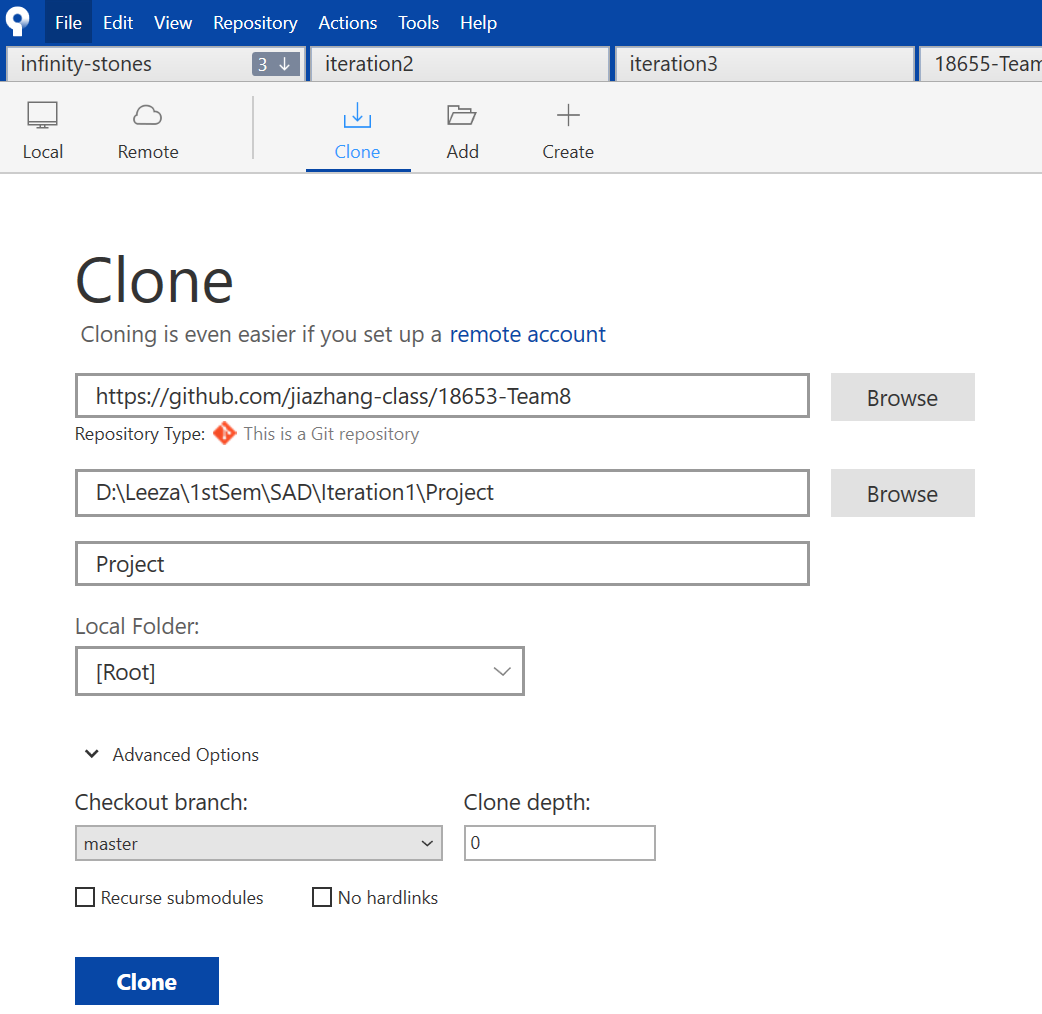


* + Or via the browser

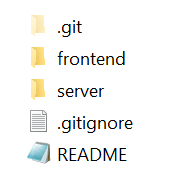


* + Or via any github desktop applications like SourceTree.

***File -> Clone/New***

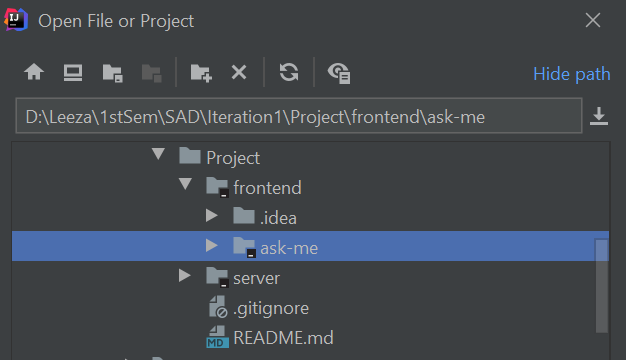


* + The codebase has two separate folders (projects) for frontend and backend named as follows

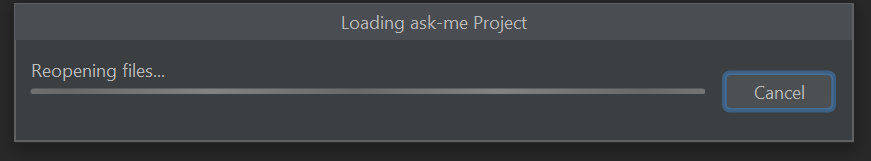


### 3. FrontEnd project set-up/startup

* + Open Intellij IDEA , ***File-> Open*** the ***frontend->ask-me*** folder

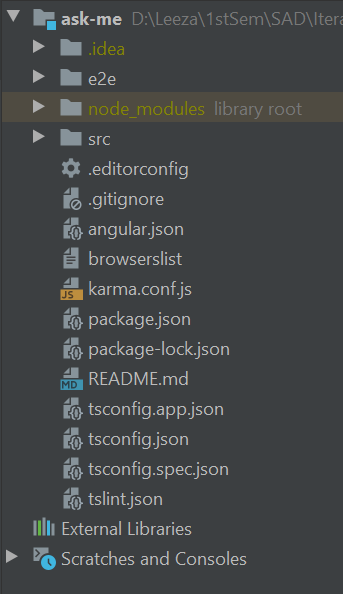


* + Wait a few moments for the project to load and indexing to complete



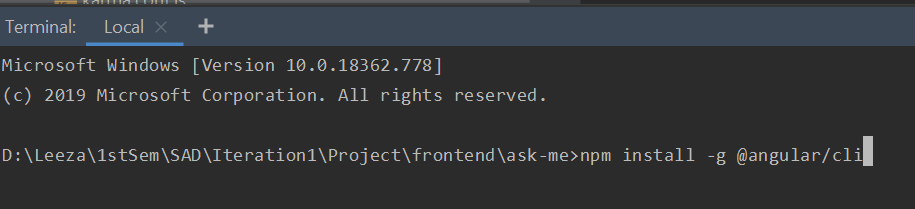


* + Once done, verify the project structure:



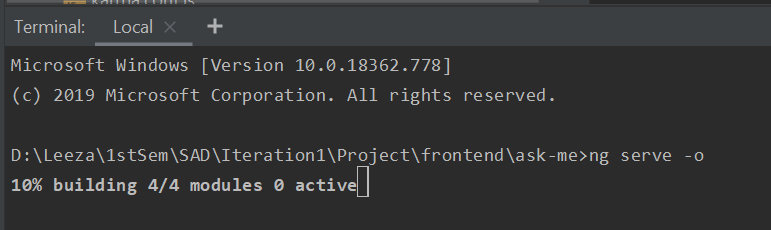
* + Navigate to ***ask-me*** folder on ***terminal*** (in INTELLIJ IDEA)and execute the following command to download and install the required project modules

***npm install -g @angular/cli***

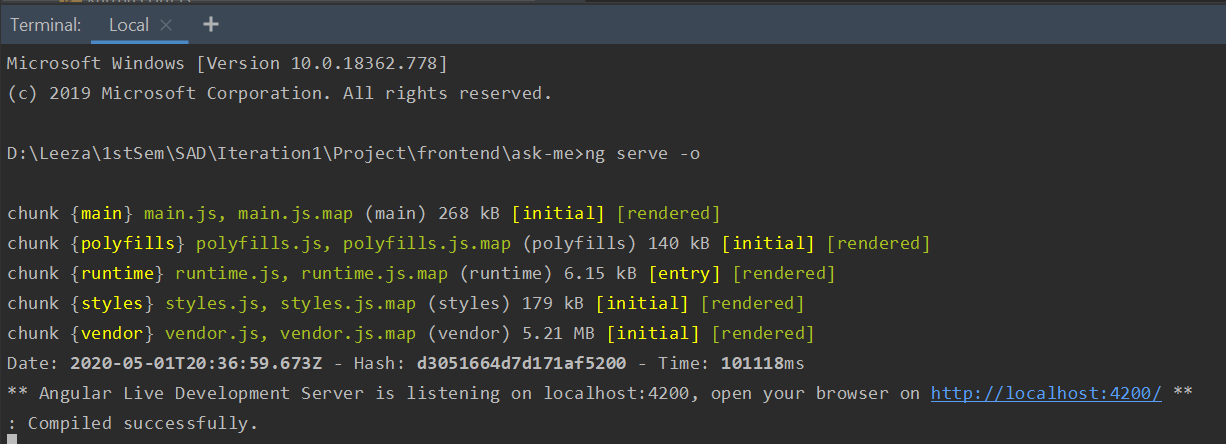


* + Once done, start the server by executing the following command from the same location:

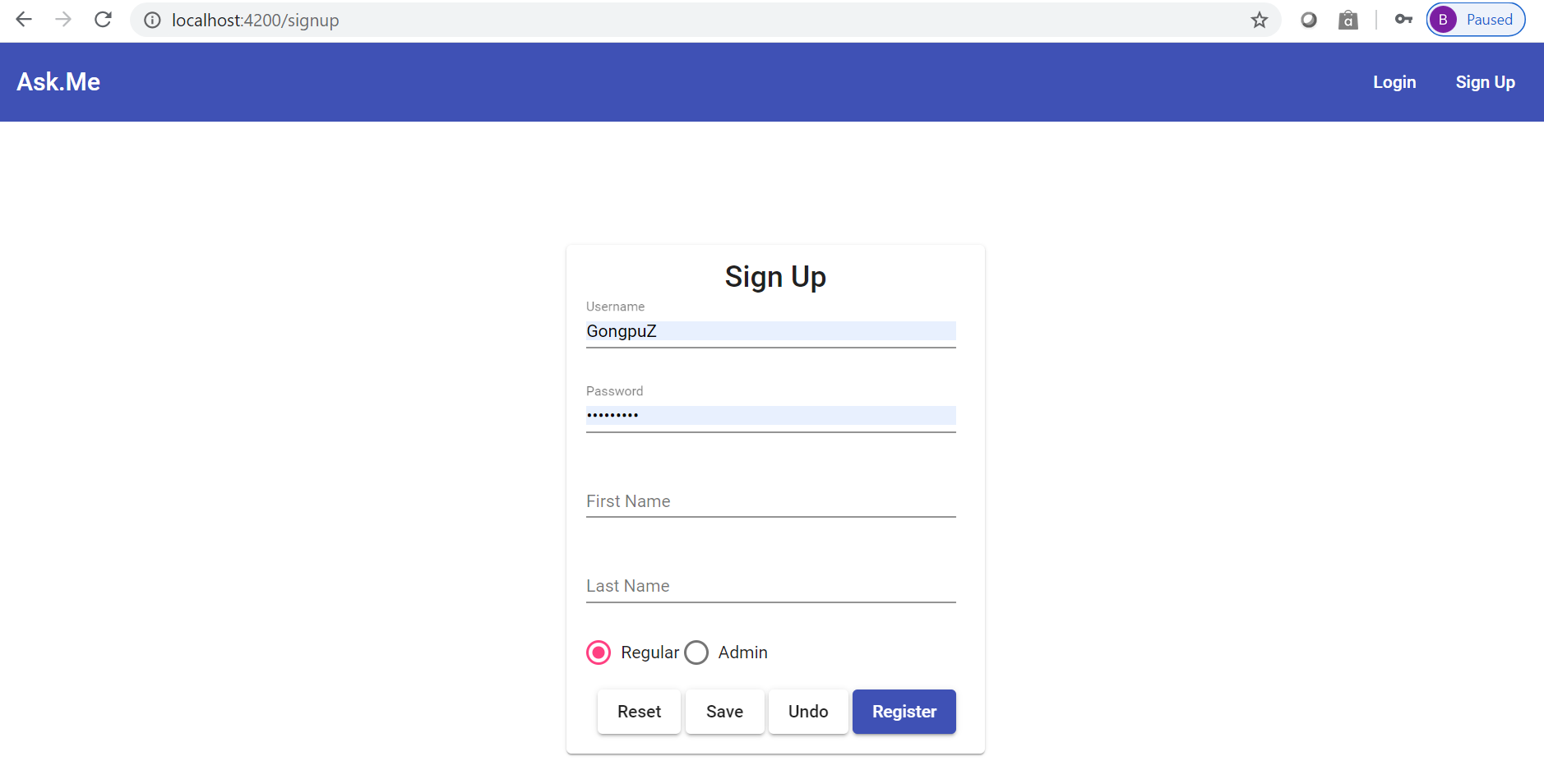
***ng serve -o***



* + This should take a few moments to compile and start the application.
  + The application startup can be monitored from the ***Run*** window:



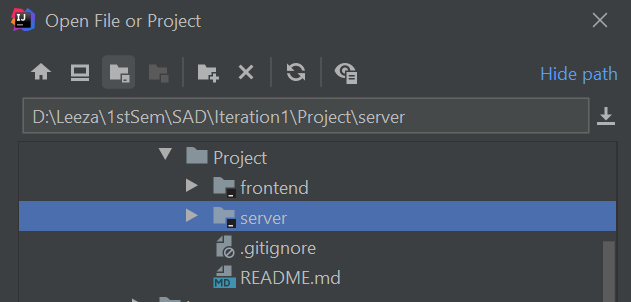
* + The home page of the application should load automatically in the default browser on the url :<http://localhost:4220>



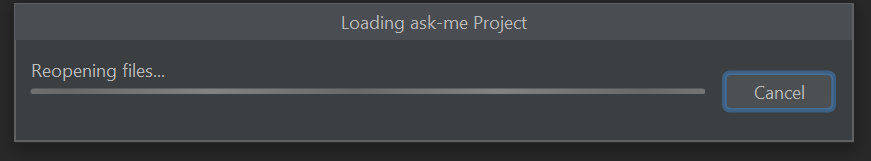
* + Do not start navigating the application yet because we still have the backend server to setup & start

### 4. Server (Backend) project setup/start

* + Open Intellij IDEA , ***File-> Open*** folder ***server***



* + This will take about a minute to load the project, index files, download/sync sbt dependencies etc. While this happens go ahead and create the database schema





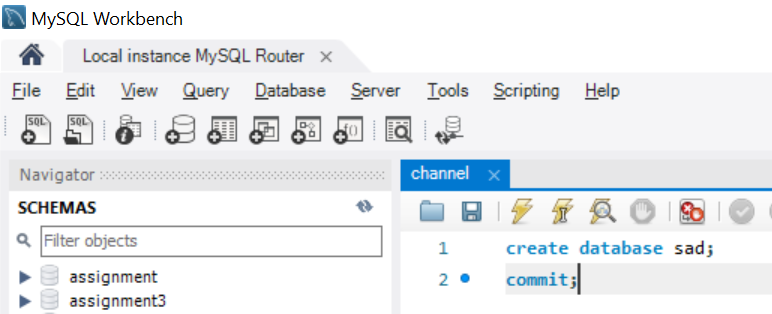




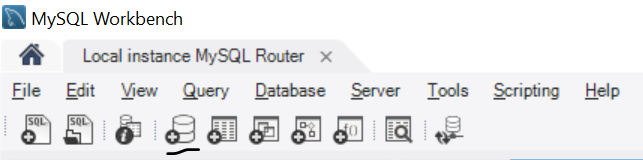
* + Create a database/schema in mysql
    - Can use mysql shell for database creation

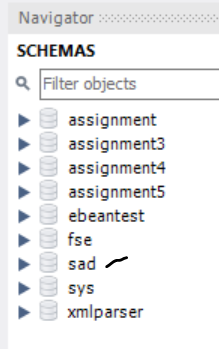
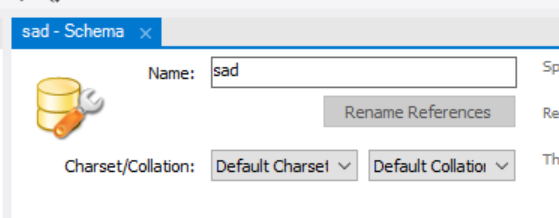
***CREATE DATABASE sad***

* + - Or can execute the command from MySql workbench command line tool

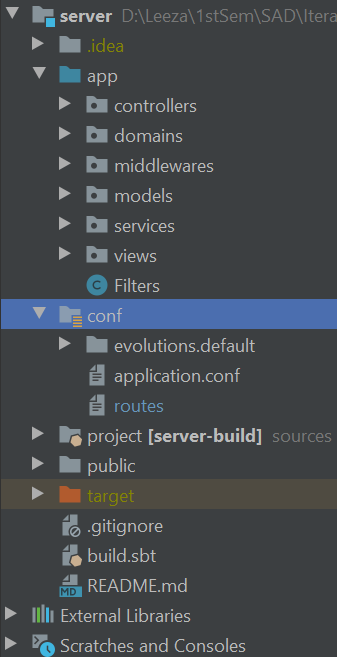


* + - Or create a new view from MySql workbench

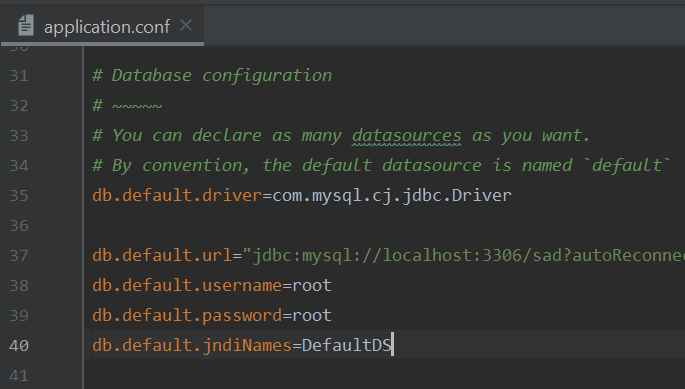




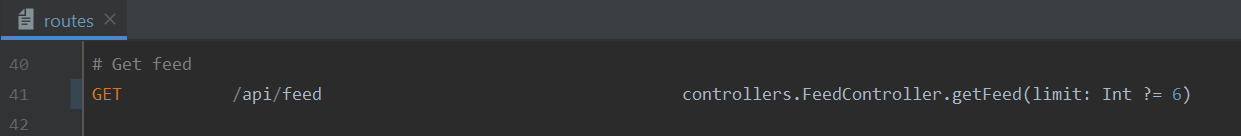
* + The project in Intellij should now be available for edit and configurations.



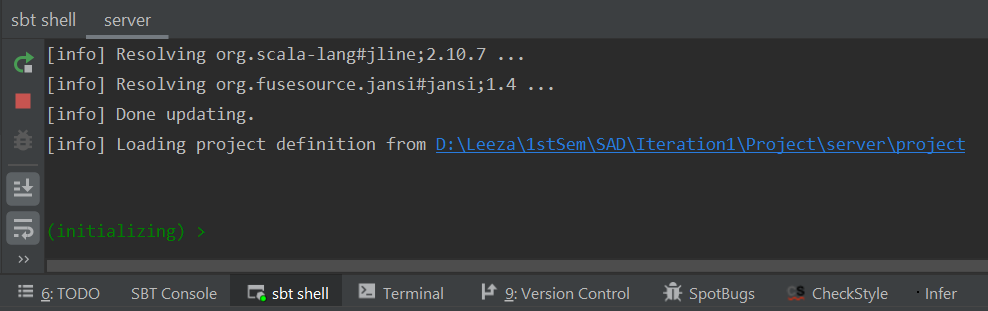
* + Open file ***conf/application.conf*** and update the database connection details and credentials to match the local environment



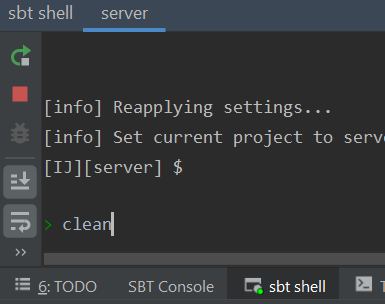
* + Goto file ***conf/routes*** andupdate the value of the following parameter to reflect the number of posts one wishes to be displayed on the feed page.



* + The application can be started in either of the following two ways:
    - SBT SHELL
      * Open SBT Shell and wait for a few moments to allow initialization

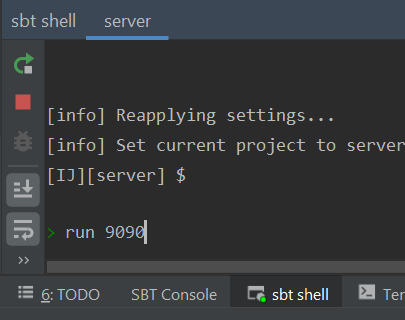


* + - * Run clean

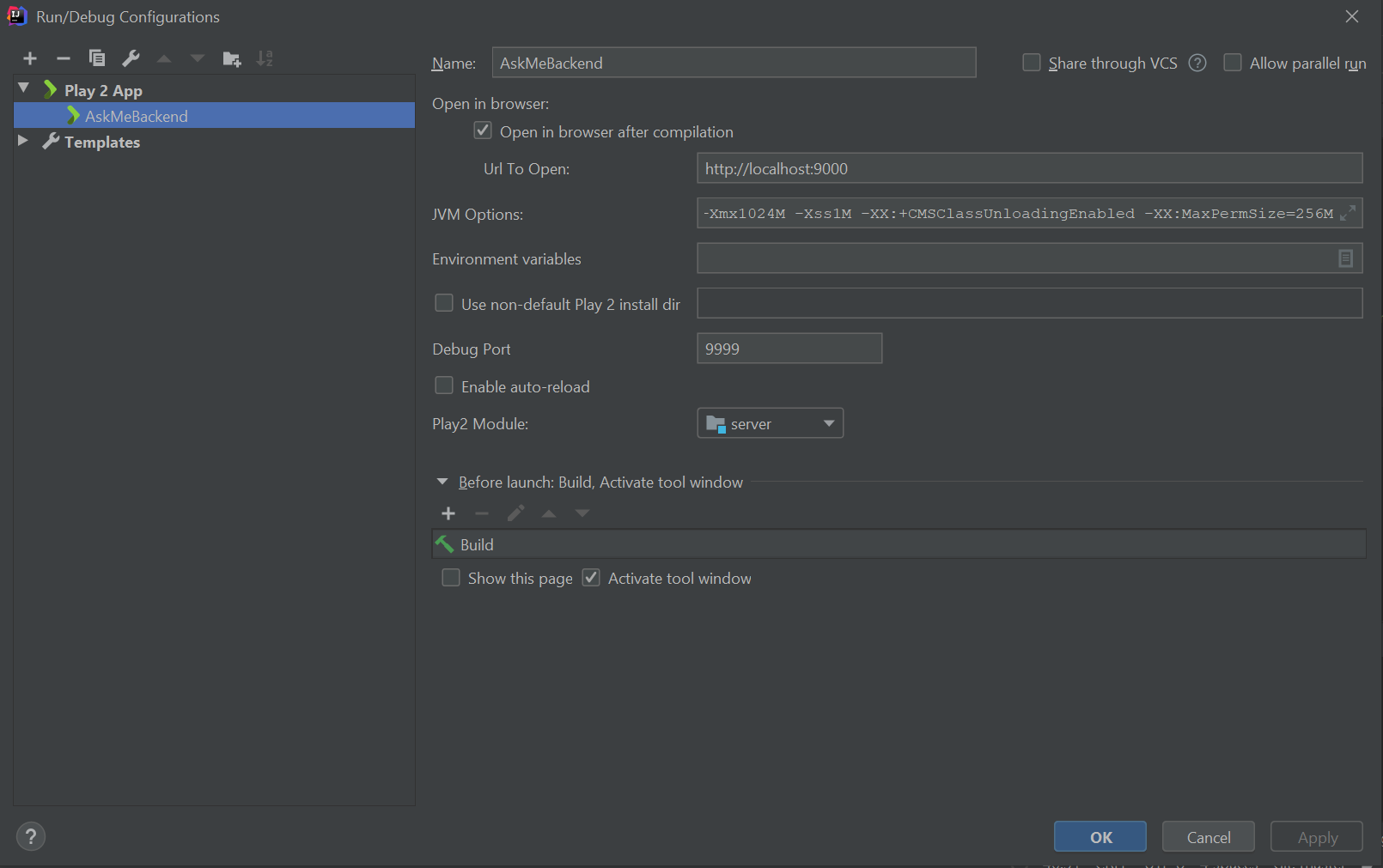


* + - * Once done, execute the following command

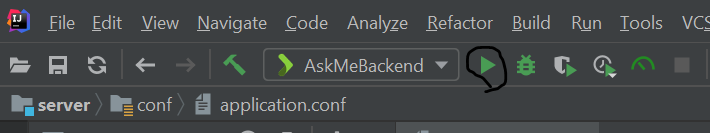
***run <desired-port>*** (recommended : 9000)



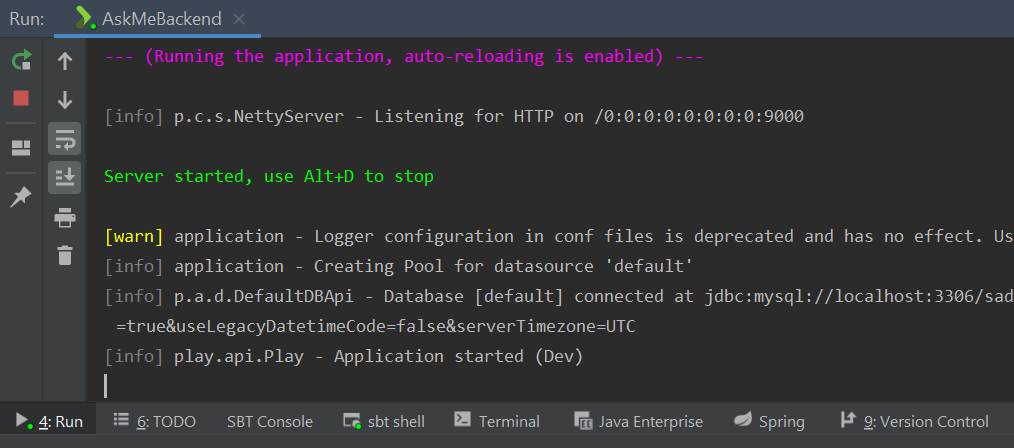
* + - RUN as PLAY APPLICATION
      * GOTO ***Run -> Edit Configurations***
      * Select ***+ -> PLAY APPLICATION***
      * Edit the following fields with the mentioned values



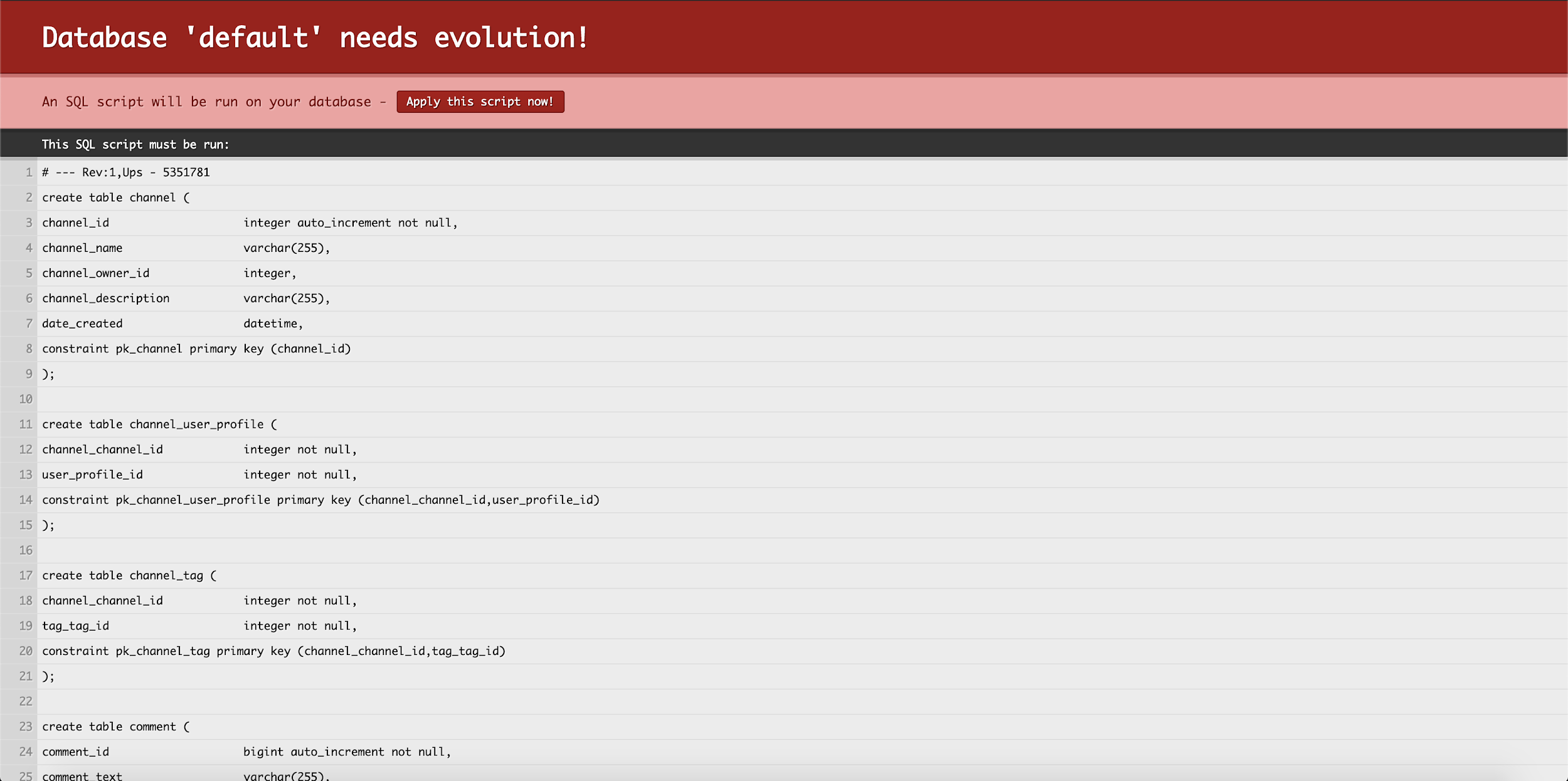
* + - * Click ***Apply***
      * Now, run the application by clicking the following button



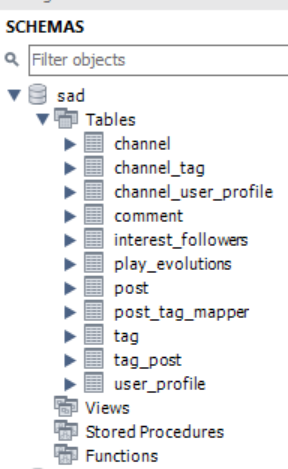
* + The application startup can be monitored from the ***Run*** windows :



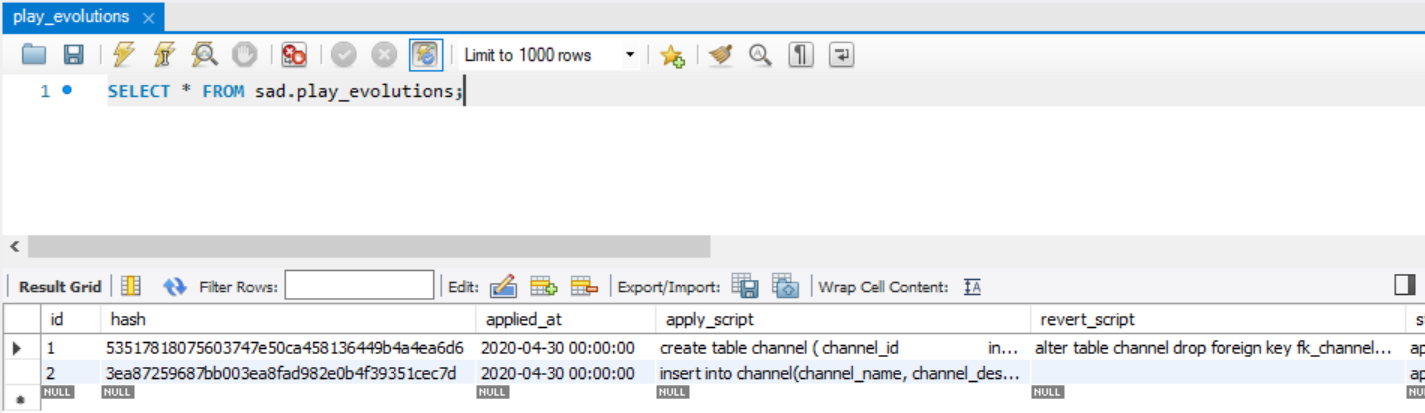
* + Somewhere during startup the default web browser will open up with the following page, prompting the user to “Execute Evolution”



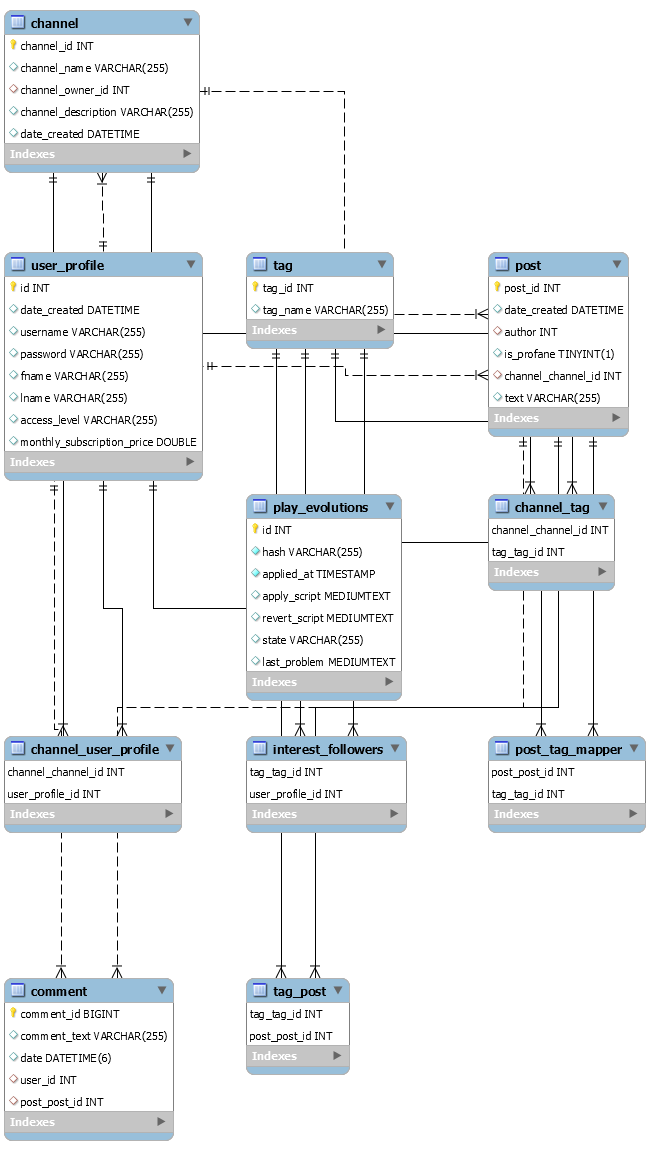
* + Click on the button to allow DB Evolution to execute, this might take a few moments.
  + Once done, verify the database to ensure that the following tables have been created.
    - Verify the tables



* + - Verify the data

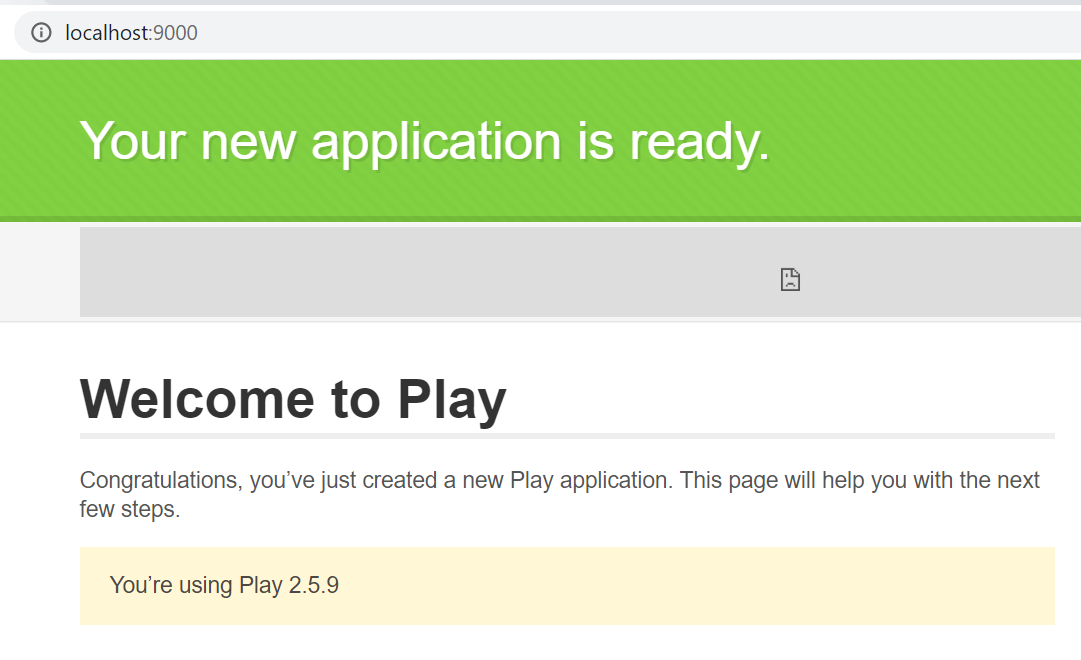


* + - Verify the ER Diagram



* + Once the DB evolution completes, the backend application will start on the browser with the following default page.

[http://localhost:9000](http://localhost:4220)



* + Navigate back to the front end application on the url

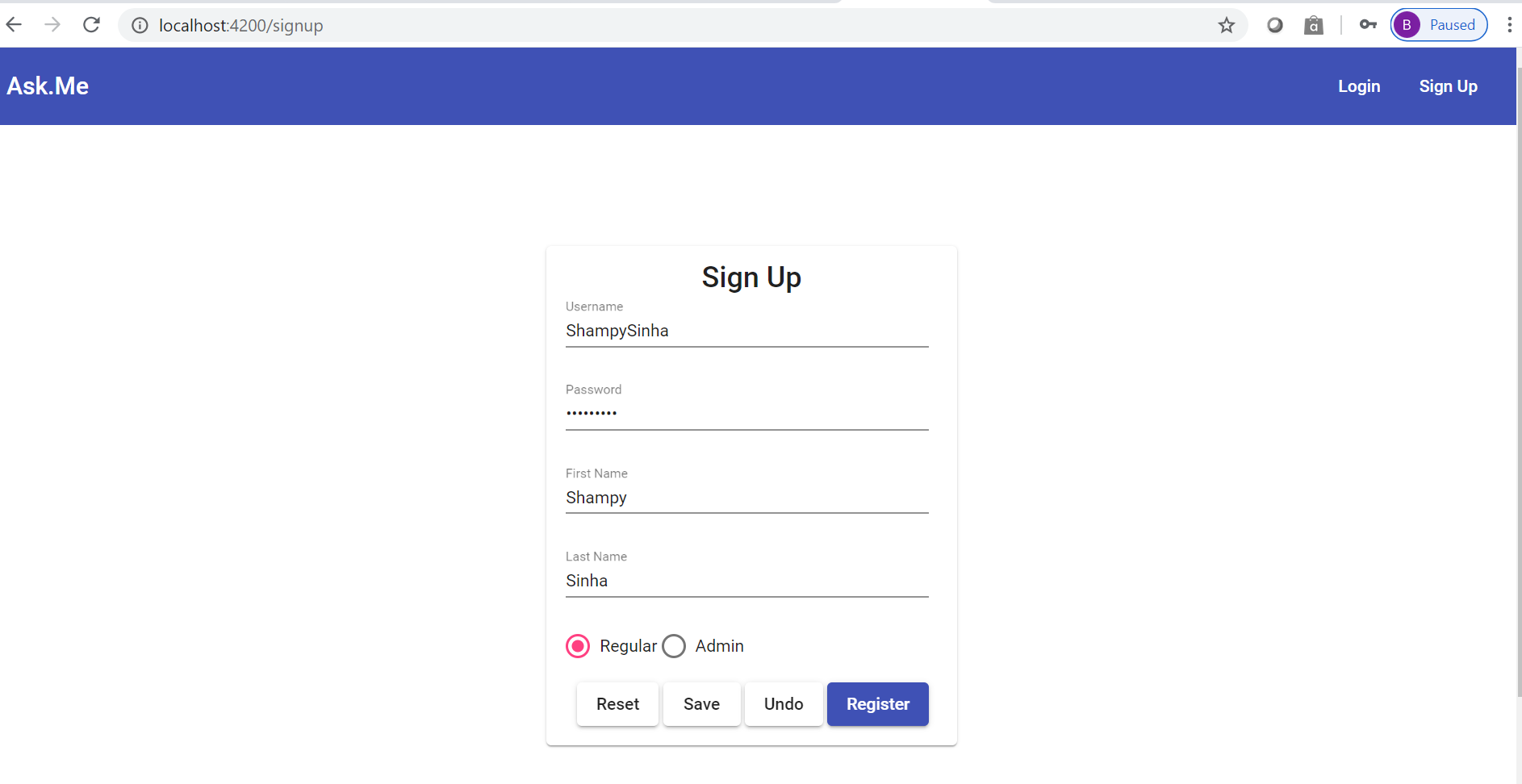
<http://localhost:4220>

## Application Demo/Execution steps

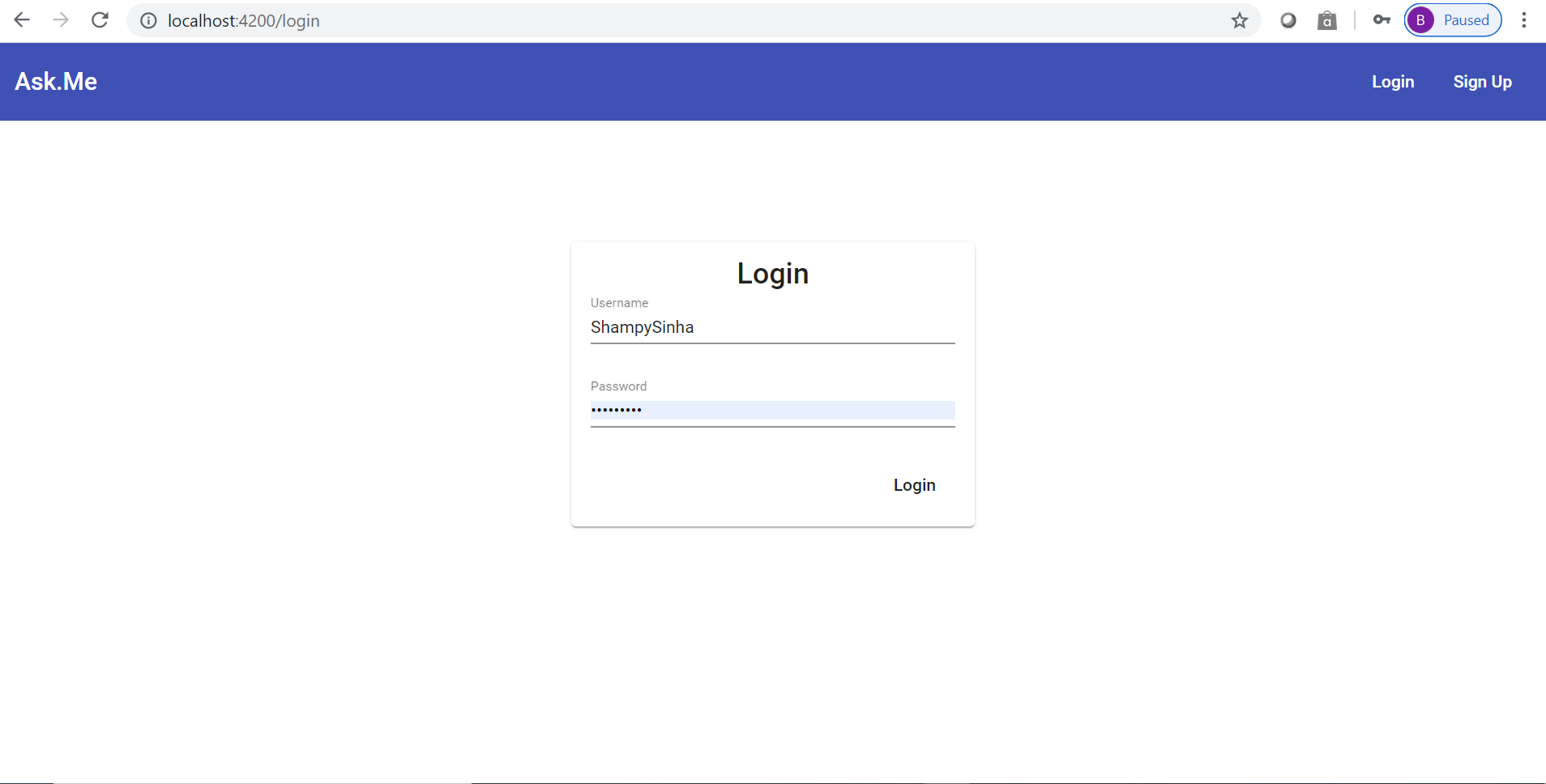
This part of the tutorial will provide you information on how to use the application, explaining the features of the application. We have made the tutorial demo as well, you can refer to the video. Start navigating and exploring as suggested in this Demo video. [AskMe Demo](https://www.youtube.com/watch?v=VvmQ9yZr9HQ)

Steps for application navigation

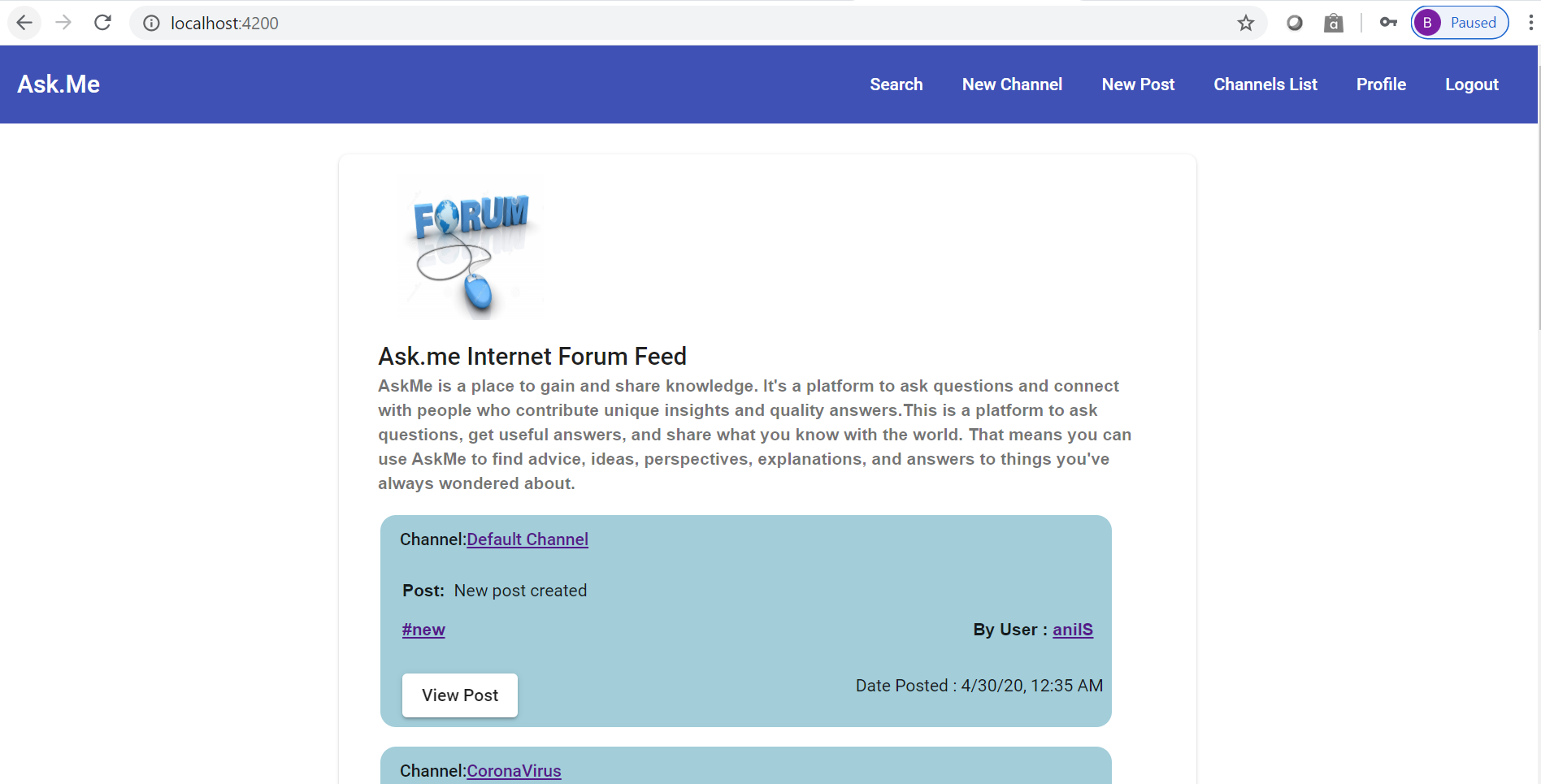
* + Register as a regular/admin user. Use the Save/Undo/Reset buttons to save the form information, undo and reset to an empty form



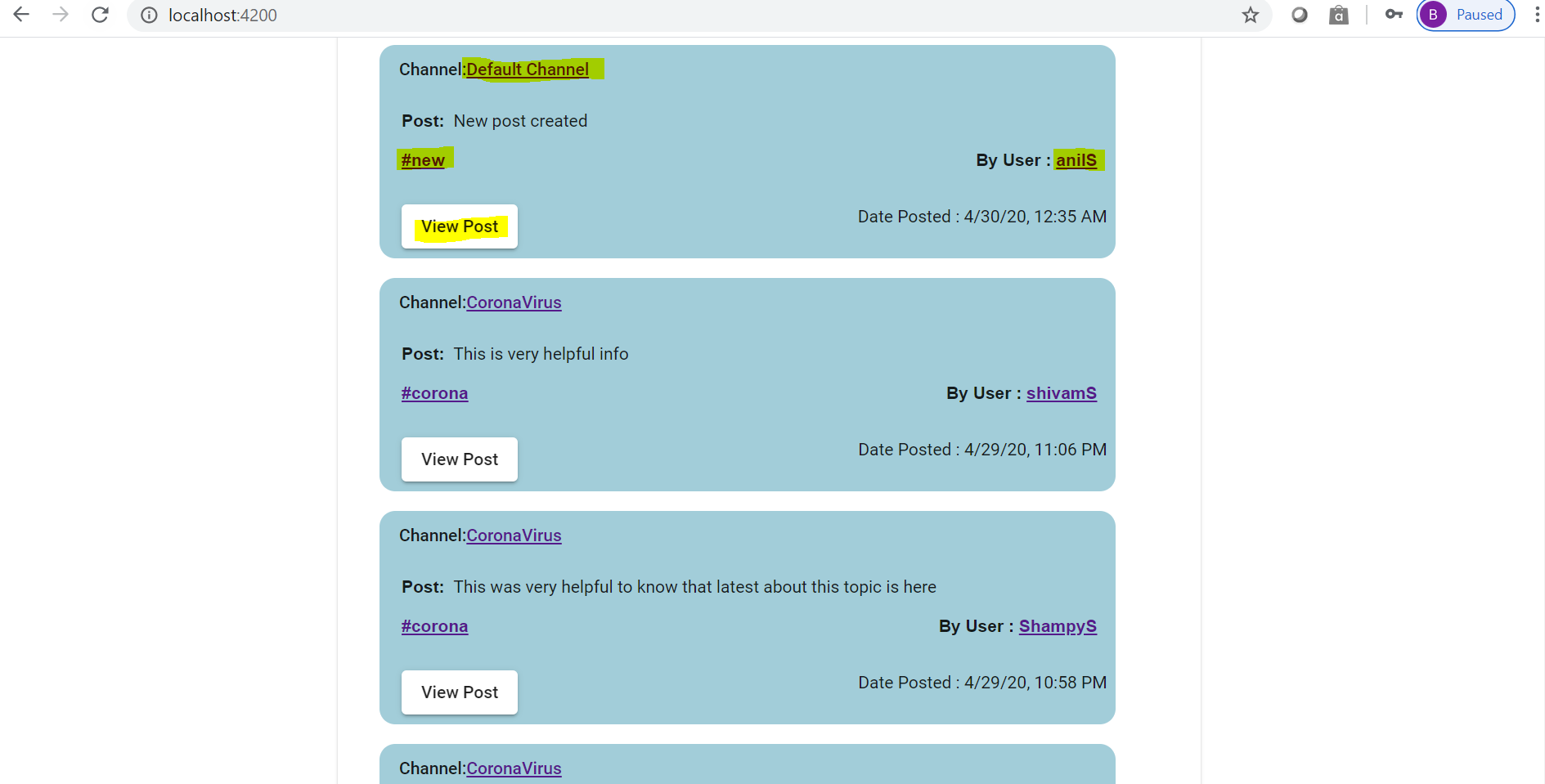
* + Login with the newly created user or one of the existing users



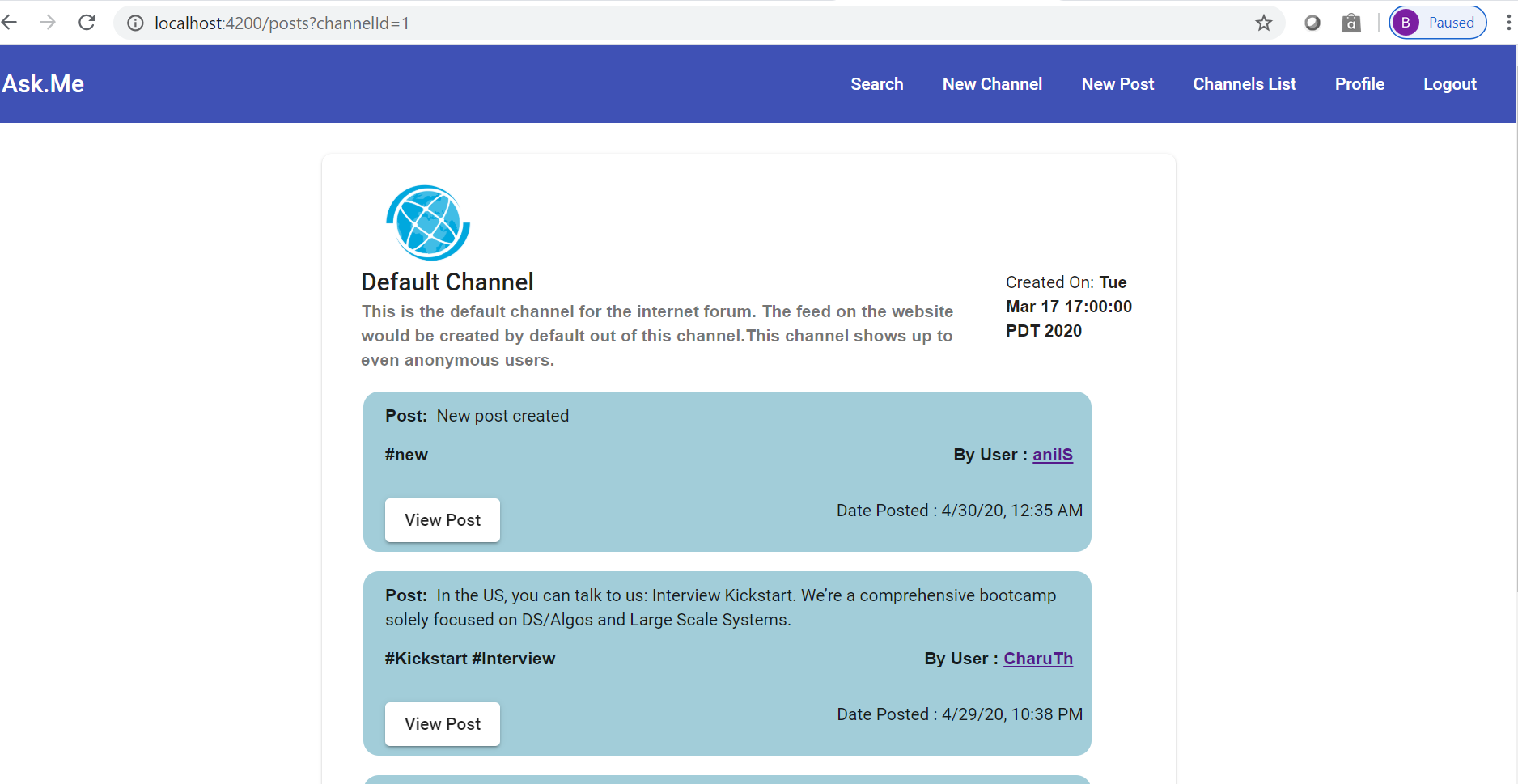
* + User lands on the feed page where he can see the the top most feed of the website.



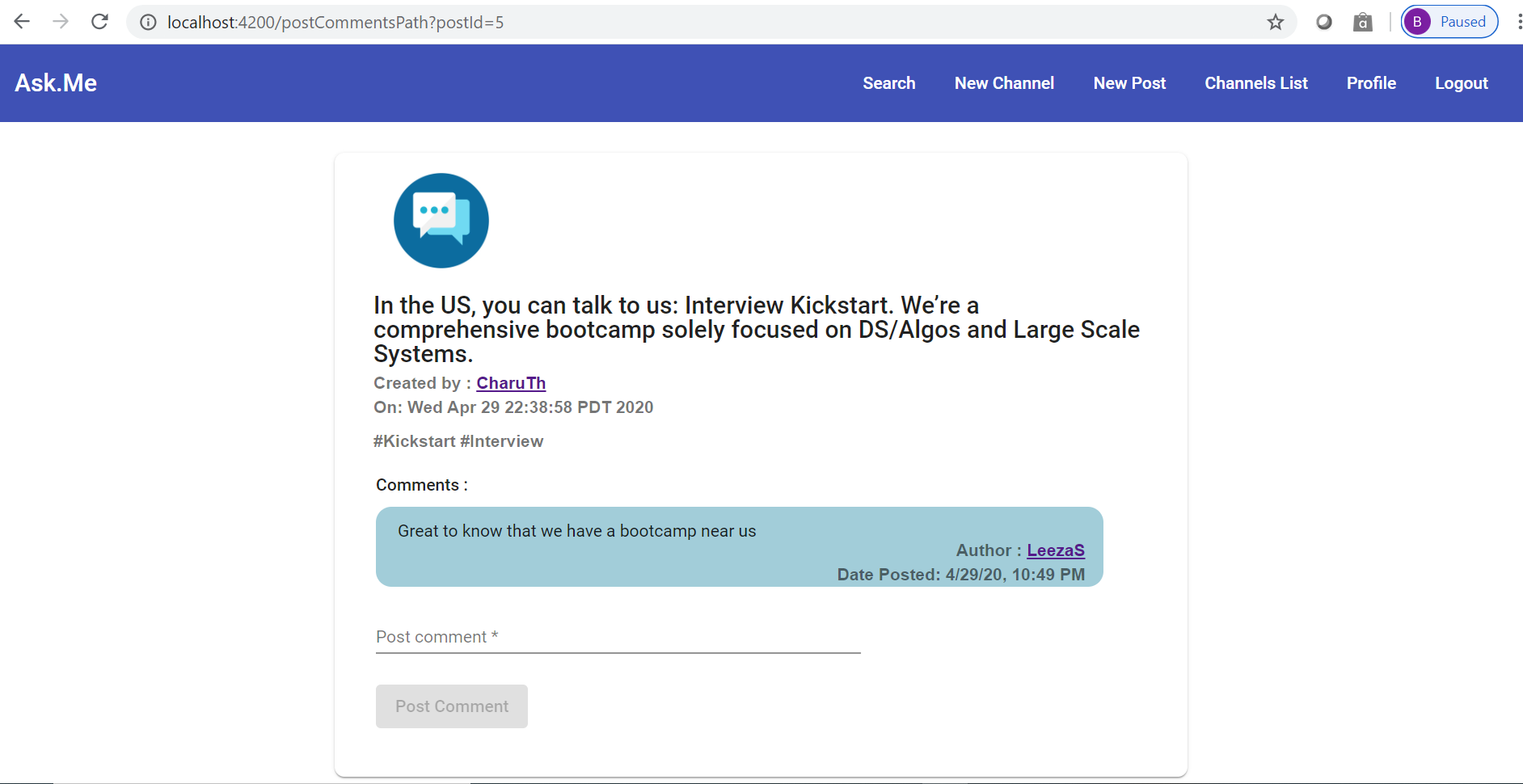
* + User navigates to the channel, hashtag, post or user by clicking on the below highlighted links.



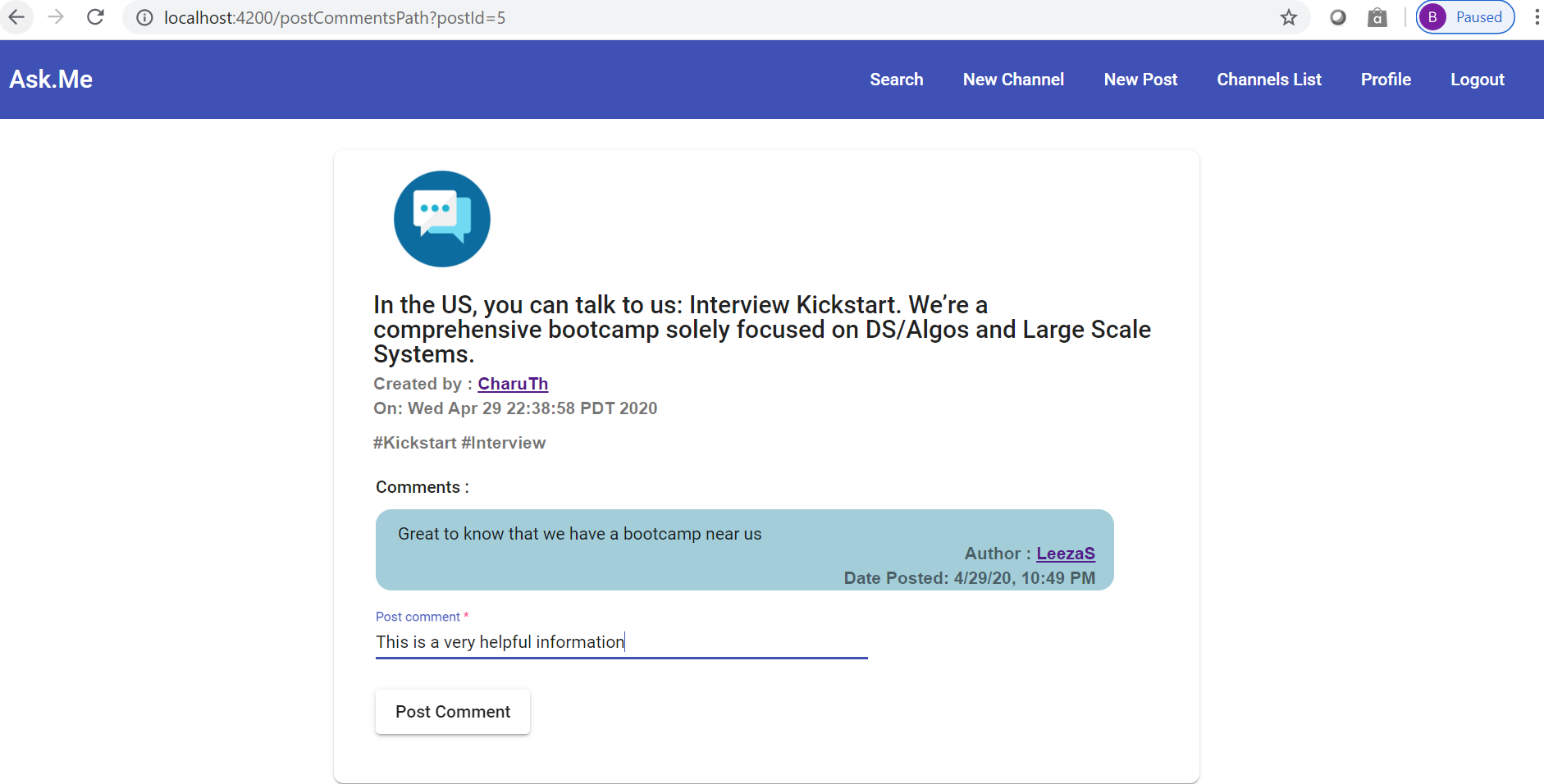
* + Channel Details page.



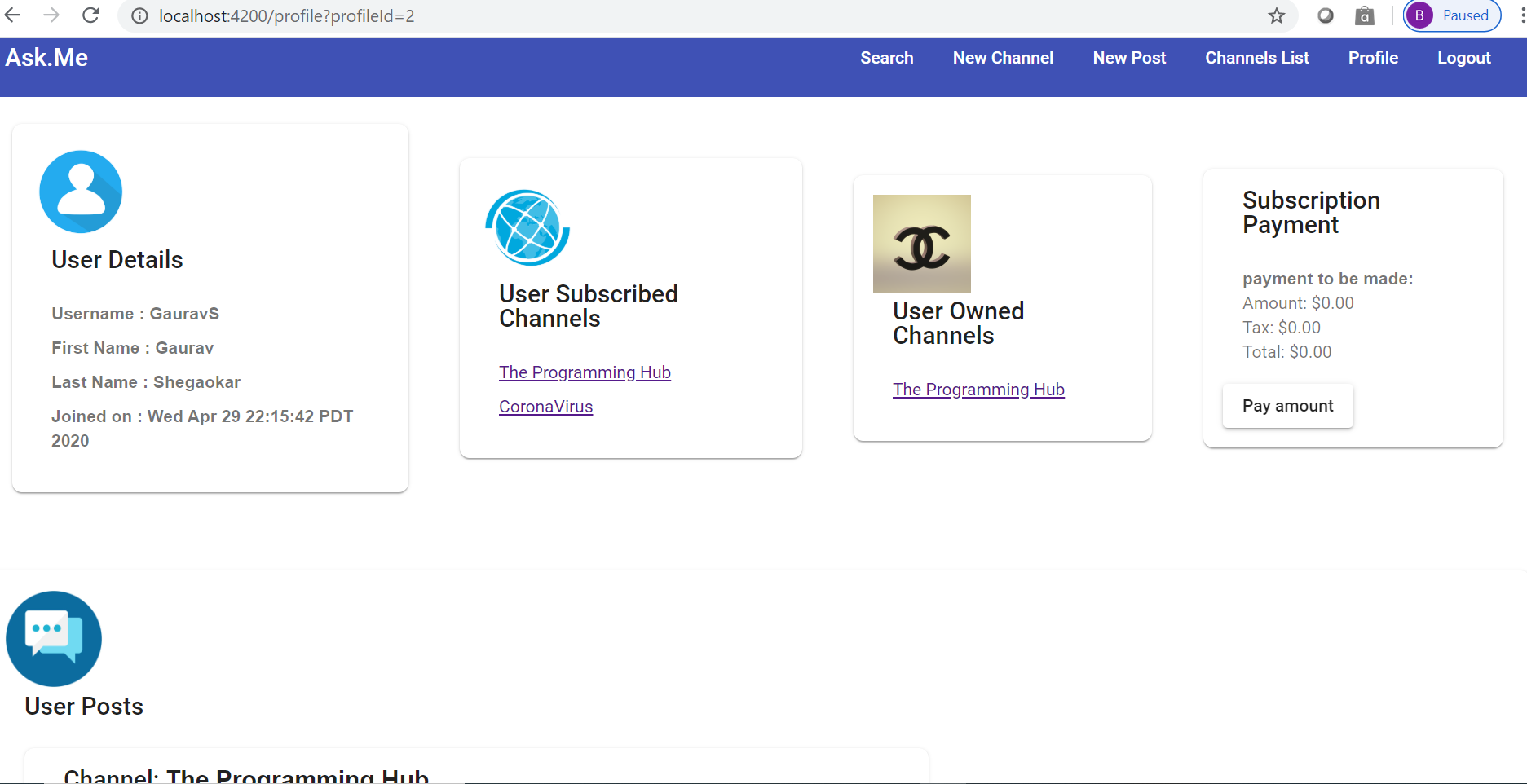
* + Post Details page



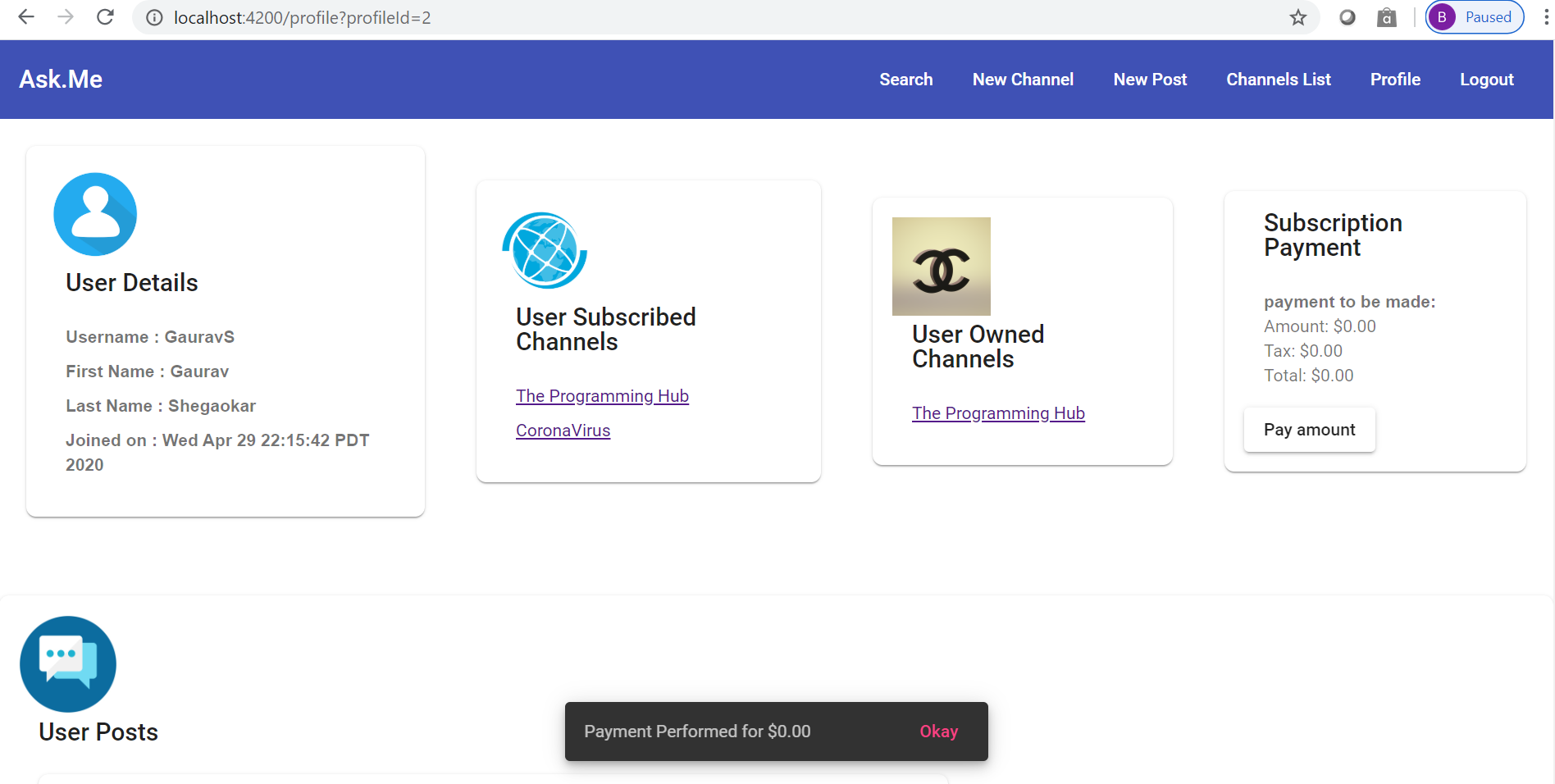
* + User chooses to add a comment on the post, which shows up immediately on submit.



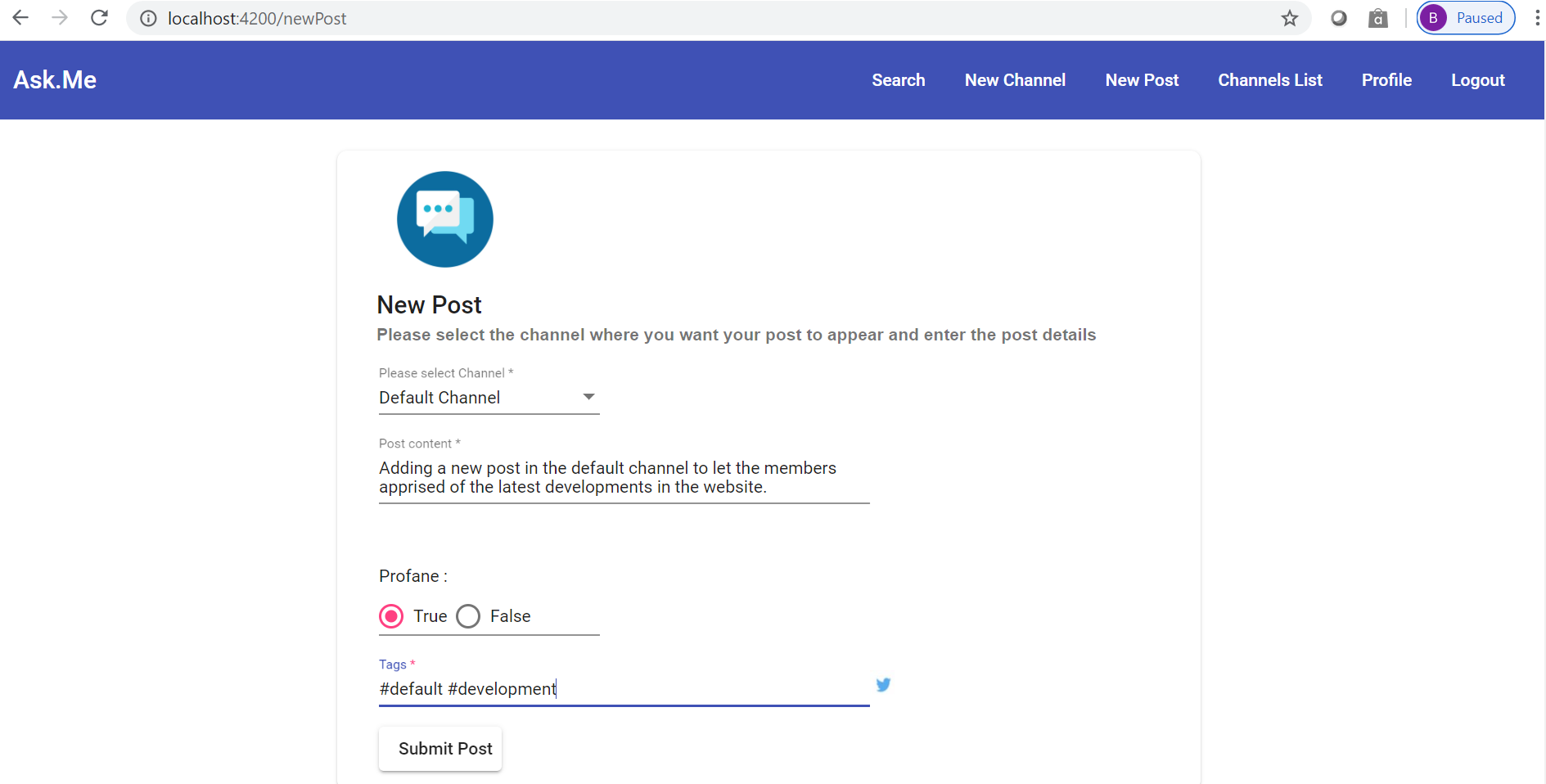
* + Profile details page.



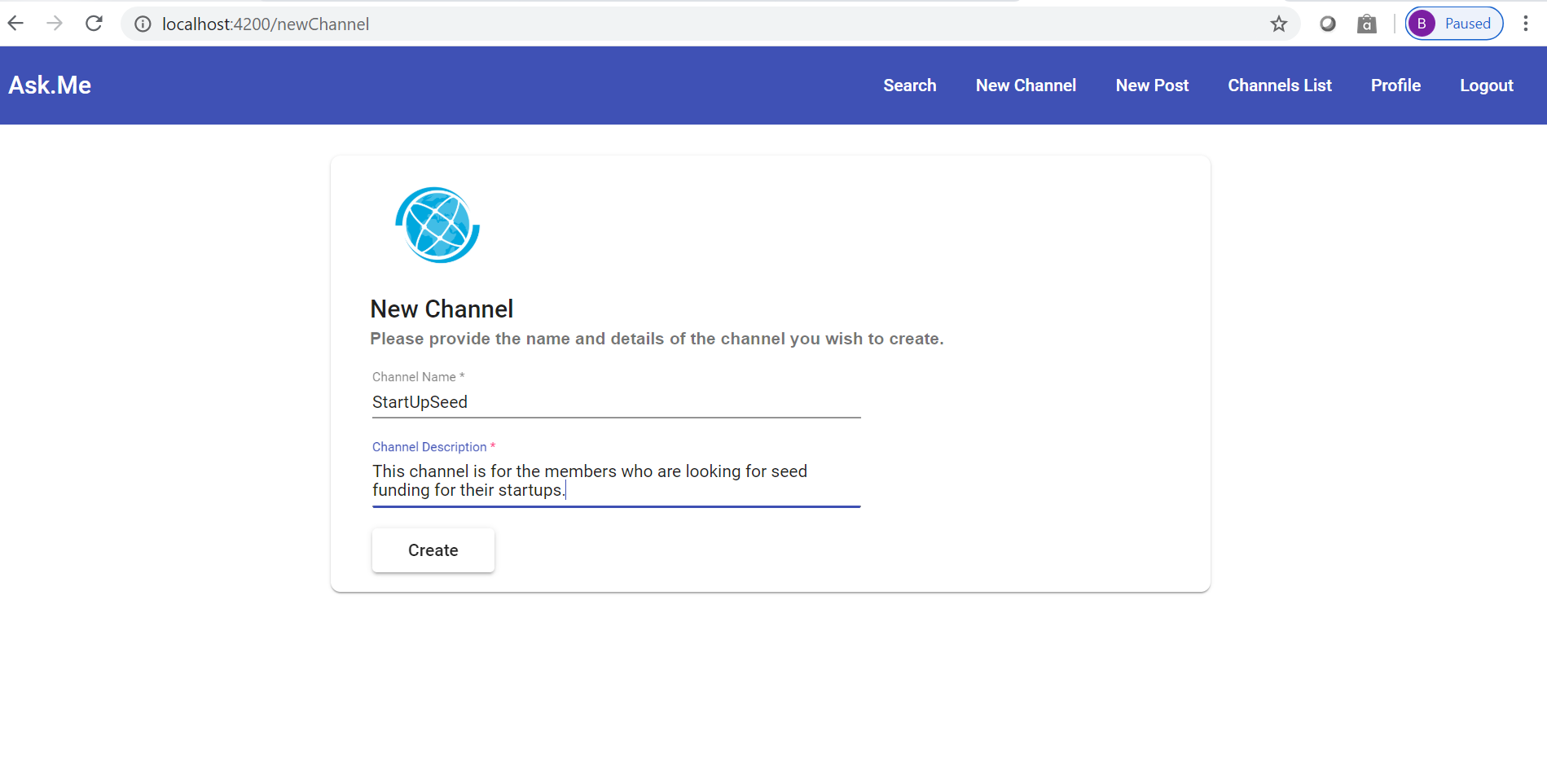
* + User clicks on “Payment amount” to trigger payment, pop up shows up on success.



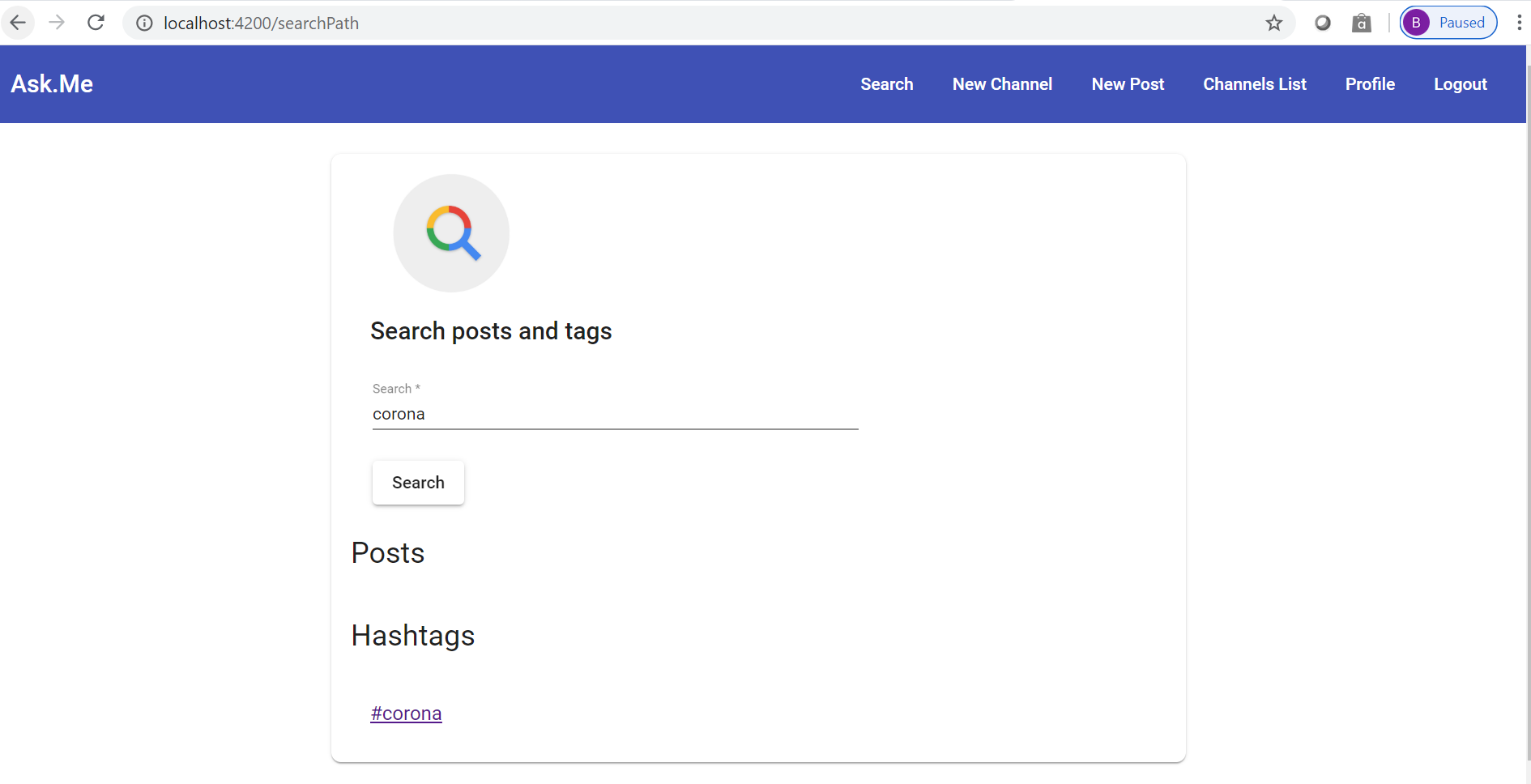
* + User creates a new post in any channel that he is subscribed to.



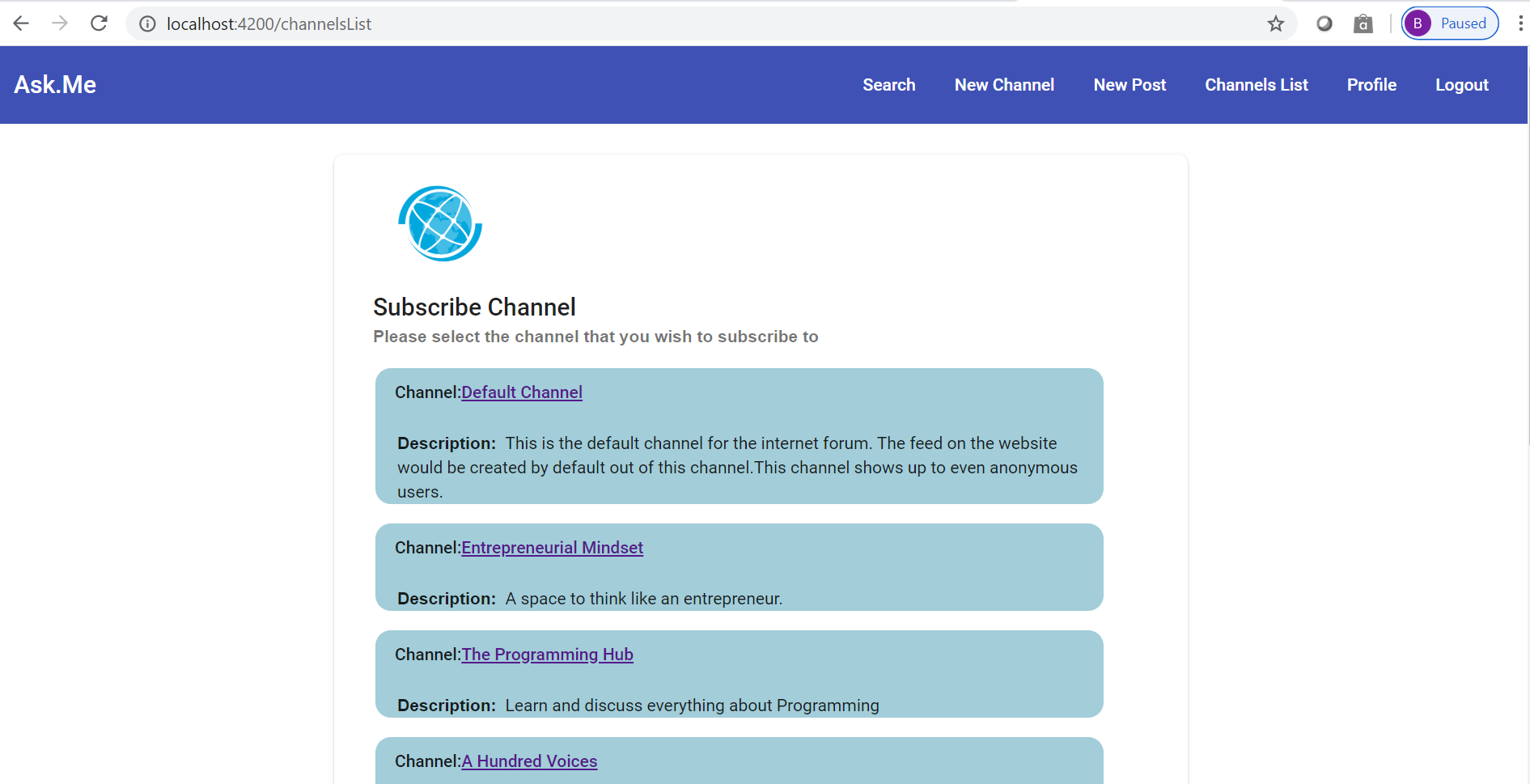
* + User creates a new channel



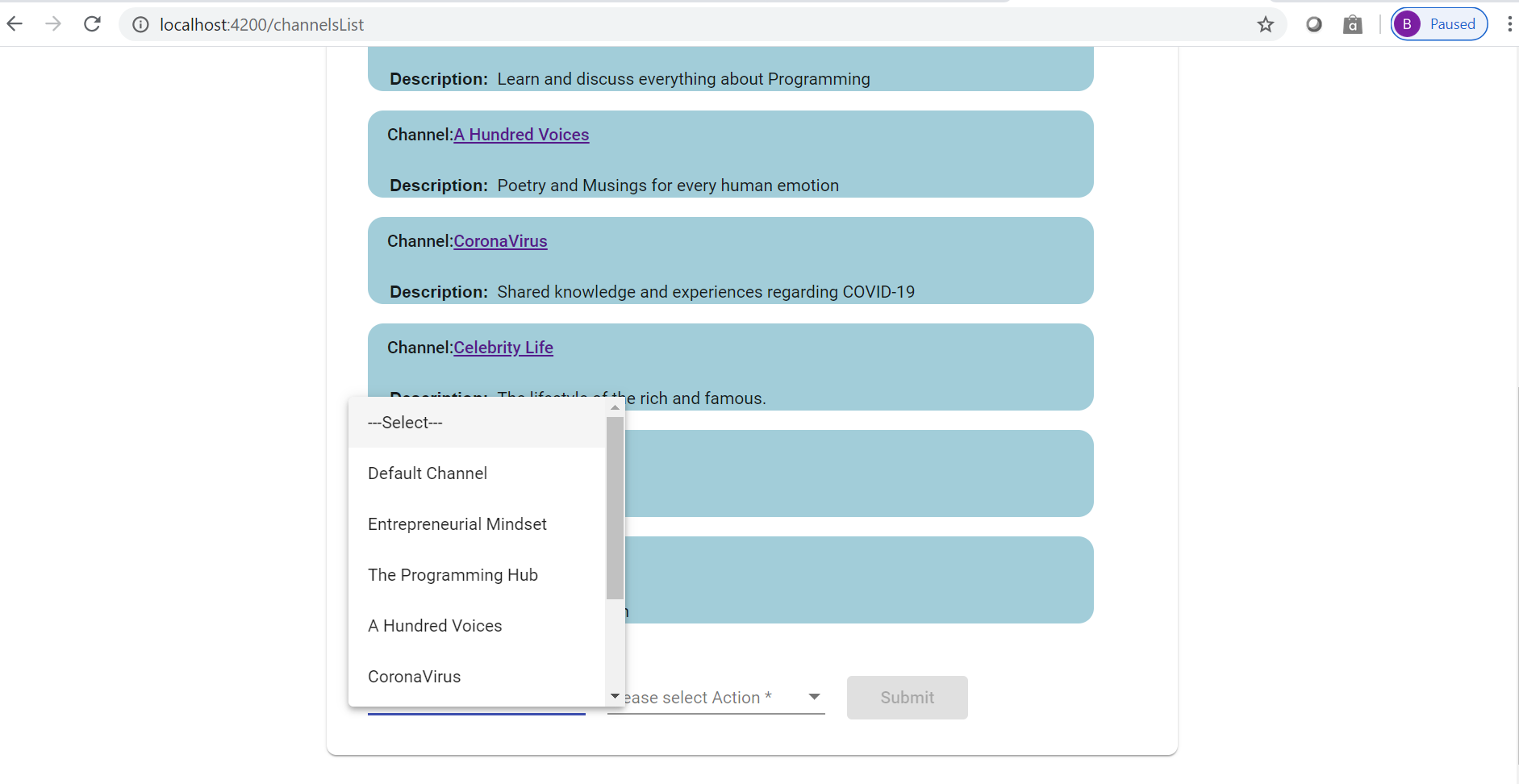
* + User searches for a post or a hashtag

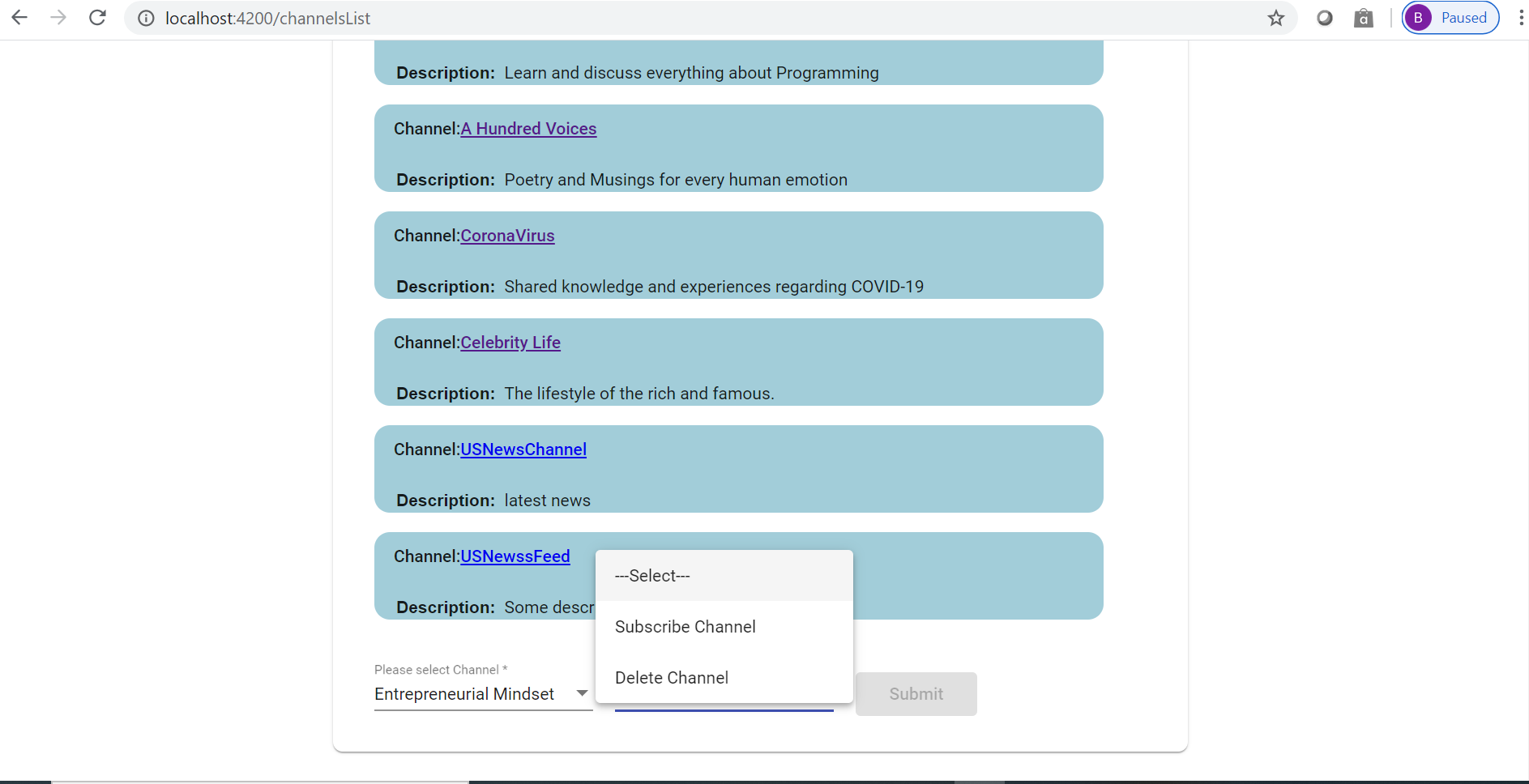


* + User views the list of channels and navigates to view further details.



* + On the channels list page, the user can select a channel and subscribe/delete (if he is the owner) a channel.





* + The navigation bar has buttons to take users to Home page, Search, New Channel, New Post, Channel List, Profile & Logout.



# 

# 

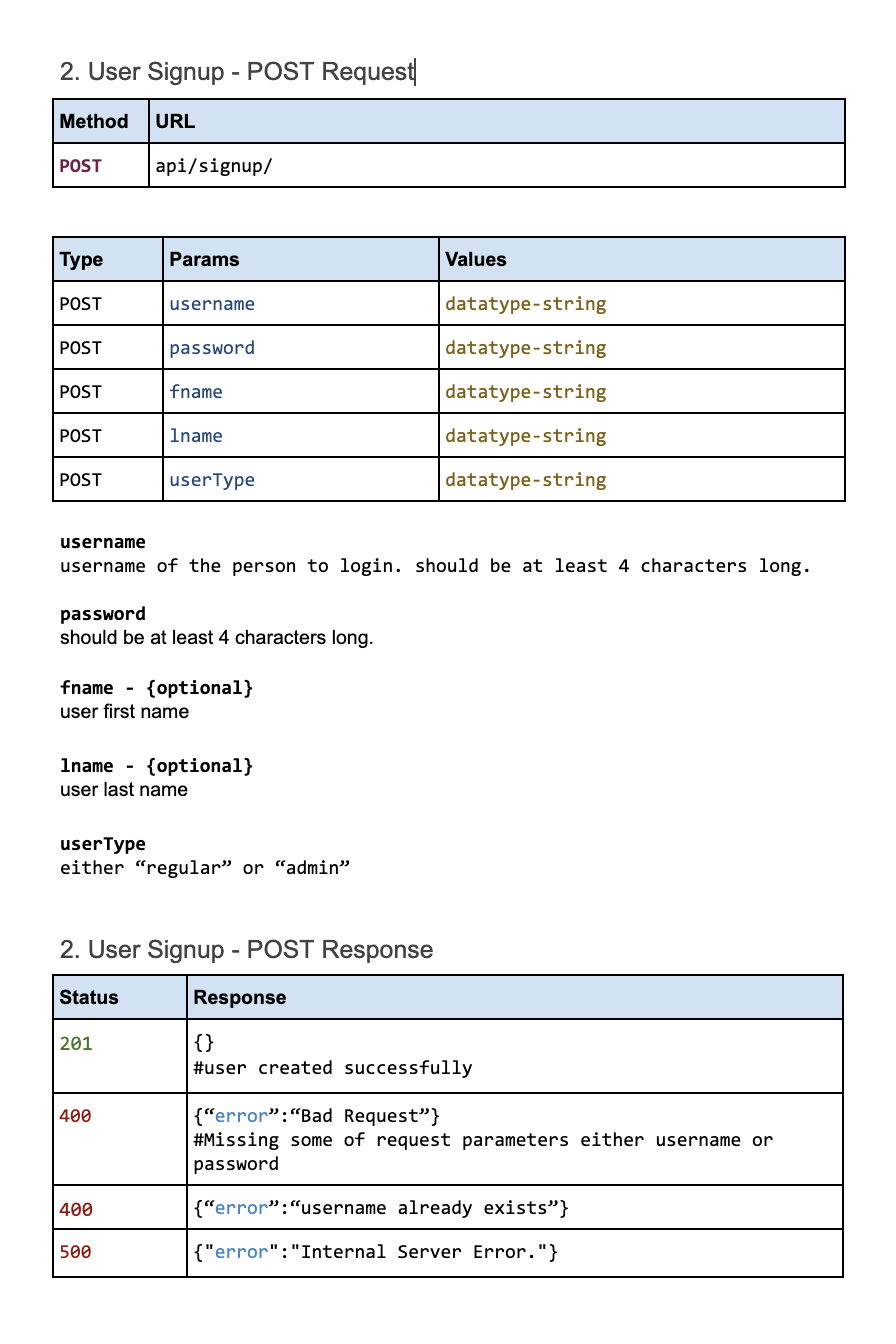
# 7. Self Analysis

Here we will discuss what went well throughout the project development.

## Better Collaboration - Comprehensive API documentation

Due to the current pandemic situation, collaboration in the team was a critical aspect. To ameliorate collaboration issues, we created comprehensive api documentation.

Here is the api documentation for User Signup API.



As you can see, the documentation provides

* API endpoint url with methods such as GET, POST, DELETE, UPDATE
* API parameters with the description whether the parameter is mandatory or optional
* Expected response from the backend, along with the error messages and statuses

We have documented a total 22 APIs with the same format as discussed. API Documentation is provided in the github repository under the documentation folder.

## Clean and Elegant UI

We believe we have a clean and elegant UI, praised by the TAs as well. We are able to achieve it using Angular Framework and Angular Material library. Component based development and the UI toolkits offered by Angular Material very essential factors to satisfy the Usability - one of our non functional requirements.

## In-Depth Understanding of Design Patterns

Due to this project, we can say that, we have in-depth understanding of the design patterns, we are very confident in the implementation of the design patterns. We had hour long team discussions on the design pattern usage part, where to use a specific design pattern, or coming up with a scenario where a particular design pattern is applied. These aspects strengthened our understanding of design patterns. Also, weekly discussions we had in the class were very helpful.

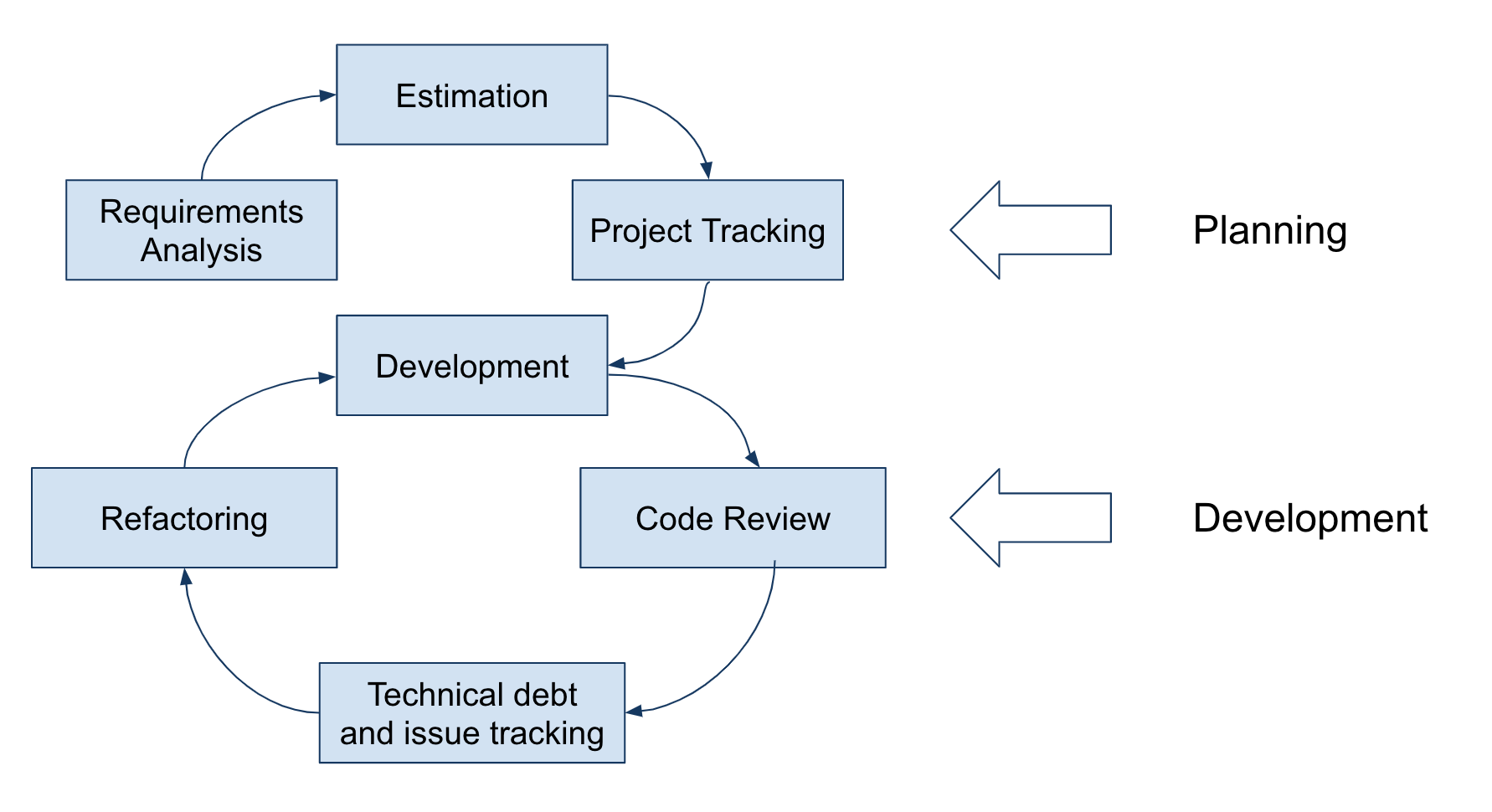
## Ebean ORM made our life easy

Development using Ebeans ORM was very easy, as we don't have to worry about writing complex sql queries to do CRUD operations on the database.

# 

# 8. Iteration workflow and contribution

## Iteration Workflow for each Sprint



Team Iteration workflow:

1. Requirements analysis

Before each sprint, we do the requirements analysis, we ensure that everyone in the team is clear on the requirements and understands them fully. Since the project was open ended we had to come up with our own requirements, also one of the main driving forces on the requirements was the design pattern usage.

1. Estimation  
   Once we are clear on the requirements we move to the estimation. We divide the requirements into tasks and subtasks. For each task we mark the criticality. We also point out the dependencies and prioritize the dependent tasks first.   
   Eg. API documentation: frontend and backend both have dependencies on this task. So prioritize it and complete it as early as possible, so that team members can work independently on their assigned tasks.
2. Project Tracking

We maintain all tasks and the respective assignee so that the progress is visible to everyone. We also have stand up calls to ensure there are no blockers.

1. Development, Code Review

We are following the PR request code review practice. We have in total 26 PRs in the repository.

1. Issue and technical debt tracking

We list down the issues and technical shortcuts taken. This practice is not strictly enforced compared to other practices.

1. Refactoring  
   We refactor the technical shortcuts taken before the delivery of the next sprint. This practice is not strictly enforced compared to other practices.

We give more emphasis on project planning so that the development process is clean and smooth.

## Contribution

* + Leeza
    - Front end development and application integration.
    - Prepared the demo and tutorials.
    - Implemented 2 design patterns
  + Gaurav
    - Project Planning

Requirements gathering, creating and assigning tasks, project tracking, project progress visibility.

* + - API documentation
    - Frontend setup. Development in collaboration with Leeza.
    - Implemented 7 Design Patterns
  + Gongpu
    - Led the backend subproject and implemented the majority of it.
    - Contributed 7 out of 20 design patterns.
    - Wrote part of the System Design section and all of the System Implementation section.
  + Charu
    - Worked on the backend project structure and designing part in collaboration with Gongpu.
    - Designed the initial database schema
    - Implemented 4 design patterns and was responsible for development view diagram in 4+1 views
    - Wrote Introduction, motivation and Related work section in the technical report.

# 

# 9. Conclusions and recommended future work

## Conclusion

We are successfully able to complete the **functional and nonfunctional requirements** for Sprint 1 and Sprint 2. All this is because of the solid teamwork and collaboration within the team. Team Brainstorming sessions helped a lot for **understanding the design patterns**, their usages, and finding the functionality that could use the specific design pattern. During Team Brainstorming sessions, we discussed it is important to know how to use design patterns, but more importantly when and where to use them. Sometimes the cost of applying the design pattern could outperform the value generated by using the design pattern. For example when we know we will have only one class for a use case for the foreseeable future, that is - we can't expect to have more similar classes. Then instead of using Factory or Abstract Factory, we can instantiate the class directly. Such pattern usages are known as “anti-pattern”.

## Future Work

1. **Implementation of 4 more design patterns**

We have to implement the following design patterns - Bridge, Abstract Factory, Composite and Chain of Responsibility. We would have to come up with the scenarios for the said design patterns so that we implement them.

1. **Microservice Architecture**

As discussed in the development view, we want to have our components deployed separately in isolation using Docker. This Microservice Architecture setup would conform to Availability, one of the non functional requirements.

1. **Testing**  
   In future we are planning to create test cases that cover at least 80% of the code.
2. **Responsive Web Page.**

People are browsing more on phones compared to computers now. It is critical for us to implement it to increase the user traction.