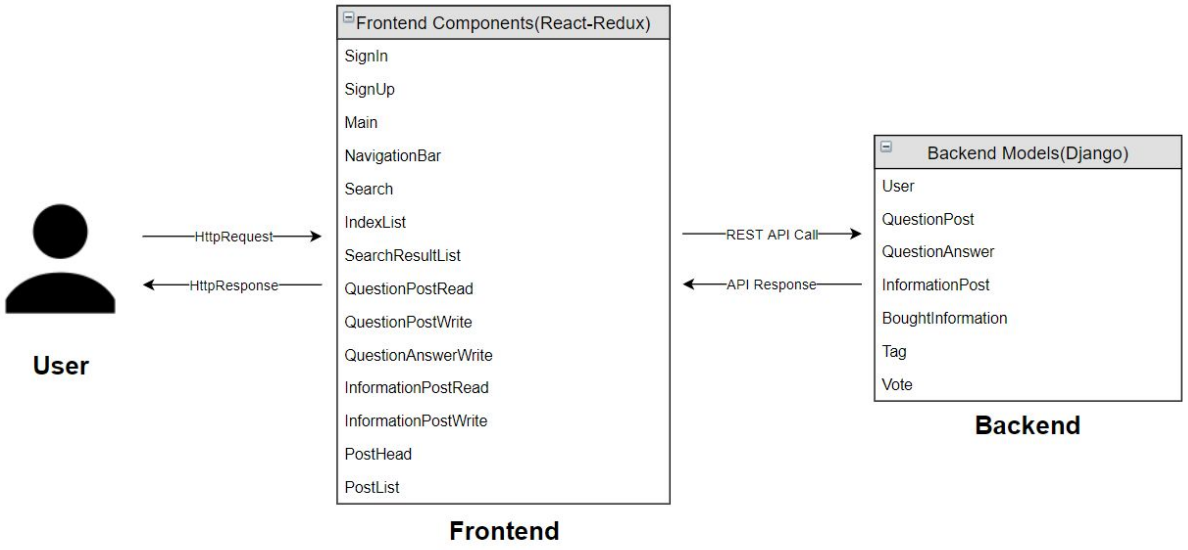
SNUwagon

Design and Planning

Group 4
김도윤 김진표 이지섭 최경재

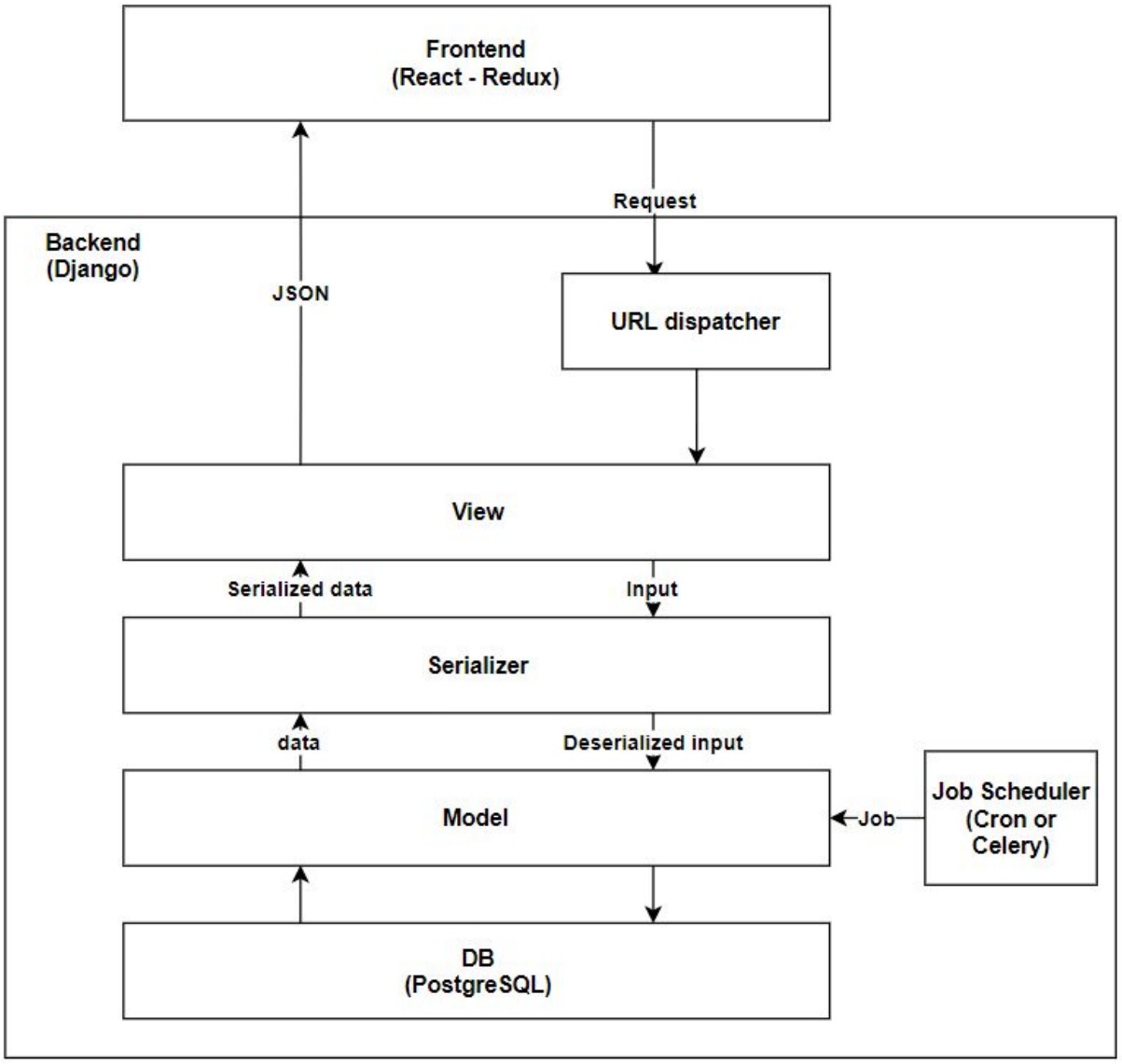
Table of Contents

1. System Architecture



This is our overall architecture diagram. For frontend, React and Redux frameworks will be used. For backend, Django REST framework will be used.

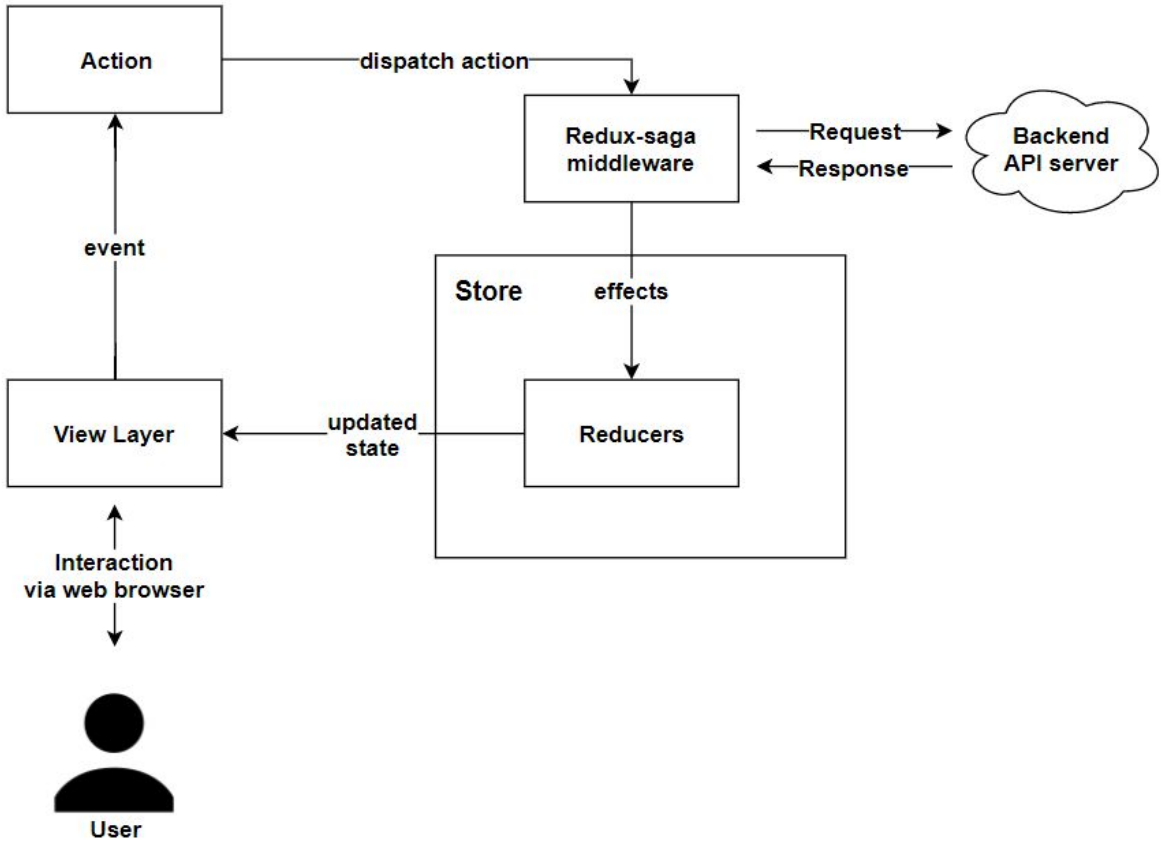
1.1. Backend Architecture



Since SQLite3, the default DB setting of Django, shows a bad performance when handling concurrent jobs, we decided to use PostgreSQL for backend DB. By using psycopg2, Django fully supports PostgreSQL.

Job Scheduler handles periodic events such as updating the states of models.

1.2. Frontend Architecture

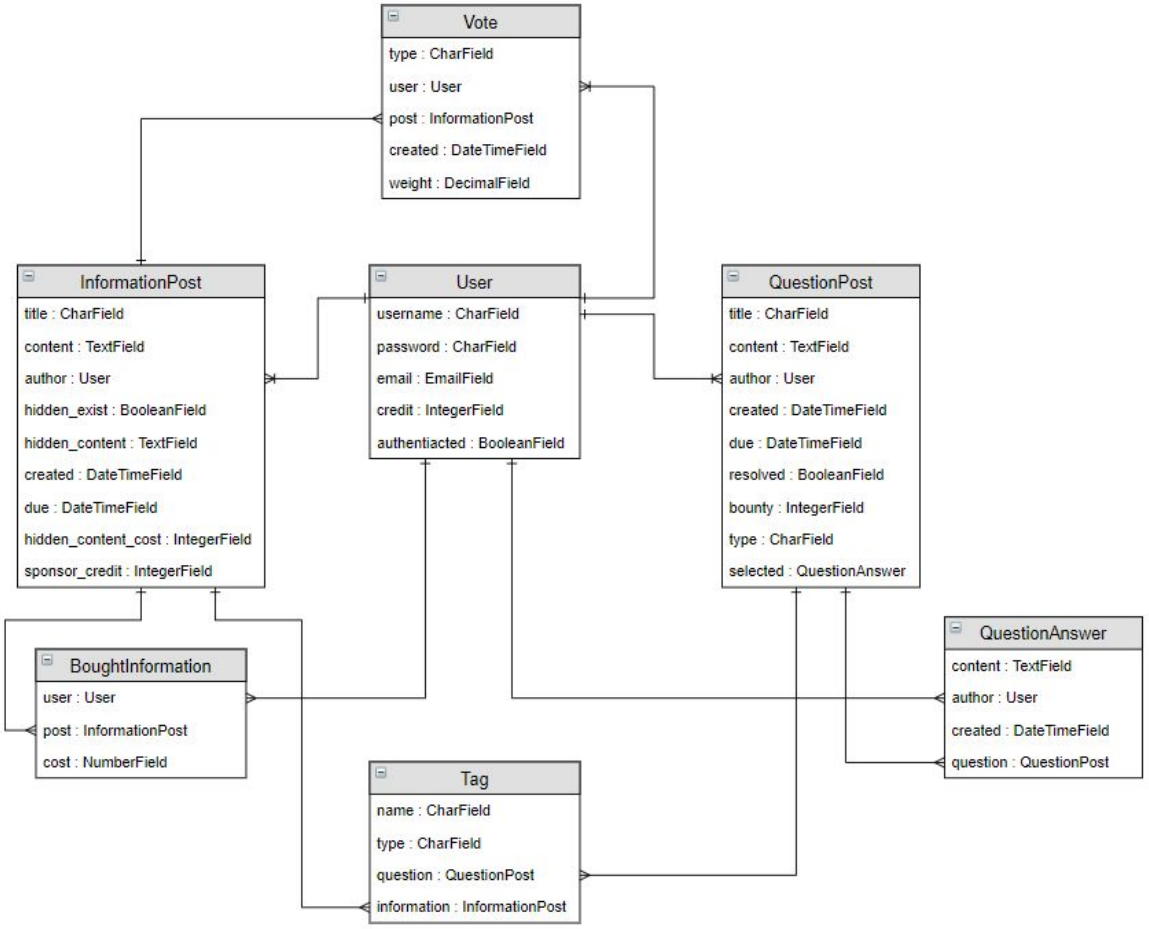


React and Redux framework will be used to implement a Flux architecture frontend. Redux-saga will handle communications with the backend server.

2. Design Details

2.1. Backend

2.1.1. Backend Model Diagram



This is the model diagram for the backend models. Each rectangle represents a Django model and each arrow represents a Foreign Key relationship. The Primary Key of each model is it's id. Since Django automatically generates the model id, it is omitted in the diagram.

Models

1. User

The User model represents an individual user. The 'email' field is used for user verification, and the 'authenticated' field indicates whether this user is valid or not.

2. QuestionPost

The QuestionPost model represents a single question. All questions in our service are private as of now, but the 'type' field is added for extensibility in future implementations. The 'selected' field holds the QuestionAnswer instance which corresponds to the answer chosen by the author of this question.

3. QuestionAnswer

The QuestionAnswer model represents a single answer to a question. The ‘question’ field indicates which question this answer is resolving.

4. InformationPost

The InformationPost model corresponds to an information post. The ‘hidden_exist’ field indicates whether this information post has a hidden content to it. If so, the ‘hidden_content’ field holds that content, and the ‘hidden_content_cost’ field holds how many credits it takes to view the content. ‘sponsor_credit’ indicates how much the author of this information post will spend to get his/her post featured in our main page.

5. Tag

The Tag model represents a single tag. The ‘type’ field indicates whether this tag is for a question post or an information post.

6. Vote

The Vote model represents a single vote to an information post. The ‘type’ field indicates whether this vote is an upvote or a downvote. Currently all votes have an equal weight in our service, but the ‘weight’ field is added for extensibility in future implementations.

7. BoughtInformation

The BoughtInformation model represents a purchased piece of hidden content from an information post. The ‘cost’ field corresponds to the price of the content.

2.1.2. Backend API Table

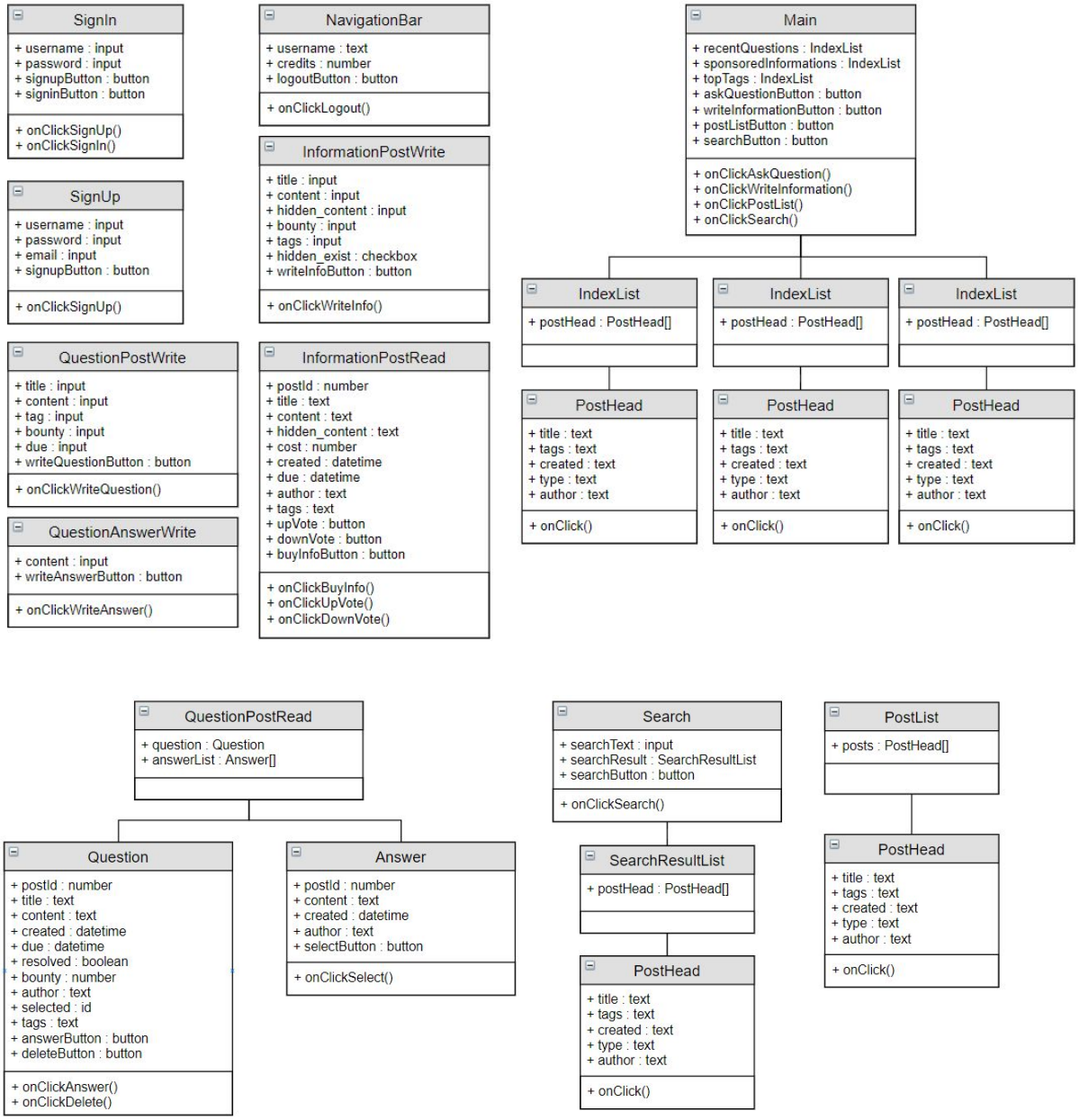
TYPE	URL	About
POST	/api/auth/signin	Perform login and create session with backend
POST	/api/auth/signup	Create new user account
GET	/api/auth/userinfo/:id	Get information about user with specific id
GET	/api/posts/index	Get information about data to show on index
POST	/api/posts/question	Write new Question
GET	/api/posts/question/:id	Get Question object correspond to specific id
DELETE	/api/posts/question/:id	Delete Question object correspond to specific id
GET	/api/posts/information/:id	Get Information object correspond to specific id
POST	/api/posts/information	Write new Information

TYPE	URL	About
GET	/api/list/questions	Get list of Questions
GET	/api/list/questions/tag/:tag	Get list of Questions with specific tag
GET	/api/list/questions/type/:type	Get list of Questions with specific type
GET	/api/list/questions/title/:title	Get list of Questions with specific title
GET	/api/list/information	Get list of Informations
GET	/api/list/informations/tag/:tag	Get list of Informations with specific tag
GET	/api/list/informations/type/:type	Get list of Informations with specific type
GET	/api/list/informations/title/:title	Get list of Informations with specific title
GET	/api/list/all	Get list of Questions and Informations
GET	/api/list/all/tag/:tag	Get list of Questions and Informations with specific tag
GET	/api/list/all/type/:type	Get list of Questions and Informations with specific type
GET	/api/list/all/title/:title	Get list of Questions and Informations with specific title
POST	/api/vote	Apply new vote on the information
GET	/api/vote/:id	Get list of votes with specific Information id

The table above shows the specifications of REST APIs. All data will be transferred in JSON.

2.2. Frontend

2.2.1. Frontend Components



Components

1. SignIn
   - onClickSignIn(username, password) : try signing in, if successful, change route to Main.
   - onClickSignUp() : change route to SignUp.

2. SignUp

- onClickSignUp(username, password, email) : try signing up, if successful, change route to SignIn.

3. NavigationBar
   - onClickLogout() : try logout, change route to SignIn.

4. QuestionPostWrite
   - onClickWriteQuestion(title, content, due, bounty) : try writing a question, if successful, change route to QuestionPostRead which shows the question written.

5. QuestionAnswerWrite
   - onClickAnswer(content) : try writing an answer to a question, if successful, change route to QuestionPostRead which shows the target question and answer written.

6. QuestionPostRead
   - Question
   - Answer

7. InformationPostWrite
   - onClickWriteInfo(title, content, hidden_exist, hidden_content, hidden_content_cost, due, sponsor_credit) : try writing an information, if successful, change route to InformationPostRead which shows the information written.

8. InformationPostRead
   - onClickBuyInfo(information_id) : try revealing the hidden content of an information, if successful, update view including hidden content.
   - onClickUpVote(information_id) : try upvoting an information, if successful, update view with increased up vote count.
   - onClickDownVote(information_id) : try downvoting an information, if successful, update view with increased down vote count.

9. Main
   - onClickAskQuestion() : change route to QuestionPostWrite.
   - onClickWriteInformation() : change route to InformationPostWrite.
   - onClickPostList() : change route to PostList.
   - onClickSearch() : change route to Search.
   - IndexList[]

10. Search
   - onClickSearch(query) : try searching posts, if successful, update SearchResultList with search results
   - SearchResultList

11. PostList
   - PostHead[]

Subcomponents

1. IndexList
    - PostHead[]

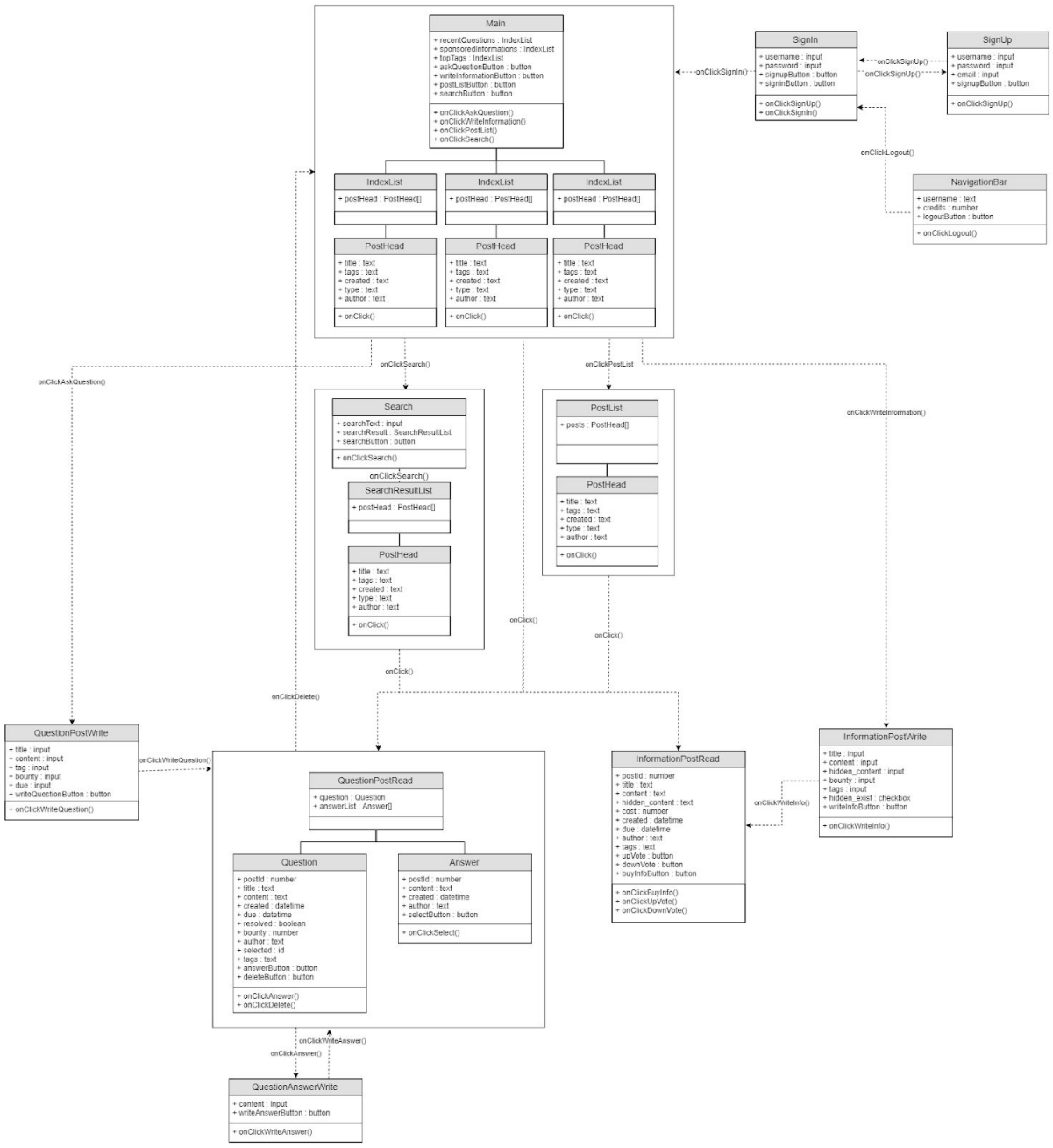
2. PostHead
    - onClick(post_id): change route to QuestionPostRead or InformationPostRead according to post_id.

3. Question
    - onClickAnswer(): change route to QuestionAnswerWrite.
    - onClickDelete(): try deleting a question, if successful, change route to Main.

4. Answer
    - onClickSelect(): try selecting the best answer, if successful, update view with highlighted selected answer.

5. SearchResultList
    - PostHead[]

2.2.2. Frontend Components Relationships



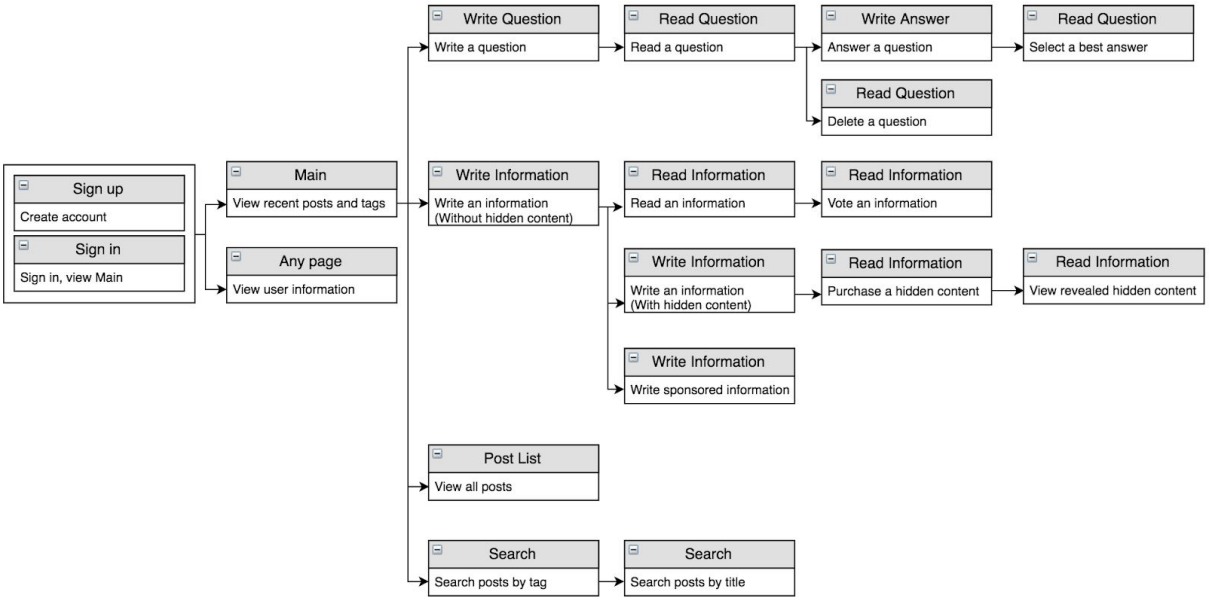
3. Implementation Plan

3.1. Current Implementation Plan

Page	Feature	Difficulty	Time (Hour)	Sprint	Details (front)	Details (back)
Sign up	Create an account	2	4	2	Input validation needed	Input validation needed
Sign in	Sign in and view Main page	1	2	2		Session or token authentication
Main	Check recent posts and tags	2	4	2		Posts/tags need to be sorted properly
Any	View user info from any page	3	3	3	User credit must be always up-to-date	
Search	Search posts by tag	3	2	3	One tag per search	
Search	Search posts by title	3	1	3		
Post List	View all questions and information posts	2	2	2	Pagination needed	
Read Question	Read a question	2	4	2		User authentication needed
Read Information	Read an information post	2	2	2		User authentication needed
Write Question	Write a question	2	4	2	Input validation needed	
Write Information	Write an information post (without hidden content)	2	2	2	Input validation needed	
Write Answer	Write an answer to a question	3	3	2	Input validation needed	
Read Question	Select the best answer	3	2	3		
Read Information	Vote	1	2	3	User should see direct feedback	Duplicate vote need to be blocked
Write Information	Write an information post (with hidden content)	3	4	3	Input validation needed	
Read Information	Buy a hidden content	3	2	3		User authentication needed
Read Information	Reveal hidden contents of bought or expired information	3	3	3		
Write Information	Write a sponsored information post	3	2	3	Input validation needed	
Read Question	Delete a question	1	1	2		User authentication needed

The table above shows the implementation plan of sprint 2 and sprint 3. Our focus is on making a running prototype system in sprint 2, and completing the rest of the features in sprint 3. Thus 'Write Information' has been further broken down into three steps, namely writing without hidden contents, writing with hidden contents, and writing a sponsored information. The basic writing feature will be implemented in sprint 2, and the two additional features are to be added in the next sprint.

There are similarities between certain tasks, such as searching by a tag or a title, or reading a question or an information. In these cases, although the objective difficulty of the tasks may be alike, implementing one feature makes the other tasks relatively easier to implement. Thus we've set the expected developing time accordingly.



The diagram above shows the dependencies between the tasks to be implemented in sprint 2 and sprint 3.

3.2. Further Implementation Plan

Page	Feature	Difficulty	Time	Sprint
Profile	View user information	3	?	4
Read Information	View author's credibility	3	?	4
Main	View notifications	5	?	4
Any	Push notification	5	?	4
Sign in	Email authentication	3	?	4
Read Information	Weighted vote	4	?	4

The table above shows our further implementation plan for sprint 4. Because these features are slightly more challenging than our basic features, depending on our time constraints, the features may be selectively implemented.

4. Testing Plan

All of the implemented code will be tested by a test code. We will write test code alongside the implementation.

4.1. Unit Test

For backend, the *unittest* module, which is Django's built-in testing tool, will be used to write unit tests. We will write tests for each unit like models or views.

For frontend, Jest will be used to write unit tests. Each component will be tested, by checking whether its inner methods are called properly, and asserting their effects.

4.2. Functional Test

For backend, Django's built-in testing tool will be used to write functional tests. We will write tests for each REST API call with mock data.

For frontend, Jest will be used to write functional tests. We will provide the input through API calls if the backend API is implemented or by using mock data.

4.3. Acceptance & Integration Test

Acceptance tests will be done after sprint 3 and sprint 4. Selenium will be used for acceptance tests.

Integration tests will be done with Travis CI. Since Travis CI offers automatic build and test, integration test will be done for each pull request.

Appendix

A. User Interface Design