# Dexter

Siddharth Papreja 19201215

Sagar Jape 19200117

William Ikenna-Nwosu 16303711

**Synopsis:**

**Describe the system you intend to create:**

***What is the application domain?***

A web application where users are shown code snippets and they can find out the possible errors in the code thereby providing solutions to correct or improve the efficiency of the code.

***What will the application do?***

Allows multiple people to play the game simultaneously i.e. millions/billions improving a particular program asynchronously.

Multiple people can view the same code snippet simultaneously and provide their feedback. The users can view feedback given by other users as well, thus improving their understanding of possible coding techniques to improve the efficiency of a code.Hence, multiple perspectives on the same program from various users can be obtained.Thus the application allows users to better their understanding by receiving feedback from several users and also provide their own feedback on a particular code snippet.

***What are the key components and how will they interact?***

*The key components include-*

1. **CodeReviewerOrchestratorService** - A microservice that receives requests from the user, sends the request to the corresponding microservice(mentioned below) based on the request and sends the response back to the client.
2. **RevieweeService** - Microservice responsible for handling the user details. Contains services to create new users and authenticate users before granting access to the application. Uses MySQL to store and retrieve the data.
3. **RevieweeCodeService** - Microservice responsible for handling code snippets. Used to store code snippets submitted by the user, fetch the list of code snippets submitted, fetch a particular code snippet along with the feedback associated with the code snippet. It saves the code snippets in MongoDB and uses Redis caching for better performance.
4. **User Interface** - The users can interact with the application through the user interface provided by the web application. The users can upload code snippets, provide feedback to other user’s code snippets and read other users feedback as well.

## Technology Stack

**Spring-boot** - To build production-ready application in order to run the microservice architecture. Also used to increase the scalability and flexibility of the application due to SpringBoot's dependency injection capabilities which also allow us to compartmentalize injector classes to extend them later when necessary without interfering with the remainder of the codebase.

**REST API** - To allow microservices such as RevieweeService and RevieweeCodeService to communicate with the CodeReviewerOrchestratorService. The application is exposed to the client using REST API as well. The CodeReviewerOrchestratorService is the entry point for the REST APIs. Based on the intended service required, the CodeReviewerOrchestratorService fetches response from the corresponding microservice by requesting access to the resource using REST API request.

**Redis** - To cache the code using a codeId in the RevieweeCodeService microservice. Thus, every time a user fetches the same piece of code, instead of fetching it from the database, the code is fetched from the cache. This ensures faster processing of the request, especially in a distributed system to ensure scalability and improved performance.

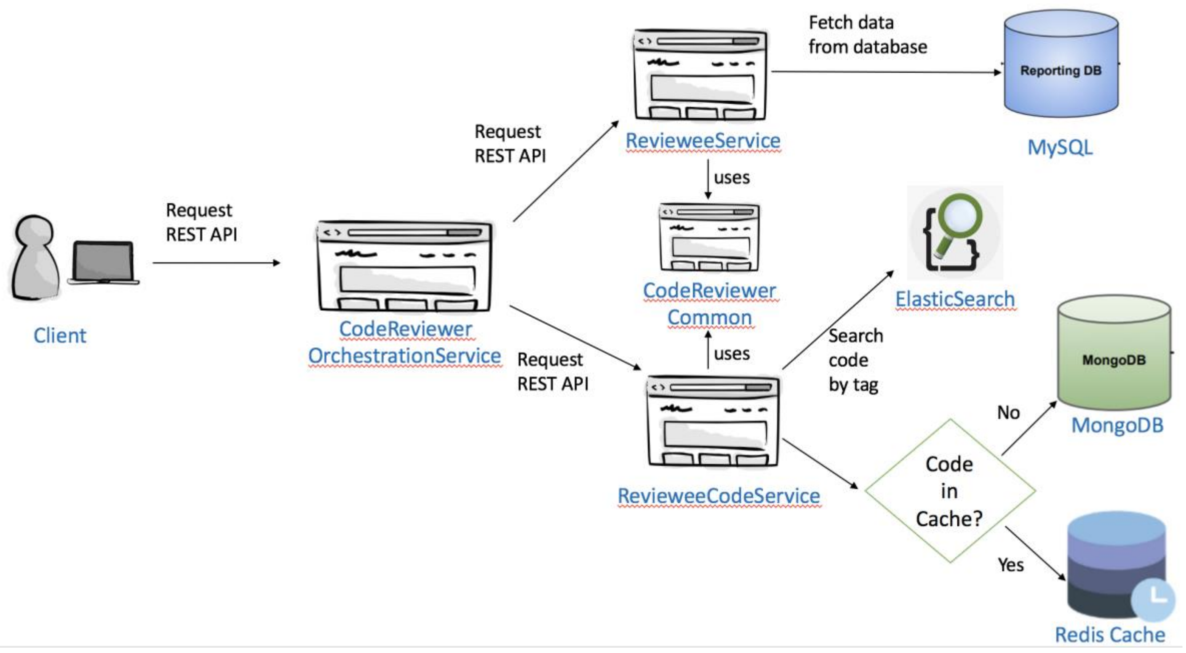
**MongoDB** - To store the code snippets every time a user uploads a code for review. The code snippet along with the author and tag is stored in the database. When other users provide their feedback on the code snippet, the document is updated and the feedback are added.

**MySQL** - MySQL is used to store the user details. The reviewer and reviewee details are handled by the RevieweeService microservice and the microservice uses MySQL database to store and authenticate the user.

**ElasticSearch** - In order to improve the efficiency and decrease the latency involved with searching for code snippets with tags that match a particular tag, ElasticSearch is used. The RevieweeCodeService uses ElasticSearch to find code snippets based on the tag and returns a list of code snippets.

**System Overview**

Include your system architecture diagram in this section. Explain how your system works based on the diagram.



The client interacts with the application using REST API call to the CodeRevieweeOrchestrationService microservice which is the coordinator microservice and manages all the requests coming in. The CodeRevieweeOrchestrationService microservices, based on the requested resource, makes a REST API call to the corresponding microservice to process the request.

Requests that involve User Creation or User Authentication are serviced by the RevieweeService while the requests involving Code Upload, Feedback on Code or Fetching Code Snippets are serviced by the RevieweeCodeService.

RevieweeService interacts with MySQL database to store and retrieve user data and also validates the user credentials during authentication.

RevieweeCodeService uses MongoDB to store and retrieve Code Snippets and list of feedback for the codes. It also uses Redis Cache in order to cache the code snippets using the CodeId as the key. If the code is not present in the cache, it fetches the code and corresponding list of feedback from the MongoDB database.

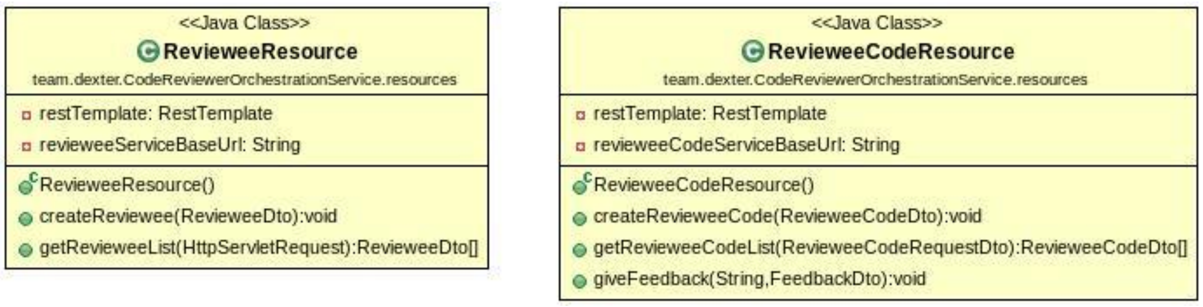
RevieweeCodeService also uses ElasticSearch in order to search for a particular code based on the tag inputted by the user. If the ElasticSearch server has the tag present, it returns the codes corresponding to that tag, thus ensuring better performance. If the tag is not present, it fetches it from the database.

Both RevieweeCodeService and RevieweeService use the common code present in CodeReviewerCommon project.

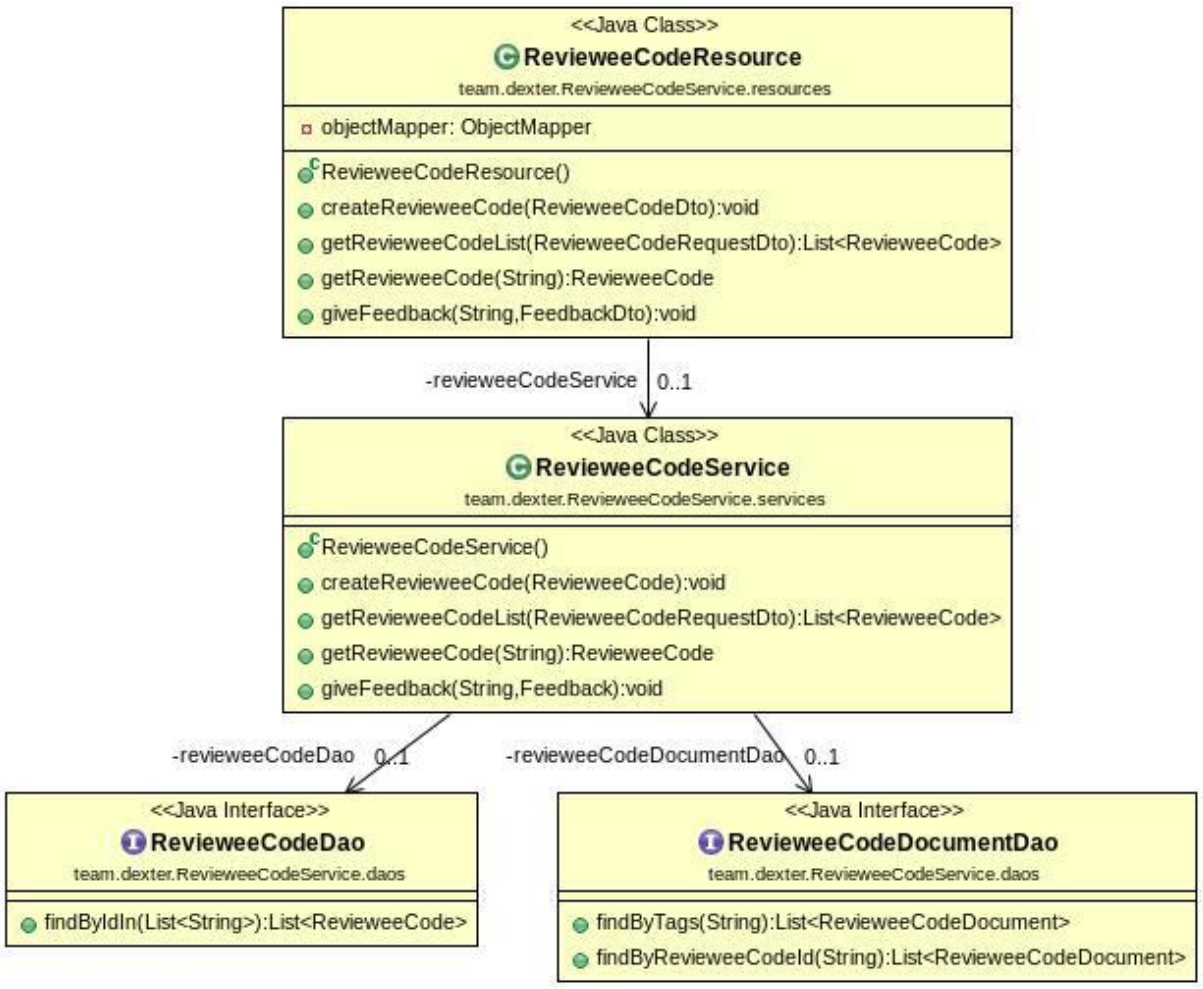
The response is returned to the client by CodeRevieweeOrchestrationService microservice.

**Class Diagrams:**

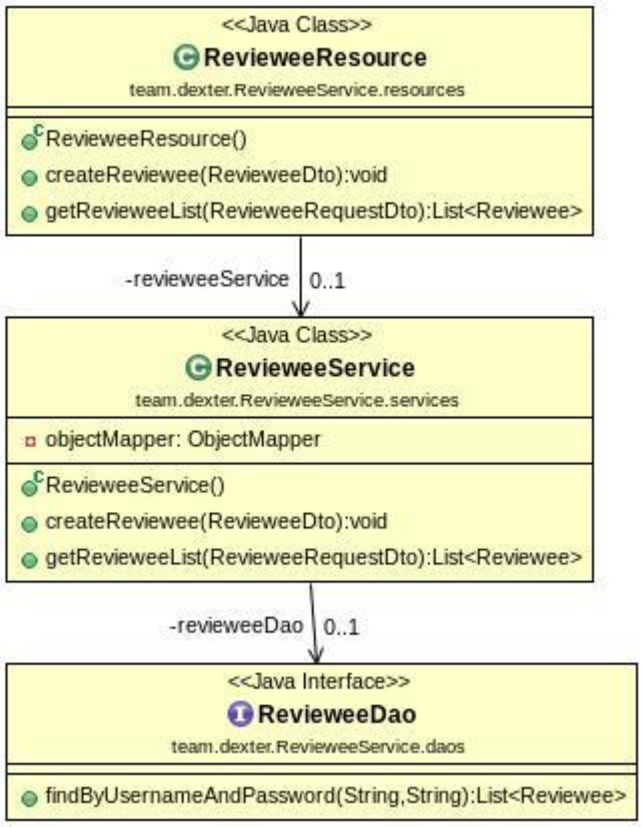
The following class diagram represents the classes of CodeRevieweeOrchestrationService:


<<Java Class>>
RevieweeResource
team.dexter.CodeReviewerOrchestrationService.resources
restTemplate: RestTemplate
revieweeServiceBaseUrl: String
RevieweeResource()
createReviewee(RevieweeDto):void
getRevieweeList(HttpServletRequest):RevieweeDto[]
<<Java Class>>
RevieweeCodeResource
team.dexter.CodeReviewerOrchestrationService.resources
restTemplate: RestTemplate
revieweeCodeServiceBaseUrl: String
RevieweeCodeResource()
createRevieweeCode(RevieweeCodeDto):void
getRevieweeCodeList(RevieweeCodeRequestDto):RevieweeCodeDto[]
giveFeedback(String,FeedbackDto):void

The following class diagram represents the classes of RevieweeCodeService:


<<Java Class>>
RevieweeCodeResource
team.dexter.RevieweeCodeService.resources
objectMapper: ObjectMapper
RevieweeCodeResource()
createRevieweeCode(RevieweeCodeDto):void
getRevieweeCodeList(RevieweeCodeRequestDto):List<RevieweeCode>
getRevieweeCode(String):RevieweeCode
giveFeedback(String,FeedbackDto):void
-revieweeCodeService
0..1
<<Java Class>>
RevieweeCodeService
team.dexter.RevieweeCodeService.services
RevieweeCodeService()
createRevieweeCode(RevieweeCode):void
getRevieweeCodeList(RevieweeCodeRequestDto):List<RevieweeCode>
getRevieweeCode(String):RevieweeCode
giveFeedback(String,Feedback):void
-revieweeCodeDao
0..1
-revieweeCodeDocumentDao
0..1
<<Java Interface>>
RevieweeCodeDao
team.dexter.RevieweeCodeService.daos
findByIdIn(List<String>):List<RevieweeCode>
<<Java Interface>>
RevieweeCodeDocumentDao
team.dexter.RevieweeCodeService.daos
findByTags(String):List<RevieweeCodeDocument>
findByRevieweeCodeId(String):List<RevieweeCodeDocument>

The following class diagram represents the classes of RevieweeService:



<<Java Class>>
RevieweeResource
team.dexter.RevieweeService.resources
RevieweeResource()
createReviewee(RevieweeDto):void
getRevieweeList(RevieweeRequestDto):List<Reviewee>
-revieweeService
0..1
<<Java Class>>
RevieweeService
team.dexter.RevieweeService.services
objectMapper: ObjectMapper
RevieweeService()
createReviewee(RevieweeDto):void
getRevieweeList(RevieweeRequestDto):List<Reviewee>
-revieweeDao
0..1
<<Java Interface>>
RevieweeDao
team.dexter.RevieweeService.daos
findByUsernameAndPassword(String,String):List<Reviewee>

## Reflections

### What were the key challenges you have faced in completing the project? How did you overcome them?

1. **Designing a scalable application:** The first challenge we faced was to design an application that could be scalable. We wanted to ensure that even if one service went down, the application did not fail. Thus, we thought of designing the application by breaking it into smaller services. Each main functionality could be treated as a service. We decided to use 2 microservices - RevieweeService to handle user related operations and RevieweeCodeService to handle code snippet and feedback related operations.
2. **Choosing a suitable database:** Next, we had to decide on the database to be used. For storing user credentials and user data, MySQL was suitable since the data was structured. However, for storing feedback given by several users and appending each feedback against the same code, we decided to use MongoDB - a NoSQL database to ensure scalability. Since the schema of the table can be later on extended by adding more columns, the use of MongoDB which doesn't require a rigid schema was beneficial. This gave us more flexibility in storage and MongoDB was a natural choice as we stored feedback in the form of array which MongoDB naturally supports given its document-oriented nature.
3. **Ensuring low latency for smooth experience:** The feedback for all code snippets would rarely be updated at the same time. Thus, fetching the code and the feedback from the database for every request- even if there has been no updates between the subsequent GET requests, would result in latency since database connections takes

a longer time. We decided to improve this by adding caching. We used Redis cache such that the first GET request against a codeId is retrieved from the database, cached in the REDIS cache so that subsequent requests can be retrieved from the cache itself ensuring better performance. The cache is cleared for the particular codeId only if a new feedback is added - in which case, the code snippet and feedback including the latest feedback is fetched from the database and updated in the cache.

4. **Packaging the application for smooth deployment and running:** We then had to find a way to package the application, libraries and dependencies so that we could easily deploy and run the application. Thus, we decided to dockerise the application so that the entire application including the microservices, caching, elastic search, plugins are easily shipped as a single package.

**What would you have done differently if you could start again?**

- **Additional functionality:** We would have added a few more functionalities such as upvoting a feedback, allotting points to users for each upvote on their feedback, maintain a leaderboard, provide difficulty levels for users to choose from.

**What have you learnt about the technologies you have used? Limitations? Benefits?**

The project provided us with an opportunity to explore technologies such as Redis Cache, Elastic Search, MongoDB, REST Api.

Each of these technologies has their own benefits in terms of how they scale, the performance improvement of the application, accessing each service independently and concurrently. Each technology has its limitations as well.

**Benefits:**

**Redis Cache:** Allows storing key-value pairs. Thus, the id of each object can be used as a key and the object itself can be stored in the cache as the value. Redis stores data in the form of a key and a map and uses its own hashing mechanism called Redis Hashing

**MongoDB:** Since the database is schema less, new fields can be added as and when required and the number of field, size and content of various documents can differ. Since it allows sharding, when the size of data grows, the data can be stored in different machines - thus allowing horizontal scaling.

**REST API:** The application can be easily scaled and is easy to build and adapt. REST API allows manipulating the resources and separates the client and the server completely allowing portability of the interface. Since it is independent of the platform, application can be tested easily in new environments.

**Elastic Search:** ElasticSearch provides a flexible query api and supports JSON based REST API. Elasticsearch handles unstructured data automatically, thus indexing JSON documents without predefining the schema is possible

## Limitations:

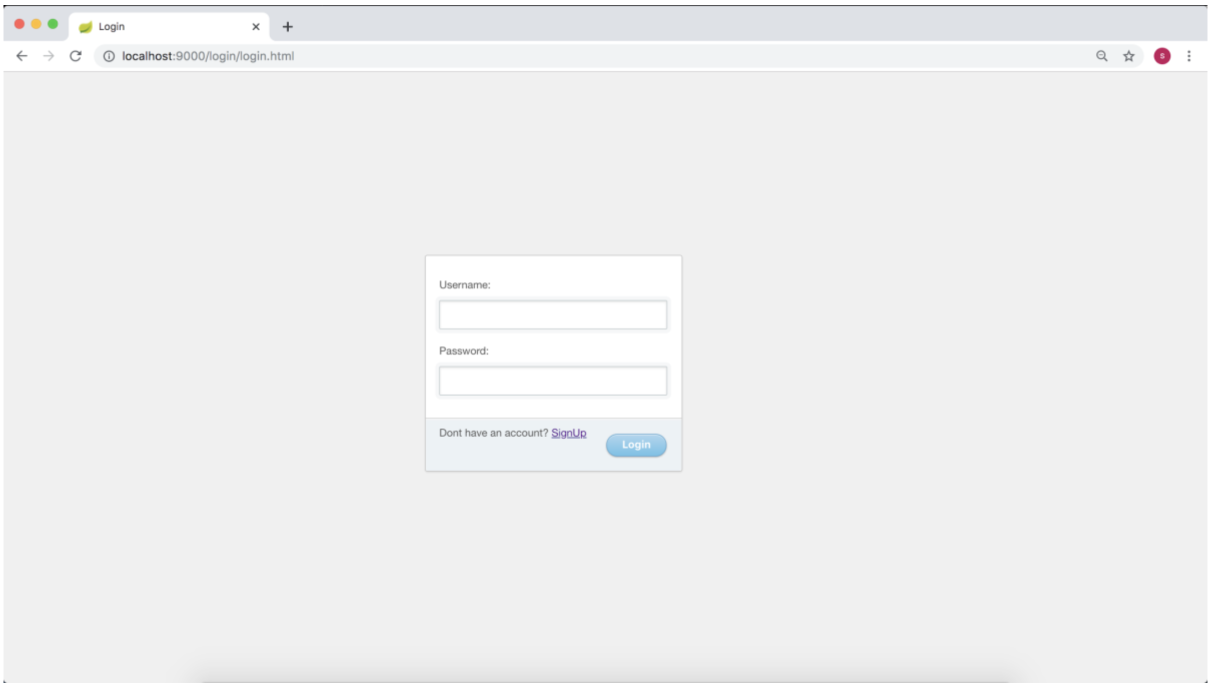
**Redis Cache:** Writing and deleting huge amounts of data in the cache results in performance degradation. The whole dataset always resides in the RAM and hence could be costly and must be used wisely.

**MongoDB:** MongoDB is quite expensive memory-wise and doesn't support joins which could make information retrieval clunky(implement join manually) and expensive(time and space complexity)

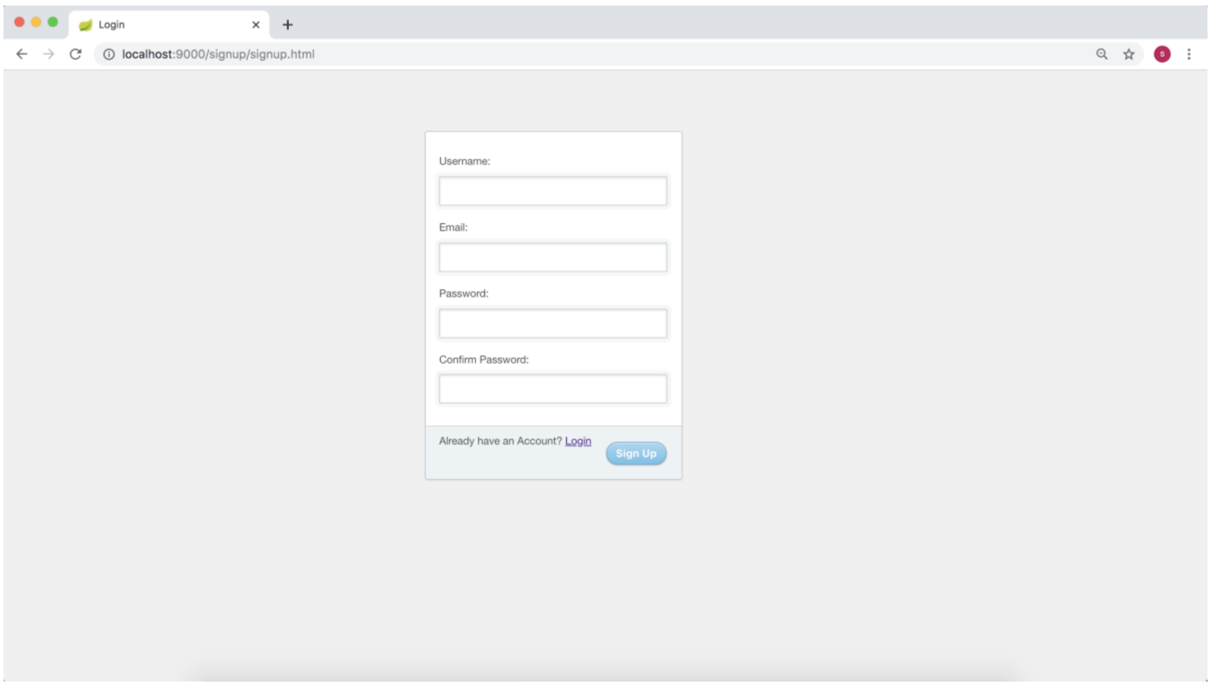
**Elastic Search:** As data grows in size to terabytes elasticsearch needs more time to refresh indexes which means every time a user submits his code it will take more time to reflect in elasticsearch. Also, a duplicate data store needs to be maintained - one in MongoDB and the other in elastic.

## Screenshots:

Login Screen - On entering the credentials and clicking the login button, a REST api is called with username and password as parameters. It validates the user credentials and based on the validation, grants user, access to the application.

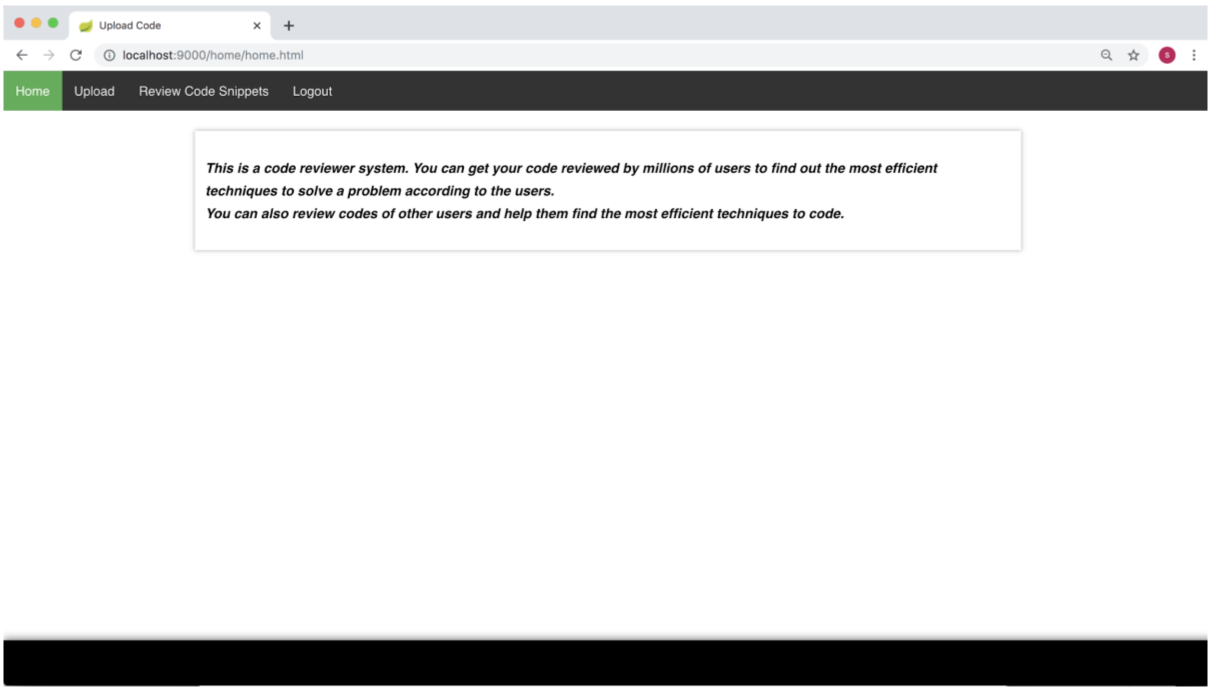


SignUp Screen- New users can signup into the application.


Login
localhost:9000/signup/signup.html
Username:
Email:
Password:
Confirm Password:
Already have an Account? Login
Sign Up

---

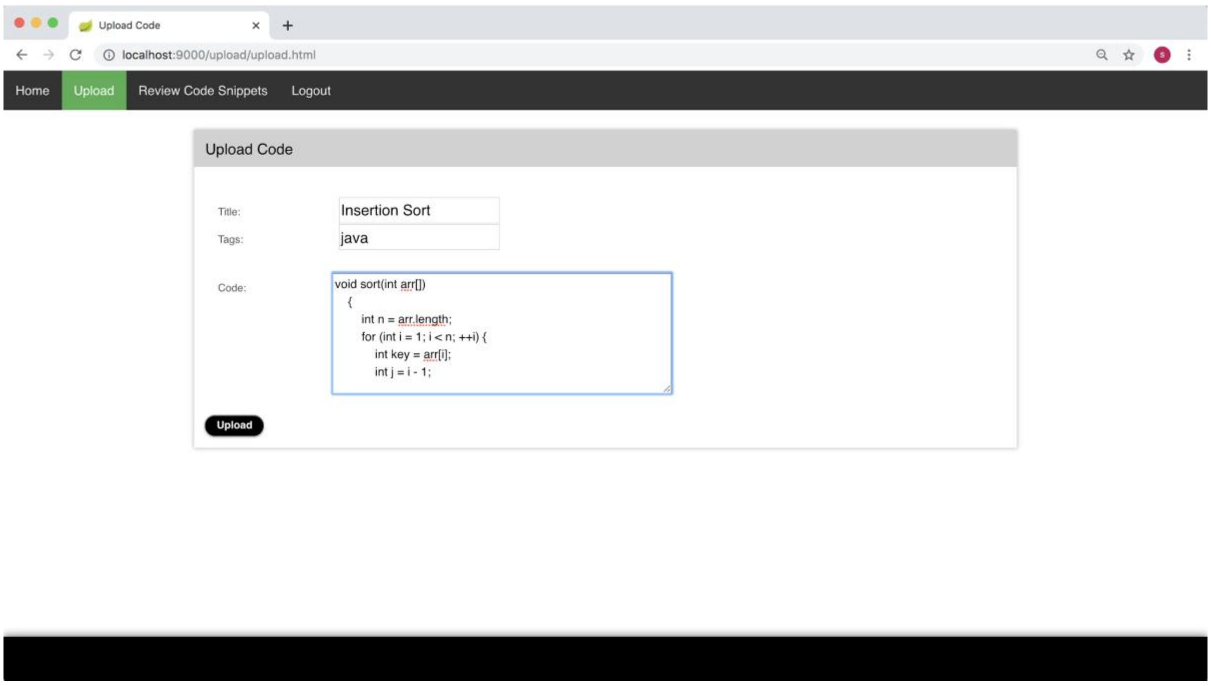
Homepage Screen - A brief description of the application and the functionalities provided by the application.



*This is a code reviewer system. You can get your code reviewed by millions of users to find out the most efficient techniques to solve a problem according to the users.*
*You can also review codes of other users and help them find the most efficient techniques to code.*

---

Upload Screen- Allows user to upload code snippets for other users to review and provide feedback.

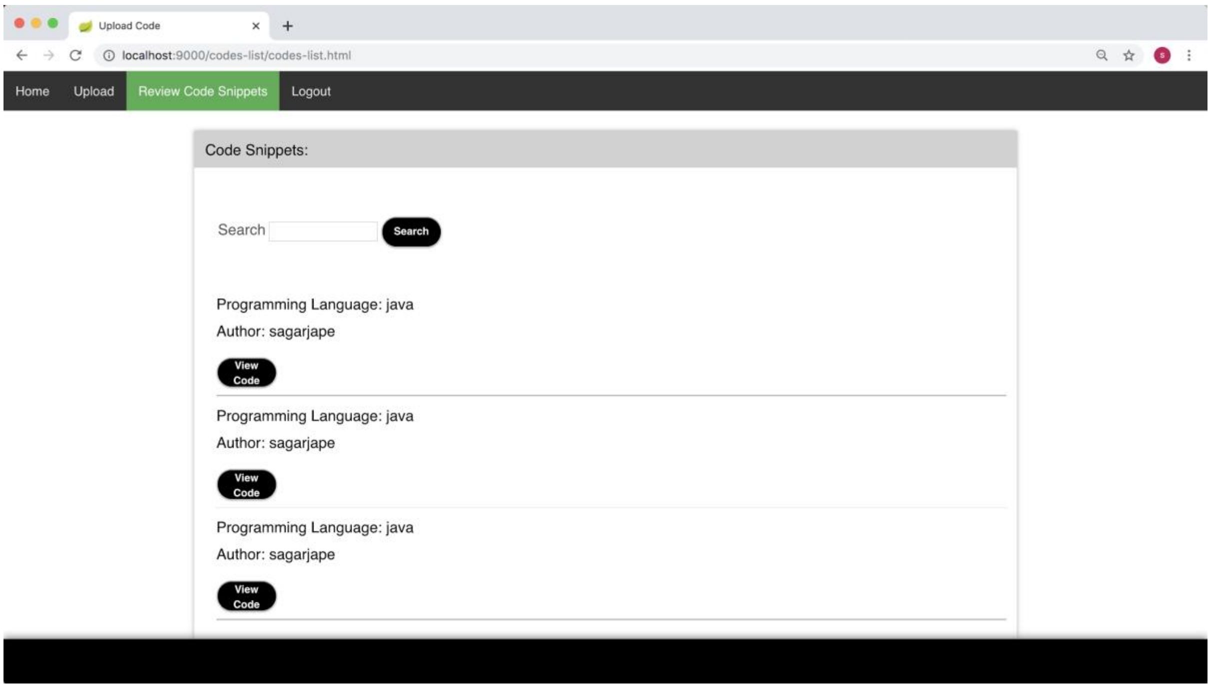


Title: Insertion Sort

Tags: java

Code:
```
void sort(int arr[])
  {
      int n = arr.length;
      for (int i = 1; i < n; ++i) {
         int key = arr[i];
         int j = i - 1;
```

Review Code Snippet Screen- Lists all the code snippets uploaded by various users. User can search for codes based on a particular tag associated with the code snippet while uploading. This search uses ElasticSearch to reduce the latency.

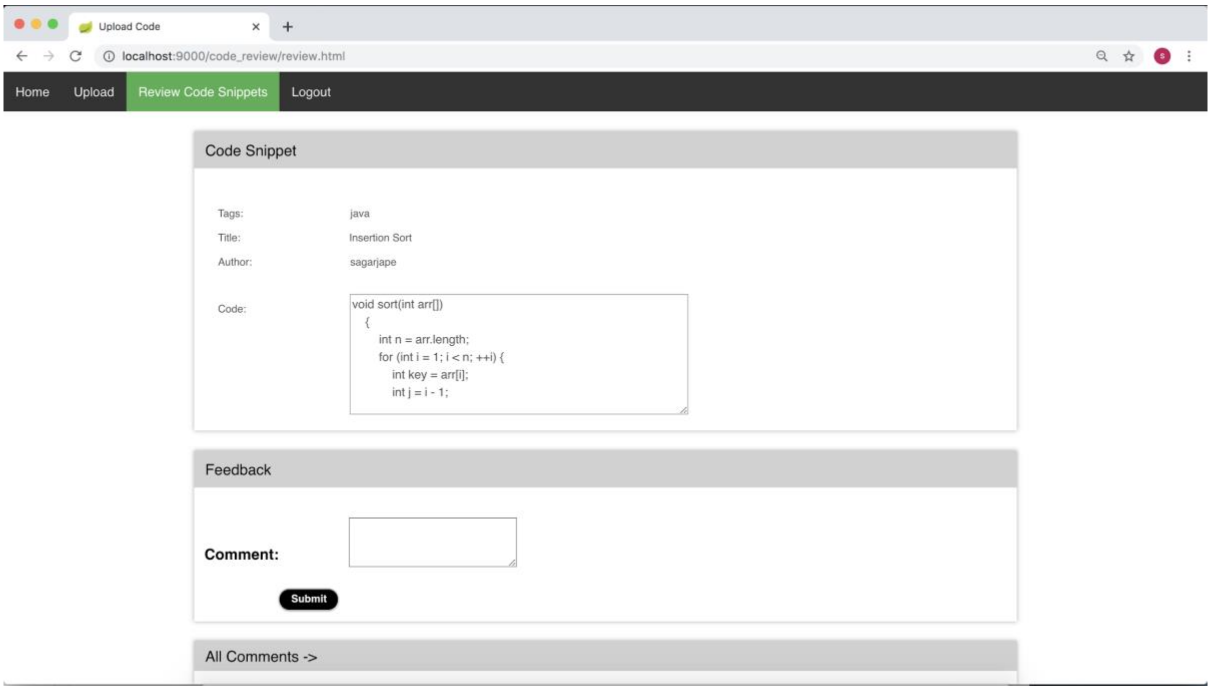


Programming Language: java
Author: sagarjape

Programming Language: java
Author: sagarjape

Programming Language: java
Author: sagarjape

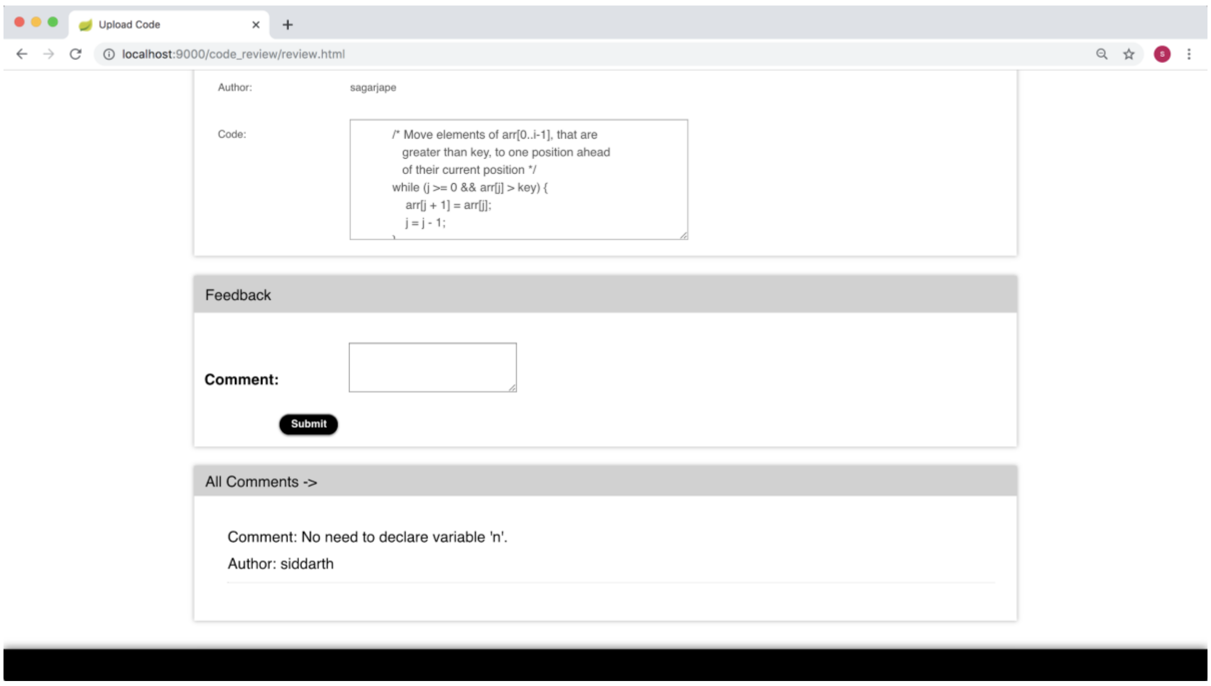
On Clicking the View Code button for a particular code on the previous screen, the user can view the code snippet. He/She can provide his own feedback related to the code snippet.



Tags: java
Title: Insertion Sort
Author: sagarjape
Code:

```
void sort(int arr[])
  {
    int n = arr.length;
    for (int i = 1; i < n; ++i) {
      int key = arr[i];
      int j = i - 1;
```

The user can also view all the feedback given by other users on a particular code snippet.



Author: sagarjape
Code:

```
      /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
      while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j = j - 1;
```

Comment: No need to declare variable 'n'.
Author: siddarth