Code_</g/re/p>

Work In Progress Document

Member Roll number	Member Name
2018007	Aditya Singh Rathore
2018069	Pranav Jaggarwal
2018100	Shikhar Tiwari
2018320	Tanupriya Das
2018357	Pragya Gandhi

Week 1

10th January [Meeting 1]

After a lot of discussion on selection of project, following points were emphasized :

- This thing is 40% of total marking.
- There are 10% of Bonus marks. Innovation will play a huge role here.
- As we are devoting lot of time here, this can serve as a good project that we have done.

We all pitched in some ideas and finally narrowed down the options to the following :

- A shoe store that has multiple sellers.
- A Banking System.

Next meeting is planned for 12th January. Till then, we will all look for examples of these types of systems.

11th January [Meeting 2]

We all realized that we are basically reinventing the wheel. There was nothing innovative in implementing the a shoe store or a Banking System. However, while exploring for other ideas, we saw Github. We realised it would be a nice idea to work with the codes like github but also have additional features which Github lacked.

We decided to make a databases for programmers to provide them with the details of the code, provide a platform for them to share it and to ask their queries. The interesting part would be to create a database of functions, classes, interfaces, Packages and Libraries. Here is what we planned :

- User can create a repository. They can add **description** and **tags** to this repository. For eg. `Sorting-algorithms`.
- We chose the syntax of Java as our base for parsing the code.
- When the user creates a library or a package, he/she can add **description** to it along with **tags**.
- When the user writes code of a class in a file, they are required to add comments in a specific style, similar to `javadoc`.
- The information can be extracted by parsing the code. User can add **tags** and description about classes. For a function, he/she can explain `return` type and `parameters`.
- All this information will be stored in database.
- Now, if somebody searches `sorting`. We will display relevant classes/functions/packages/library etc.

We also identified the following additional features :

- A **Bug tracker** system where somebody can submit their bugs and ask for help. Others can follow a bug and if it feels good, follow it.
- A **Q&A** system where you can ask questions and find answers to them.
- User can also put price on his/her code and sell it.
- A user will have **reputations** which dictate behavior on the system.

We also identified the following stakeholders :

- **Programmers** : They can ask questions, share codes, solve bugs, ask for bugs, get their codes documented etc. As they are the main users, they will act as moderators,editors in the forum.
- **Communities** : Open source communities, educators etc., can share their codes in a much better way. They can also get contributions. They can share their knowledge through a **Q&A** portal.
- **Companies**: Companies can buy and sell codes. We have options for keeping a repository `public` or `private`. Companies can document their codes there.
- **Recruiters**: Recruiters can see how many bugs a user has solved, his reputation, his answers and his repositories and then can directly contact them,if they want.

Week 2

15th January [Meeting 3]

Here are the question our system will answer to stakeholders:

- **Programmers**: Our system is for programmers, by programmers, of the programmers.
    - Programmers can document their codes in a convenient manner.
    - They can search for an algorithm or data structure and see its implementation
    - Ask questions in community forums
    - Solve and ask for solutions to bugs
    - Sell their codes and earn money
    - Gain employment
    - Become moderators and editors in the forum.
- **Communities**: By communities, we identified open source communities and educators

    - The communities can share their code. As we have a tag system, their code will be searchable.
    - They can ask and answer their own question, sharing their knowledge.
    - Find code from other communities.
    - Ask help in solving a bug.
    - Reputed communities can act as moderators and editors in Q&A website.
- **Companies**: Companies can buy codes and sell codes. They can also get a very good documentation of their code.
    - As the code is organized very well, a programmer of company won't spend time looking through code to what it does.
    - Company can buy a piece of code from a free lance coder.
    - Company can offer prizes on solving bugs in their code.
    - Companies can write pieces of software that others may be interested in and earn money.
    - Start a community driven project.
- **Recruiters** : The recruiter is here to find good programmers. Our system will answer their queries as we can **profile user data**. We can identify a good programmer by number of bugs solved etc. Here are questions that our system can answer for recruiter which will help him/her find a good programmer.
    - To find a programmer with reputation > x.
    - To find someone who has implemented a software, for eg. game engine or worked on it.
    - To find someone who has solved at least y bugs with tag says, "POSIX Threads".
    - To find the user who answered a particular question which had signifacant number of upvotes.
    - To find someone who has sold some codes or their repositories that are shared by many.

Following are the data entities that we have identified:

- Repository
- File
- Users
- Tags
- Questions
- Answers
- Bugs
- Library
- Package
- Class
- Functions
- Data (Class Variables)
- Interfaces
- Exceptions
- Transactions

Work started for Database Schema

Week 3

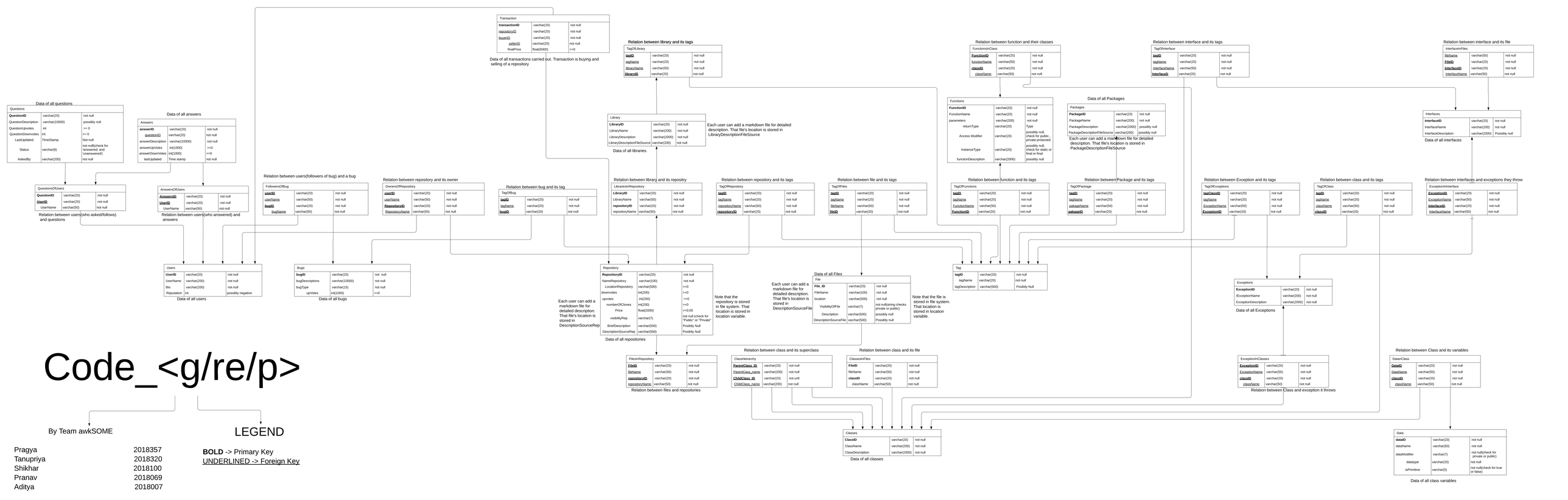
23rd January [Meeting 4]

Here are the role assigned to each member :

Roll number	Name	Role
2018007	Aditya	Implement the string parser and Bug Tracker page
2018069	Pranav	Q&A page and user page
2018100	Shikhar	Q&A page and user page
2018320	Tanupriya	Database for managing personal accounts, home page and bug tracker as user Interface.
2018357	Pragya	Databases for transactions and managing repositories as well as User Interface

Next we designed the database schema. Apart from data entities identified last week, we also identified some relations as entities that were frequent in queries. We drew the schema and implemented it.

Next Page for Schema


Transaction
transactionID varchar(20) not null
repository varchar(20) not null
buyerID varchar(20) not null
sellerID varchar(20) not null
finalPrice float(5000) >=0
Data of all transactions carried out. Transaction is buying and selling of a repository
Relation between library and its tags
TagOfLibrary
tagID varchar(20) not null
tagName varchar(20) not null
libraryName varchar(50) not null
libraryID varchar(20) not null
Library
LibraryID varchar(20) not null
LibraryName varchar(200) not null
LibraryDescription varchar(2000) not null
LibraryDescriptionFileSource varchar(200) not null
Each user can add a markdown file for detailed description. That file's location is stored in LibraryDescriptionFileSource
Data of all libraries
Relation between function and their classes
FunctionsInClass
FunctionID varchar(20) not null
functionName varchar(50) not null
classID varchar(20) not null
className varchar(50) not null
Functions
FunctionID varchar(20) not null
FunctionName varchar(20) not null
parameters varchar(200) not null
returnType varchar(20) Type
Access Modifier varchar(20) possibly null, check for public , private protected
InstanceType varchar(20) possibly null, check for static or final or final
functionDescription varchar(2000) possibly null
Relation between interface and its tags
TagOfInterface
tagID varchar(20) not null
tagName varchar(20) not null
InterfaceName varchar(50) not null
InterfaceID varchar(20) not null
Relation between interface and its file
InterfaceInFiles
fileName varchar(50) not null
FileID varchar(20) not null
InterfaceID varchar(20) not null
InterfaceName varchar(50) not null
Data of all packages
Packages
PackageID varchar(20) not null
PackageName varchar(200) not null
PackageDescription varchar(2000) possibly null
PackageDescriptionFileSource varchar(200) possibly null
Each user can add a markdown file for detailed description. That file's location is stored in PackageDescriptionFileSource
Interfaces
InterfaceID varchar(20) not null
InterfaceName varchar(200) not null
InterfaceDescription varchar(2000) Possibly null
Data of all interfaces
Data of all questions
Questions
QuestionID varchar(20) not null
QuestionDescription varchar(10000) possibly null
QuestionUpvotes int >= 0
QuestionDownvotes int >= 0
LastUpdated TimeStamp Not null
Status varchar(8) not null(check for 'answered, and 'unanswered')
AskedBy varchar(200) not null
Data of all answers
Answers
answerID varchar(20) not null
questionID varchar(20) not null
answerDescription varchar(10000) not null
answerUpVotes int(1000) >=0
answerDownVotes int(1000) >=0
lastUpdated Time stamp not null
QuestionsOfUsers
QuestionID varchar(20) not null
UserID varchar(20) not null
UserName varchar(50) not null
Relation between users(who asked/follows) and questions
AnswersOfUsers
AnswerryID varchar(20) not null
UserID varchar(20) not null
UserName varchar(50) not null
Relation between users(who answered) and answers
Relation between users(followers of bug) and a bug
FollowersOfBug
userID varchar(20) not null
userName varchar(50) not null
bugID varchar(20) not null
bugName varchar(50) not null
Relation between repository and its owner
OwnersOfRepository
userID varchar(20) not null
userName varchar(50) not null
RepositoryID varchar(20) not null
RepositoryName varchar(50) not null
Relation between bug and its tag
TagOfBug
tagID varchar(20) not null
tagName varchar(20) not null
bugID varchar(20) not null
Relation between library and its repository
LibrariesInRepository
LibraryID varchar(20) not null
LibraryName varchar(50) not null
repositoryID varchar(20) not null
repositoryName varchar(50) not null
Relation between repository and its tags
TagOfRepository
tagID varchar(20) not null
tagName varchar(20) not null
repositoryName varchar(50) not null
repositoryID varchar(20) not null
Relation between file and its tags
TagOfFiles
tagID varchar(20) not null
tagName varchar(20) not null
fileName varchar(50) not null
fileID varchar(20) not null
Relation between function and its tags
TagOfFunctions
tagID varchar(20) not null
tagName varchar(20) not null
FunctionName varchar(50) not null
FunctionID varchar(20) not null
Relation between Package and its tags
TagOfPackage
tagID varchar(20) not null
tagName varchar(20) not null
pakageName varchar(50) not null
pakageID varchar(20) not null
Relation between Exception and its tags
TagOfExceptions
tagID varchar(20) not null
tagName varchar(20) not null
ExceptionName varchar(50) not null
ExceptionID varchar(20) not null
Relation between class and its tags
TagOfClass
tagID varchar(20) not null
tagName varchar(20) not null
className varchar(50) not null
classID varchar(20) not null
Relation between Interfaces and exceptions they throw
ExceptionInInterface
ExceptionID varchar(20) not null
ExceptionName varchar(50) not null
InterfaceID varchar(20) not null
InterfaceName varchar(50) not null
Users
UserID varchar(20) not null
UserName varchar(200) not null
Bio varchar(200) not null
Reputation int possibly negative
Data of all users
Bugs
bugID varchar(20) not null
bugDescriptions varchar(10000) not null
bugType varchar(15) not null
upVotes int(1000) >=0
Data of all bugs
Repository
RepositoryID varchar(20) not null
NameRepository varchar(100) not null
LocationRepository varchar(500) >=0
downvotes int(200) >=0
upvotes int(200) >=0
numberOfClones int(200) >=0
Price float(2000) >=0.00
visibilityRep varchar(7) not null (check for "Public" or "Private"
BriefDescription varchar(500) Possibly Null
DescriptionSourceRep varchar(500) Possibly Null
Each user can add a markdown file for detailed description. That file's location is stored in DescriptionSourceRep
Note that the repository is stored in file system. That location is stored in location variable.
Data of all repositories
Data of all Files
File
File_ID varchar(20) not null
FileName varchar(100) not null
location varchar(500) not null
VisibilityOfFile varchar(7) not null(string checks private or public)
Description varchar(500) possibly null
DescriptionSourceFile varchar(500) Possibly null
Each user can add a markdown file for detailed description. That file's location is stored in DescriptionSourceFile
Note that the file is stored in file system. That location is stored in location variable.
Tag
tagID varchar(20) not null
tagName varchar(20) not null
tagDescription varchar(500) Possibly Null
Exceptions
ExceptionID varchar(20) not null
IExceptionName varchar(200) not null
ExceptionDescription varchar(2000) not null
Data of all Exceptions
FilesInRepository
FileID varchar(20) not null
fileName varchar(50) not null
repositoryID varchar(20) not null
repositoryName varchar(50) not null
Relation between files and repositories
ClassHierarchy
ParentClass_ID varchar(20) not null
ParentClass_name varchar(200) not null
ChildClass_ID varchar(20) not unll
ChildClass_name varchar(200) not null
Relation between class and its superclass
ClassesInFiles
FileID varchar(20) not null
fileName varchar(50) not null
classID varchar(20) not null
className varchar(50) not null
Relation between class and its file
Classes
ClassID varchar(20) not null
ClassName varchar(200) not null
ClassDescription varchar(2000) not null
Data of all classes
ExceptionInClasses
ExceptionID varchar(20) not null
ExceptionName varchar(50) not null
classID varchar(20) not null
className varchar(50) not null
Relation between Class and exception it throws
DataInClass
DataID varchar(20) not null
DataName varchar(50) not null
classID varchar(20) not null
className varchar(50) not null
Relation between Class and its variables
Data
dataID varchar(20) not null
dataName varchar(50) not null
dataModifier varchar(7) not null(check for private or public)
datatype varchar(20) not null
isPrimitive varchar(5) not null(check for true or false)
Data of all class variables
Code_<g/re/p>
By Team awkSOME
Pragya 2018357
Tanupriya 2018320
Shikhar 2018100
Pranav 2018069
Aditya 2018007
LEGEND
BOLD -> Primary Key
UNDERLINED -> Foreign Key

Week 4️⃣

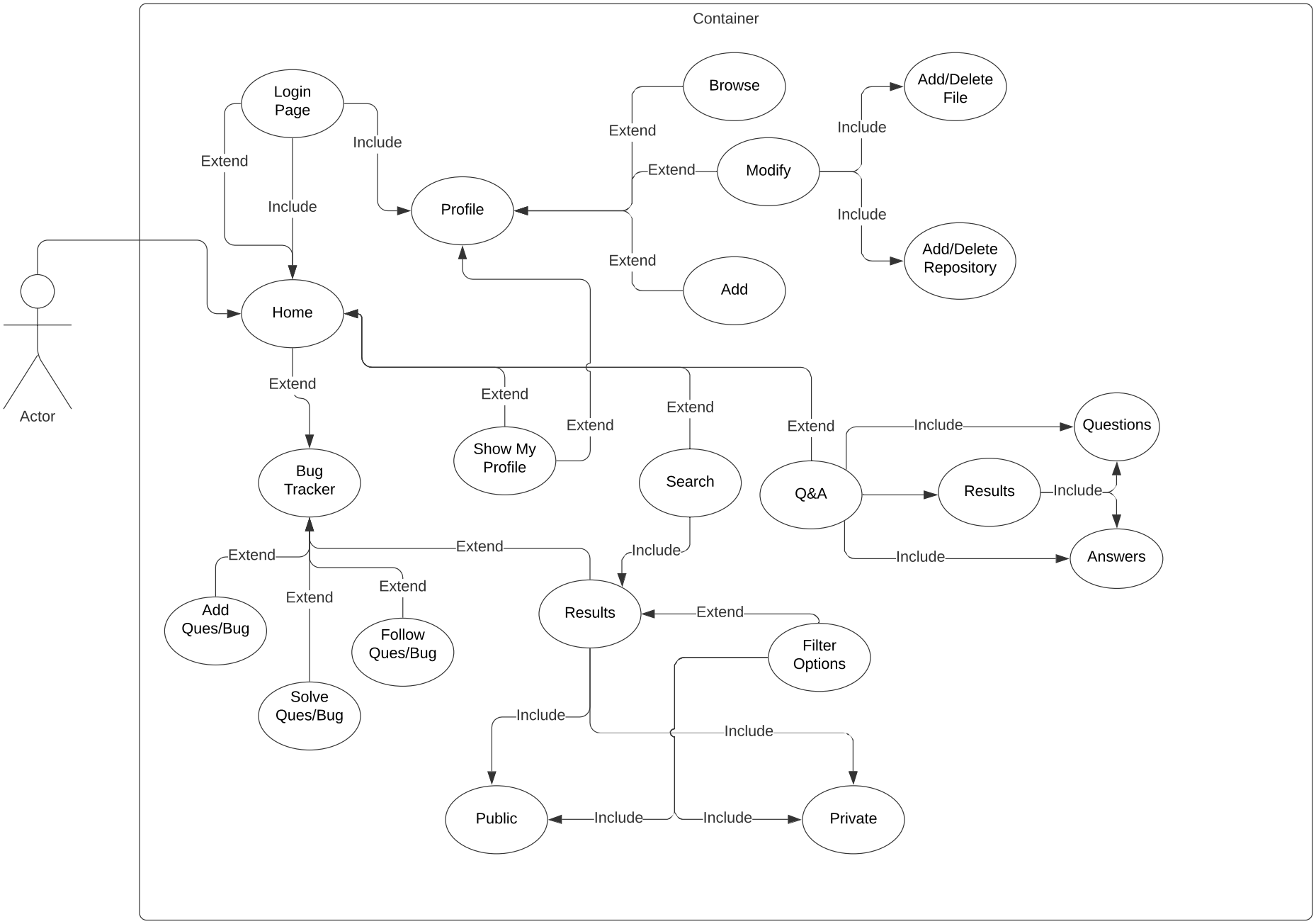
30th January [Meeting 5]

We made the tables on sql using Data Grip. We had also set up constaints for each table. We also worked on the use case diagram for our application. We also brain stormed about the User Interface for our application. After making the tables in sql, we went ahead with populating the data. We gave 20 entries in each table. We divided the work equally among ourselves.

Here is the role each member performed after Week 4:

Roll number	Name	Role
2018007	Aditya	Designed the schema and Use Case. Worked on population of data
2018069	Pranav	Made the tables on Data Grip
2018100	Shikhar	Made the tables on Data Grip and populated the data
2018320	Tanupriya	Prepared the Use Case and populated the data
2018357	Pragya	Designed the schema and Use Case. Worked on population of data

code_grep


Container
Actor
Login Page
Home
Profile
Browse
Modify
Add
Add/Delete File
Add/Delete Repository
Extend
Include
Bug Tracker
Show My Profile
Search
Q&A
Questions
Results
Answers
Add Ques/Bug
Follow Ques/Bug
Solve Ques/Bug
Results
Filter Options
Public
Private

Week 7:

We created indexes for our tables. Indexes were created by giving complete thought to the optimization.

The tables and their indexes are:

- answers ID,QUESTION_ID,ANSWERED_BY
- buginfile: FILEID
- bugs: ID
- classes: ID
- classhierarchy: CLASSID
- classimplementsinterface : CLASSID
- classinfile : FILEID
- downvoterofbugs : BUGID
- downvotersofanswers
- downvotersofquestions : QUESTIONID
- downvotersofrepository : REPOSITORYID
- exceptions : ID
- exceptionthrownbyfunctions : FUNCTIONID
- files : ID,LOCATION
- filesinrepository : FILEID
- followerofanswer : ANSWERID
- followerofbugs : BUGID
- followerofquestions : QUESTIONID
- followerofrepositories : REPOSITORYID
- functions : ID
- functionsinclasses : FUNCTIONID
- functionsininterfaces : FUNCTIONID
- interfaceinfile : ID
- interfaces : ID
- objectdefinitioninclass : CLASSID
- objects : ID
- parameters_of_function
- questions : ID,QUESTION, ASKED_BY
- repositories : ID
- tags : ID,NAME
- tagsofanswers
- tagsofbugs
- tagsofclasses : CLASSID
- tagsofexceptions : EXCEPTIONID
- tagsoffiles : FILEID
- tagsoffunctions : FUNCTIONID

- tagsofinterfaces: INTERFACEID
- tagsofobjects : OBJECTID
- tagsofquestions : QUESTIONID
- tagsofrepositories : ANSWERID
- transaction : ID, REPOSITORYID,BUYER_ID,SELLER_ID
- upvotersofanswers : ANSWERID
- upvotersofbugs : QUESTIONID
- upvotersofquestions : QUESTIONID
- upvotersofrepository : REPOSITORYID
- users : ID, USER_ID

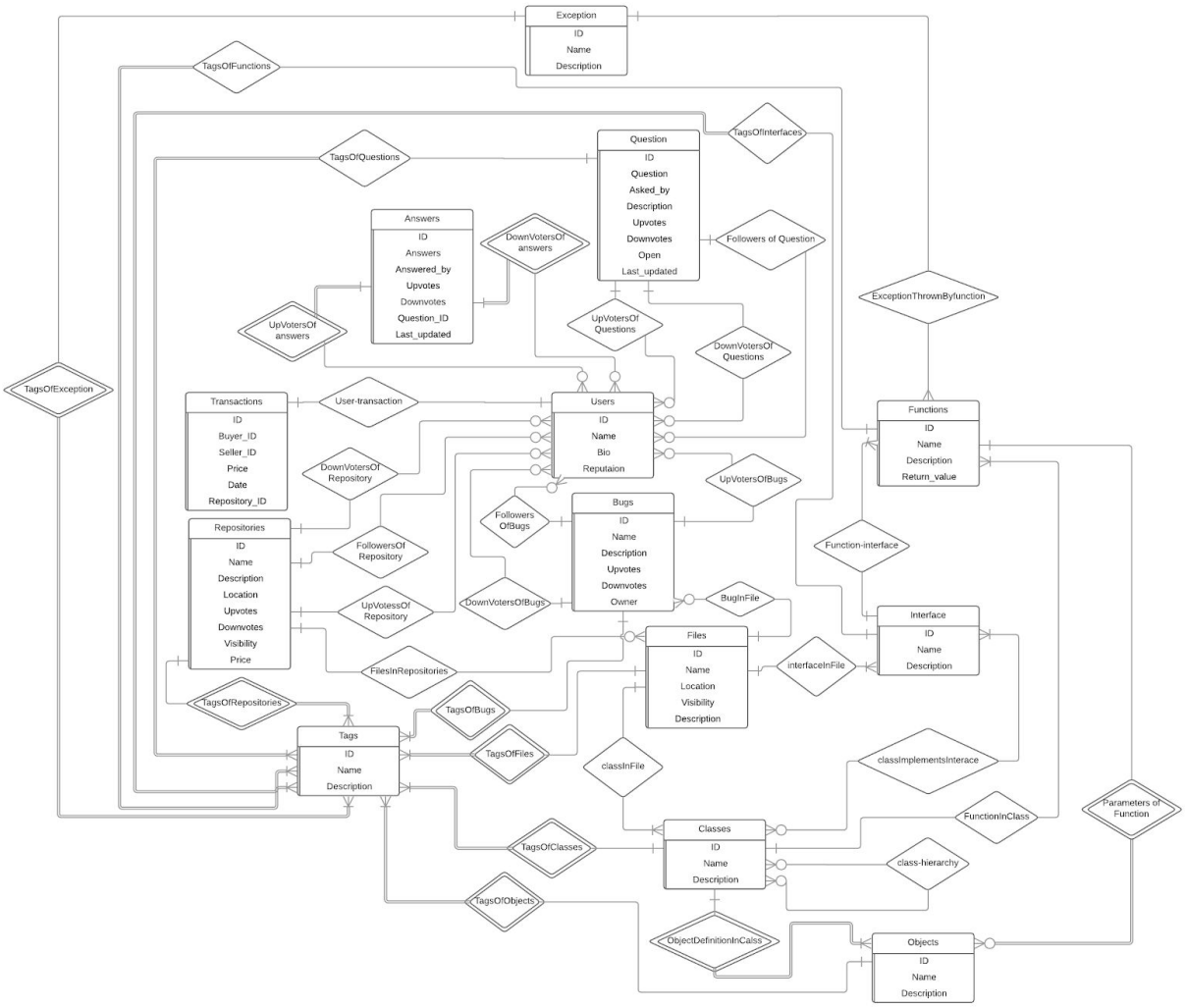
We also wrote the relational queries. They are listed below:

- $\sigma_{Name= \text{'Raymond'}}(repositories)$
- $\sigma_{Upvotes> 7000 \text{ and } Downvotes < 3000}(questions)$
- $\pi_{Name}(\sigma_{users.ID= Questions.Asked\_by}(\sigma_{Questions.Upvotes > 5000}(users) x (questions))$
- $\pi_{Name}(\sigma_{Price < 8000}(repositories)) - \pi_{Name}(\sigma_{Downvotes < 4000}(Students))$
- $\sigma_{Upvotes> 7000 \text{ and } Downvotes < 3000}(questions)$
- $\pi_{Name}(\sigma_{bugs.ID= tagsofbugs.BugID}(bugs))$
- $\pi_{Name}(\sigma_{Reputation> 7000}(users))$
- $\pi_{Name}(\sigma_{Visibility= 1}(files))$
- $\sigma_{Object\_ID= \text{'object87'}}(parameters\_of\_function)$
- $\pi_{ID(users)} - \pi_{Buyer\_ID(transaction)}$
- $\pi_{Name}(\sigma_{bugs.ID= tagsofbugs.BugID}(bugs))$

Week 8:

The ER model is attached below

Code_Grep ER Diagram



The Strong entities identified are:

- Repositories
    - ID
    - Name
    - Description
    - Location
    - Upvotes
    - Price
    - Downvotes
    - Visibility
- Files
    - ID
    - Name
    - Location

    - Visibility
    - Description

- Bugs
    - ID
    - Name
    - Description
    - Upvotes
    - Downvotes
    - Owner
- Question
    - ID
    - Question
    - Asked_by
    - Description
    - Upvotes
    - Downvotes
    - Open
    - Last_updated
- Users
    - ID
    - Name
    - Bio
    - Reputation
- Transactions
    - ID
    - Buyer_ID
    - Seller_ID
    - Price
    - Repository_ID
    - Date
- Interface
    - ID
    - Name
    - Description
- Exception
    - ID
    - Name
    - Description
- Classes
    - ID
    - Name
    - Description

- Functions
    - ID
    - Name
    - Description
    - Return_value

And the weak entities identified are:

- Tags
    - ID
    - Name
    - Description
- Answers
    - ID
    - Answers
    - Answered_by
    - Upvotes
    - Downvotes
    - Question_ID
    - Last_updated
- Objects
    - ID
    - Name
    - Description

Relations used are:

ParametersOfFunction
Exception-function
Function-interface
User-Transaction
User-Repository
User-Question
User-Question-Follow
User-question-upvotes
User-question-Downvotes
User-Answer
User-Answer-Follow
User-Answer-Upvotes
User-Answer-Downvotes
User-Bugs
User-Bugs-Follow
User-Bugs-Upvotes
User-Bugs-Downvotes
User-Repository-Follow

Tags of repositories
Tags of file
Tags of class
Tags of exceptions
Tags of interface
Tags of functions
Tags of object
Tags of questions
Tags of answers
Tags of bugs
Transactions of repositories
Class-function
Class-object
Class-file
Class hierarchy
Class-interface
Interface-file
Bugs-File
Repository-file

Week 10:

The key innovative feature. We planned on implementing tag hinting. It basically involves providing a hint to the user for a particular tag searched.
Another feature we had added was password encryption which helps with the security of users accounts.

If given more time we could have implemented a string parser and file system to support our app.