Codeversation:

Design Document

CS 307 - Software Engineering

Alex Aralis, Patricia Chun, Jeremy Lehman, Zach Perry

1 TABLE OF CONTENTS

2 Purpose

There are many obstacles to code debugging. Codeversation is a website that facilitates live collaboration on code snippets with instant feedback in the form of evaluation. The history of a codeversation will be conveniently exposed using Git diffs and branching to show who changed what and in what order.

Many sites offer complex, cumbersome ways of tracking projects with Git or other SVC software. These services are large because the scope of problem they try to solve is large. Codeversation will be different because it does not aim to track huge software repos, with multiple files and directories. Instead Codeversation is targeted at short code “snippets”. Because of the reduced scope a codeversation can be lightweight and fast, simple and easy for anyone to intuitively understand. Codeversation is for anyone who wants to collaboratively debug, improve, or learn about code.

2.1 Functional Requirements

2.1.1 Profile Features

1. As a user, I would like to have a specific username when I post my question.
2. As a user, I would like to delete my post.
3. As a user, I would like to log in to save my information and post history (if time allows).
4. As a user, I would like to have onboarding for the app.
5. As a user, I would like to save other user’s posts and comments.
6. As a user, I would like to block other users.
7. As a user, I would like to change the theme.

2.1.2 Codeversation Features

8. As a user, I would like to post programming-related questions.
9. As a user, I would like to see things that other users have posted.
10. As a user, I would like to create snippets based on other snippets.

11. As a Codeversation creator, I would like to select any comment or post as a solution.
12. As a user, I would like to have private Codeveration.
13. As a user, I would like there to be specific moderators.

2.1.3 Snippet Block Features

14. As a user, I would like to see nested comments on a snippet (if time allows).
15. As a user, I would like to see and make comments about the snippet.
16. As a user, I would like to flag comments and posts for inappropriate content.
17. As a user, I would like to be able to see the output of my code updated live.
18. As a user, I would like to tag my post.
19. As a user, I would like to see other posts related to mine.

2.1.4 Compiler Microservice Features

20. As a user, I would like to use any programming language I prefer.
21. As a user, I would like compilation to have low response times.

2.1.5 Front Page Features

22. As a user, I would like a dashboard/front page of popular posts (if time allows).

2.1.6 Server Features

23. As a user, I would like to share my question via a web link.
24. As a user, I would like to get email updates. (if time allows)

2.1.7 Anonymous Features

25. As a user, I would like to anonymously browse posts and comments.
26. As a user, I would like to be able to register a Codeversation account.

2.1.8 External Integration Features

27. As a user, I would like to be able to reference threads on Stack Overflow. (if time allows)
28. As a user, I would like to be able to import code from github. (if time allows)
29. As a user, I would like to create an account through github.
30. As a user, I would like to create an account through google.

2.2 Non-Functional Requirements

2.2.1 Response Time

1. Must be run on an Isomorphic server (server side rendering).
2. Page must be loaded in 60 seconds, or the page will timeout.

2.2.2 Scalability

3. Must be deployed on production server.
4. Must be able to service at least 100 simultaneous users.

2.2.3 Usability

5. Must be able to be used on Android. (if time allows)
6. Must be able to be used on iOS. (if time allows)
7. Must be able to be used in a web browser.
8. Must be mobile compatible.

2.2.4 Security

9. User accounts must have a hashed password.
10. Users can view posts and comments, but can not post or comment themselves until they have logged in.
11. New users should have a semi random Captcha to prevent spamming.
12. Users should have a security question and answer to recover their password.

3 Design Outline

3.1 Design Decisions

The overarching architecture of Codeversation is client-server. The client will be run in a user's browser and the server is NodeJS running in a cloud.

The client itself has an MVC architecture.

The Model is handled by a service oriented architecture. There is a service for snippet compilation (Custom microservice), user management and Codeveration storage (mlab/MongoDB).

Views are rendered with React, and the Controller is implemented with Redux.

The Server is implemented with NodeJS and the Express framework. It uses the Transaction-Processing architecture. Depending on the HTTP Headers and the URL, requests are handled differently.

3.2 Component Interactions

Client:

- Makes AJAX requests to server to login users, receive comments, and posts along with all other app data
- Receives html pages from server to be rendered
- Sends data to server about users and posts to be stored in database

Server:

- Handles data requests from client
- Sends html pages to be rendered
- Queries database for information
- Uses compilation microservice for the in browser compilation

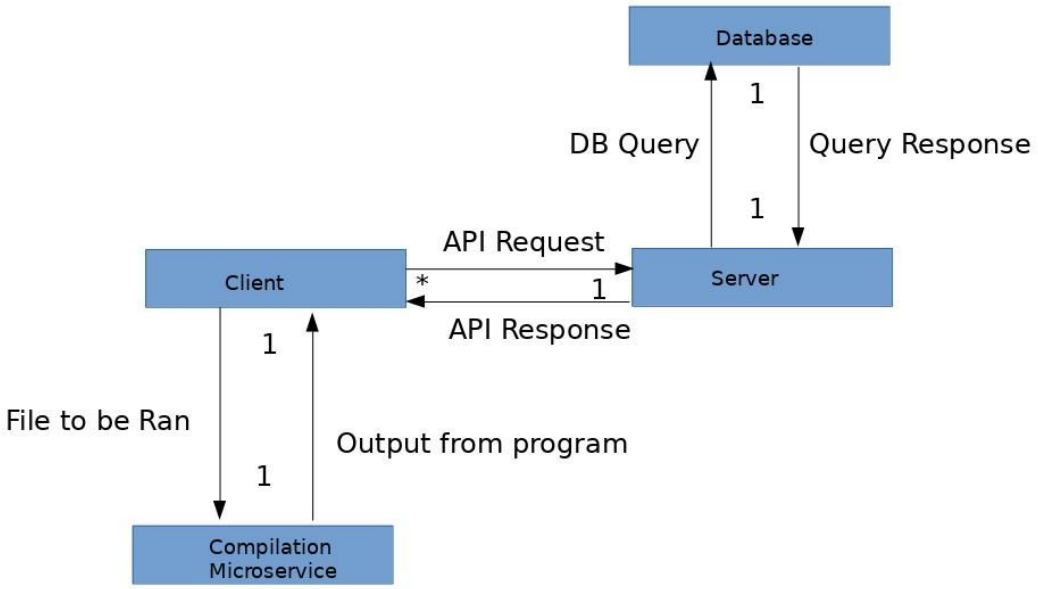
Database:

- Stores data about posts, users

Compilation Microservice:

- Takes file from server that was sent by client and compiles and runs the file, then returns the output

3.3 UML Diagram



4 Design Issues

4.1 Functional Issues

4.1.1 Snippet Evaluation Compilation Environment

Options

1. Client-Side
2. **Server-Side**

Decision

By compiling on the server-side the compiler does not need to be written in JavaScript. A server running native code will be able to run that code more quickly and efficiently. By encapsulating the component in a REST API it can be reused any place in the Codeversation client, server, or even 3rd party applications.

4.1.2 How should users appear

Options

1. Anonymously
2. **User created accounts**

Decision

We decided to have users create accounts so that we can save their content. This way, they can save codeversations, as well as create connections with other users. Also, they can save preferences such as languages color themes, syntax highlight, or keybindings.

4.1.3 How should users appear

Options

1. Anonymously
2. **User created accounts**

Decision

We decided to have users create accounts so that we can save their content. This way, they can save codeversations, as well as create connections with other users. Also, they can save preferences such as languages color themes, syntax highlight, or keybindings.

4.2 Non-Functional Issues

4.2.1 Front End Framework

Options

1. HTML/CSS, Vanilla.js
2. **ReactJS + Redux**
3. AngularJS

Decision

There are a multitude of frameworks that can be used to create web applications, but we decided to use React for the view component and Redux for the controller because of familiarity. Both are lightweight libraries that will allow us to extend it for our own uses easily. Additionally, we will be producing and displaying dynamic content, which both libraries are very good at doing.

4.2.2 REST API and database

Options

1. Split REST API and database into separate servers
2. **Combine REST API and database in one server**

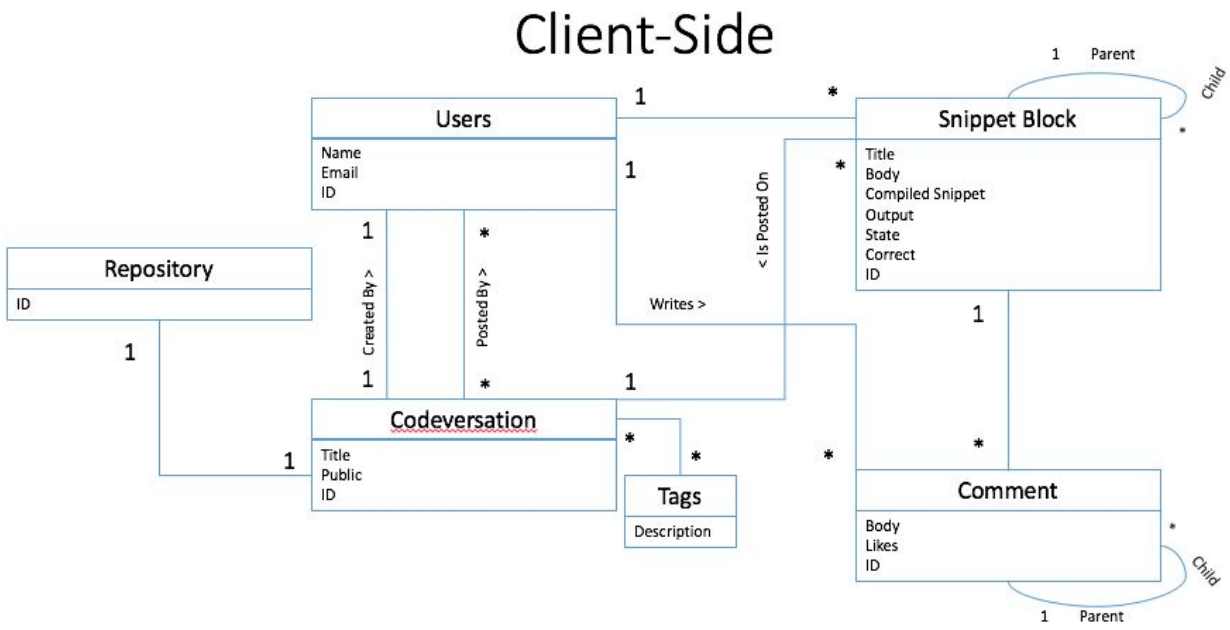
Decision

We decided to combine the REST API and database on one server, because we believe it will be easier to just maintain one server. Also, it will be simple to separate the REST API into its own server if needed.

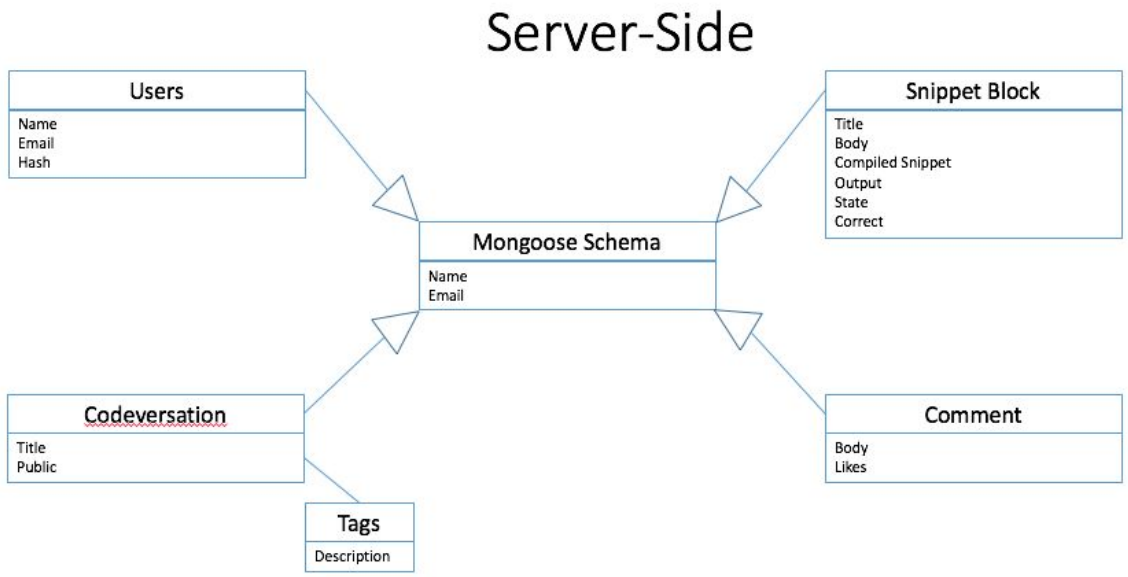
5 Design Details

5.1 Class Diagram

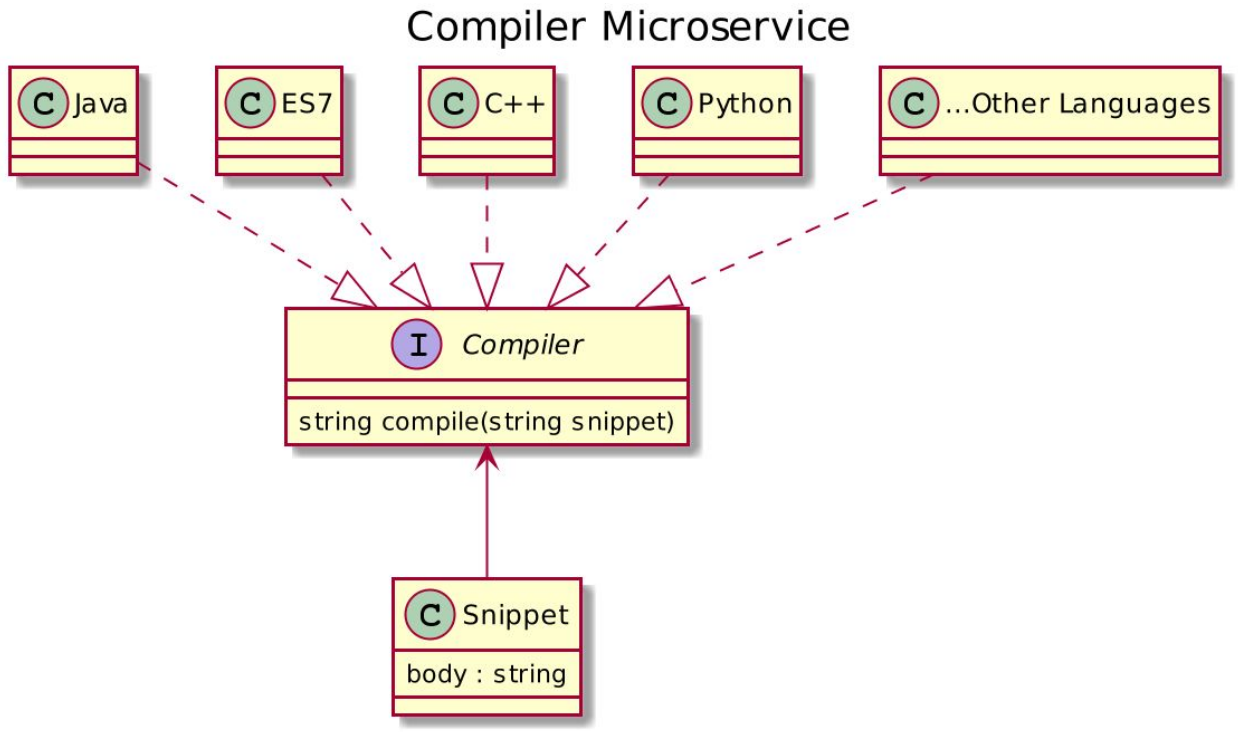
5.1.1 Client-Side



5.1.2 Server-Side



5.1.3 Compiler Microservice



5.2 Class Descriptions

5.2.1 User

This class represents a logged in user's information. On the client side it will provide ways of accessing a user token, user name, user email, and any other user information required by the GUI. The user token will be provided to the client on login and will be used as authentication of a user session.

On the server-side this class will be defined by a Mongoose Schema. The user ID stored in the client side token can be used to select persistent user information from the data backing MongoDB. This schema also has fields for name, email, Snippet Blocks, Codeverations created, and Comments posted of the user.

5.2.2 Snippet Block

This is a class that represents a block/snippet of code posted by a user along with the snippet title, comments, language, likes, whether the snippet is the accepted solution, and git commit hash.

In the client this class will also store the compiled snippet (if the code is being evaluated), as well as a Mongoose ID for making API requests. Also any GUI specific snippet information will be stored here.

On the server the Snippet Block class will be a Mongoose Schema that has fields for all the persistent data of the snippet.

5.2.3 Codeversation

The Codeversation class stores general information about Codeversation threads. This information include title, Snippet Blocks, creator, Tags, Resolved State, Main Snippet Block, and Git Repo.

On the client there will be GUI state Codeversation information and an Codeversation Mongoose ID.

On the server Codeversation will be a Mongoose schema that has fields for all the persistent data of the Codeversation.

5.2.4 Comment

A comment is has a body, a parent Comment, reply Comments, likes, and creator.

On the client a Comment also has some GUI state information as well as an Mongoose ID.

On the server a Comment will be a Mongoose schema that has fields for all the persistent data of the Comment.

5.2.5 Tag

Tags are small classes that represent a descriptive tag that can be associated with a codeversation.

On the client a Tag will have a description, as well as a Mongoose ID.

On the server Tag will be a Mongoose schema that has fields for all the persistent data of the Tag.

5.2.6 Repo

The Repo class encapsulates the Git repository that is used to store the tree structure of a Codeversation. Every Repo will have an associated Codeversation.

5.2.7 Compiler << Interface >>

Compiler is an Interface that specifies the compile method that can be used to compile simple code snippets and produce JavaScript output.

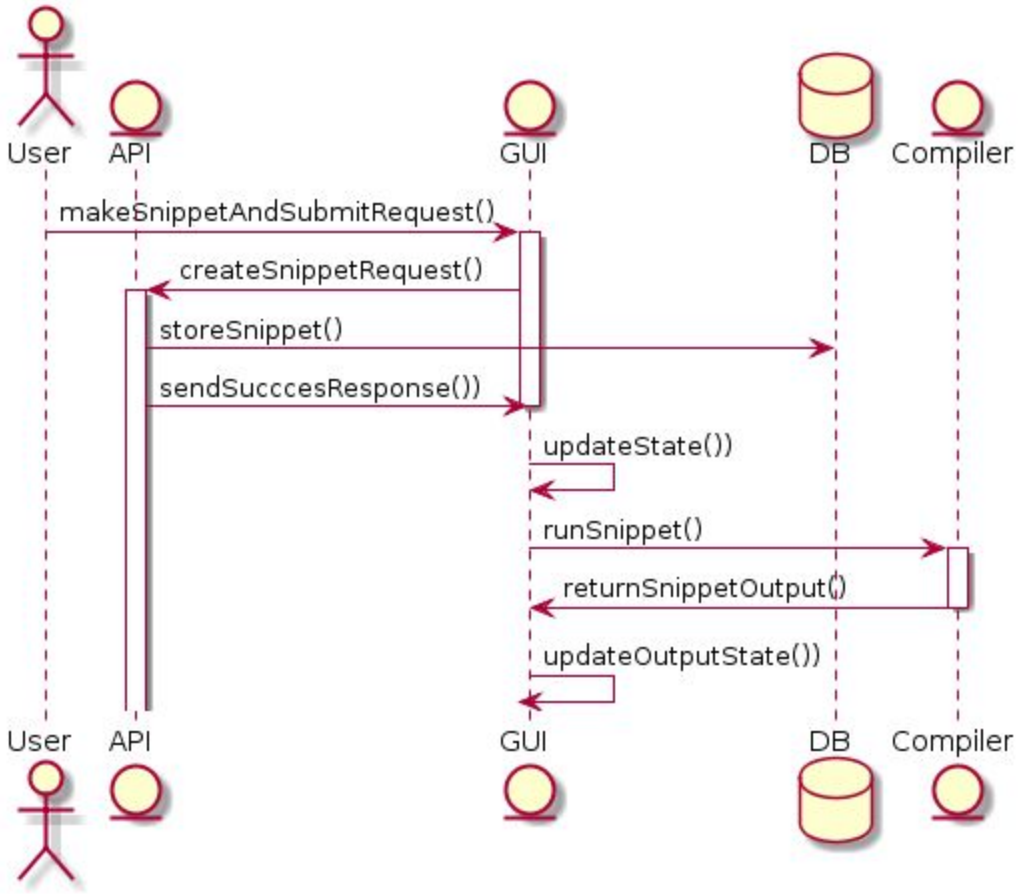
This Interface will be implemented by specific language classes on the compiler Microservice. For instance there will be a Java class that implements Compiler. Java can then be used to compile Java code into JavaScript via Java.compile().

5.2.8 Snippet

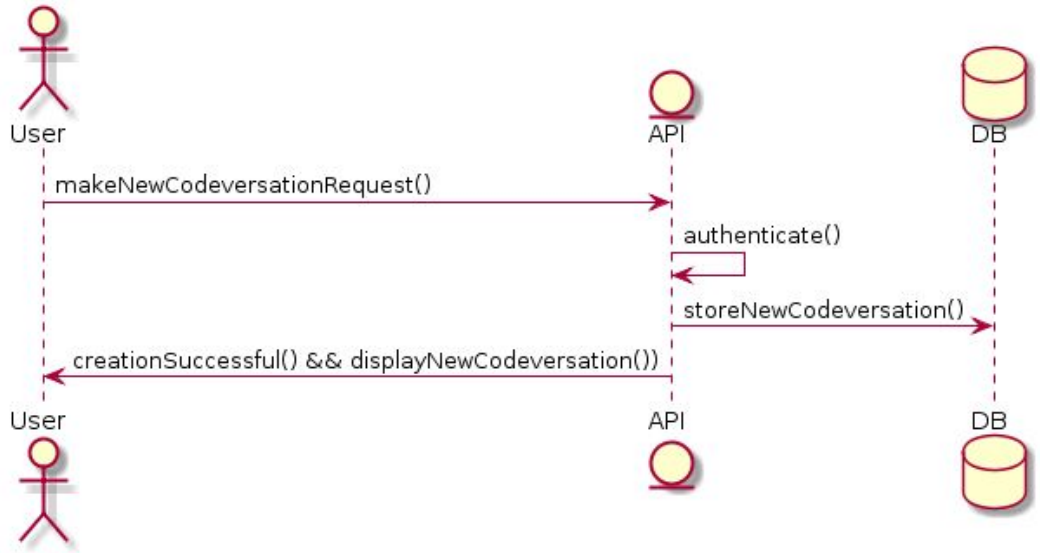
The snippet is a simple class that with a language and a code body attributes. A snippet is sent to the Compiler Microservice to be compiled.

5.3 Sequence Diagrams

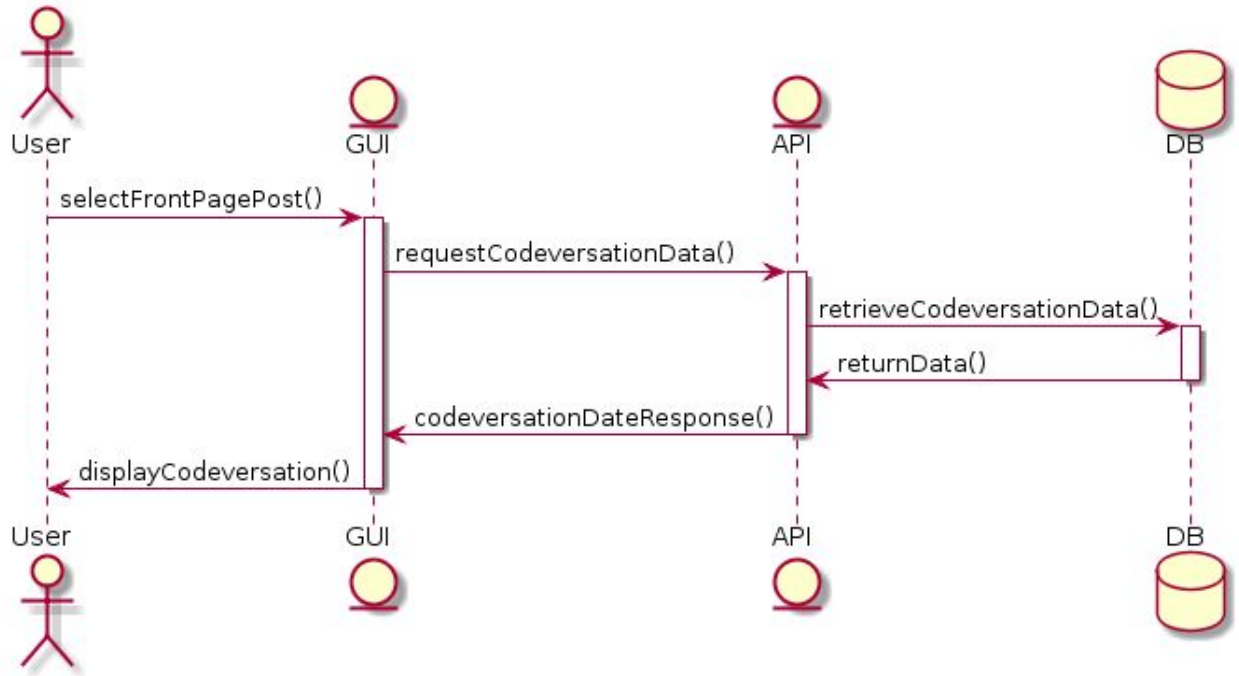
5.3.1 Adding a new Snippet



5.3.2 Creating a new Codeversation

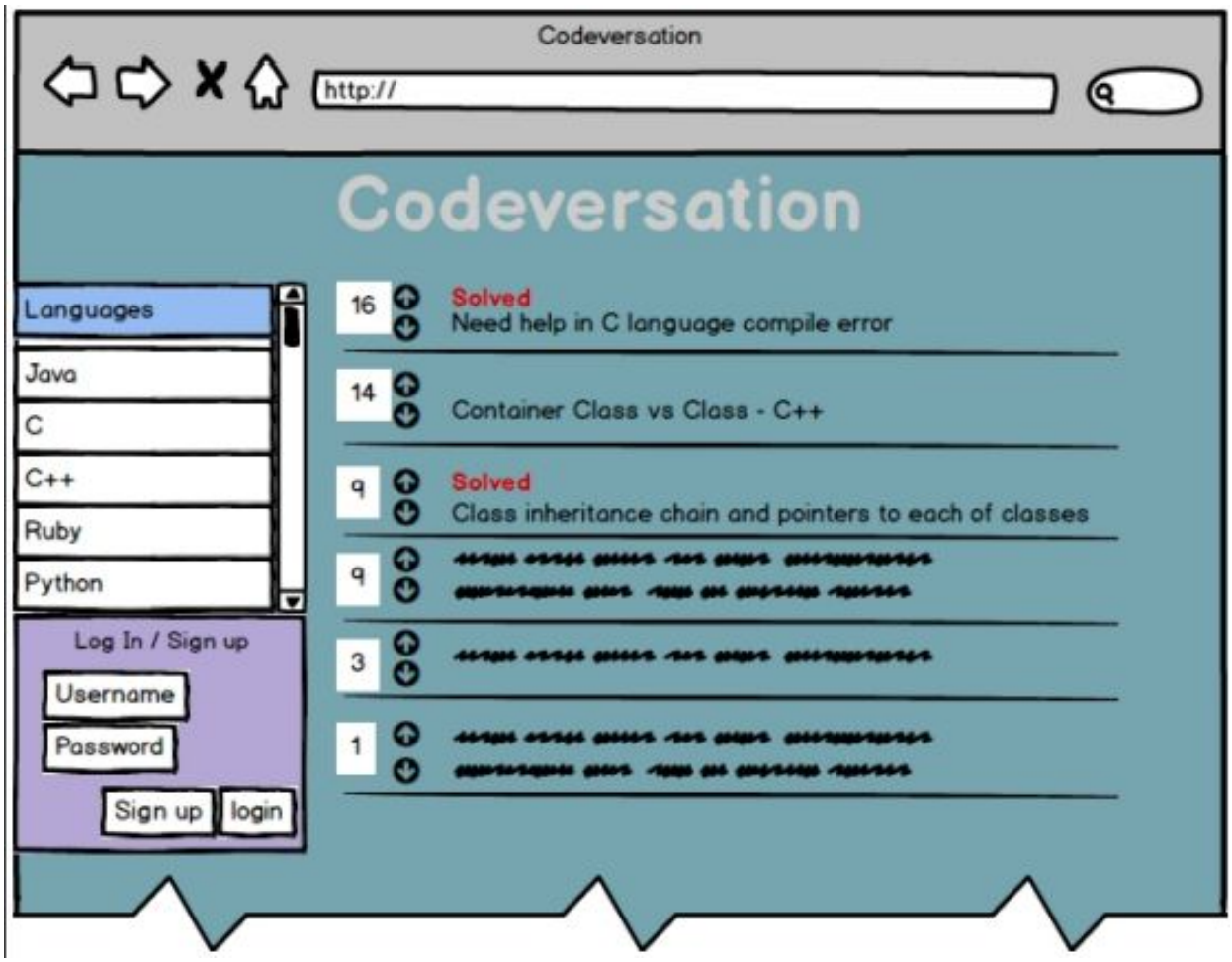


5.3.3 Open a codeversation



5.4 UI Mockups

5.4.1 Front Page



5.4.2 Codeversation View