SNUwagon

Design and Planning

Revision 1.1 2018-06-11

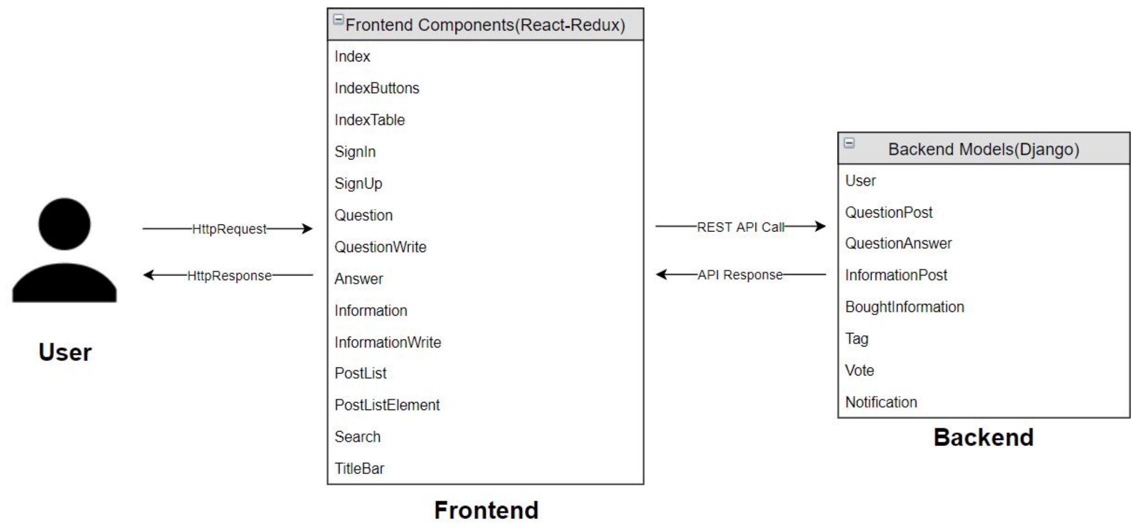
Group 4
김도윤 김진표 이지섭 최경재

Table of Contents

Revision Bullet List

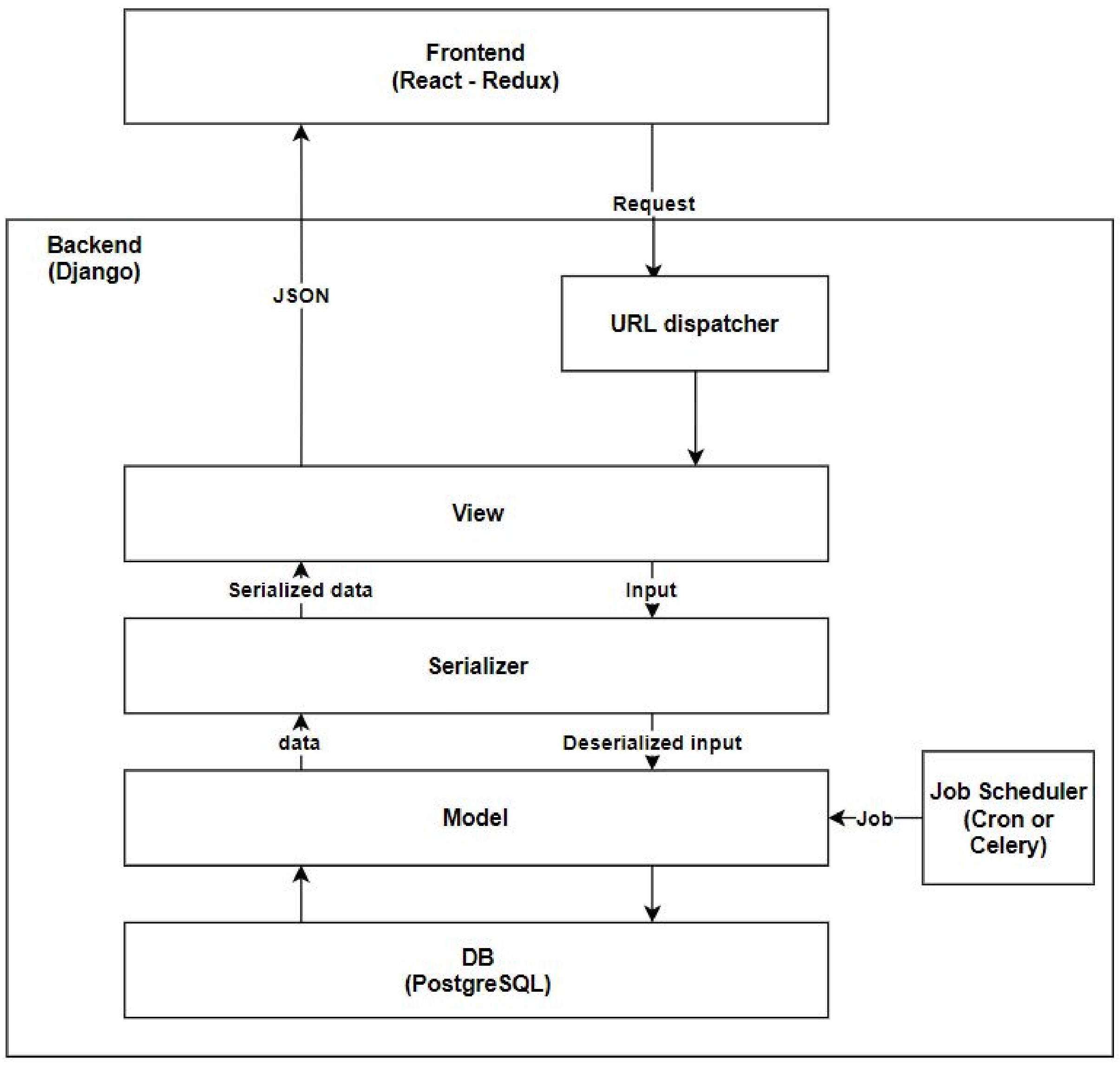
- System Architecture - update overall Frontend Components and Backend Models.
- Backend models - add Notification model and update some attributes.
- Frontend components - simplified components' hierarchy, changed their names.
- API table - updated APIs which are implemented on sprint 4.
- Implementation Plan - updated sprint 4 implementation plans.

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.

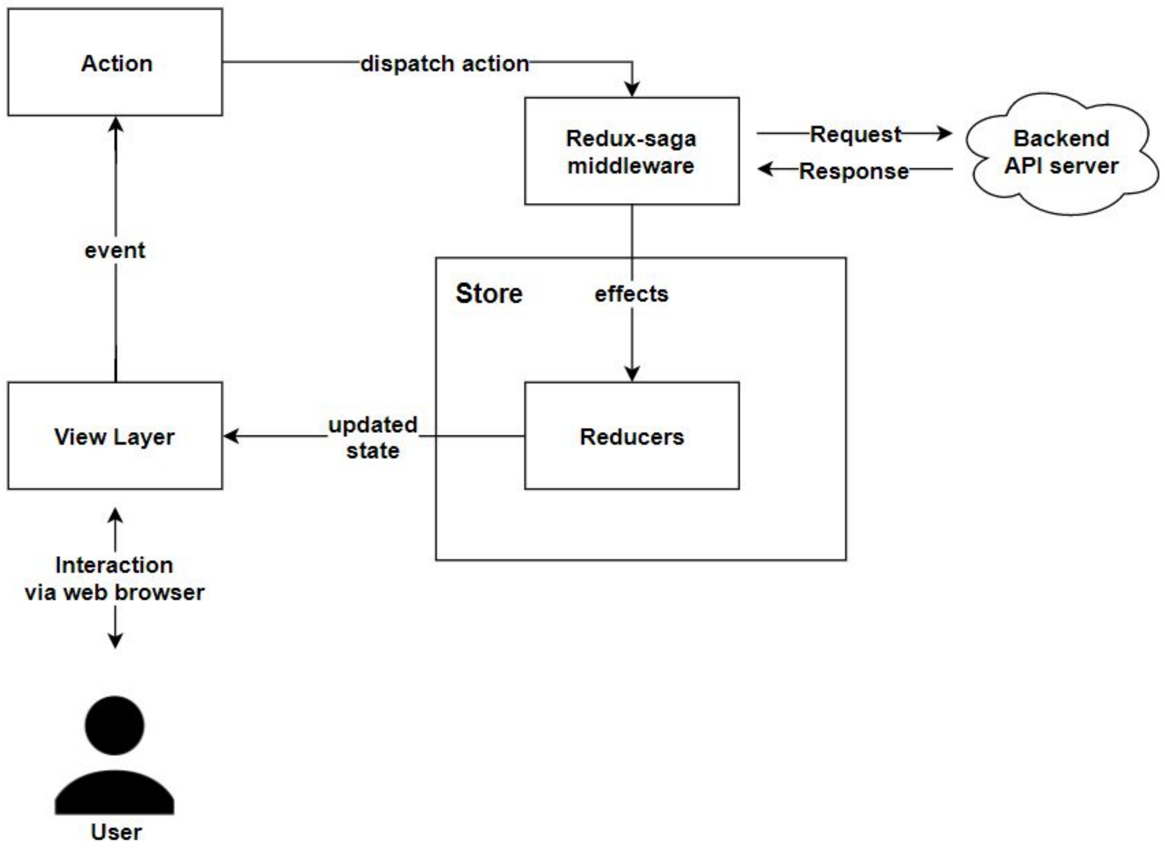
1.1. Backend Architecture



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

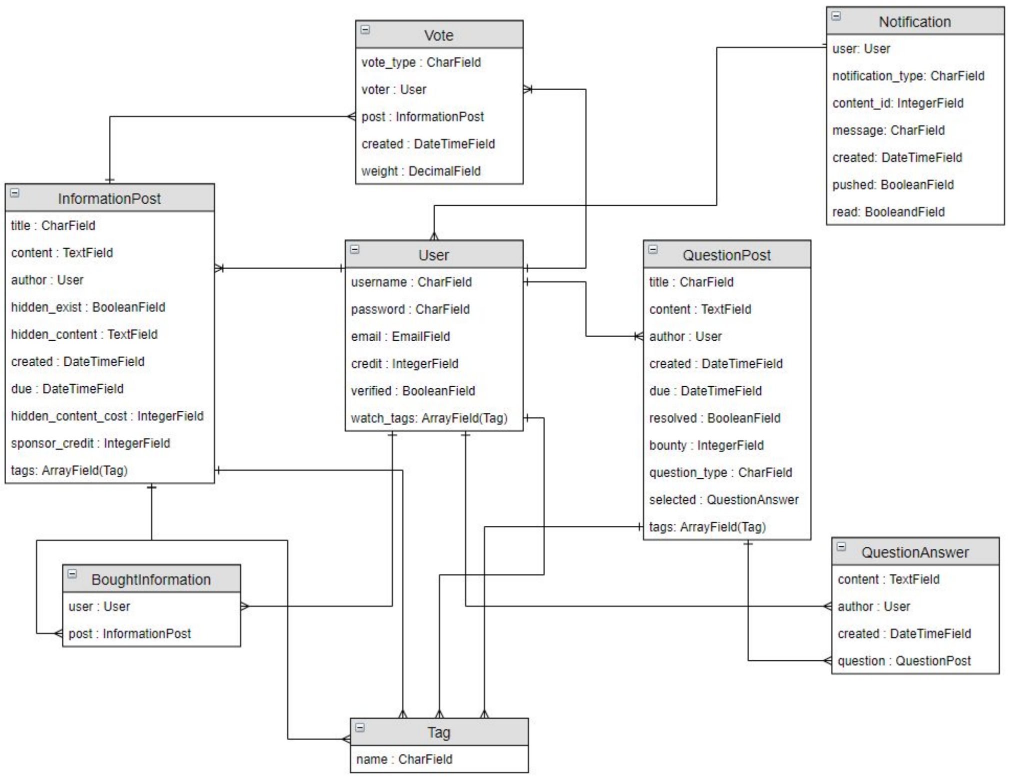


React and Redux framework will be used to implement a Flux architecture frontend. Redux-saga will handle communications with the backend server.

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 'verified' field indicates whether this user has completed email verification 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 'name' field is primary key that represents a tag.

6. Vote

The Vote model represents a single vote to an information post. The ‘vote_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.

8. Notification

The Notification model represents a notification and newsfeed information for a user. The 'notification_type' field represents what is the notification about. The 'pushed' field represents if the user have seen the notification. The 'read' field represents if the user read the notification in newsfeed section.

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

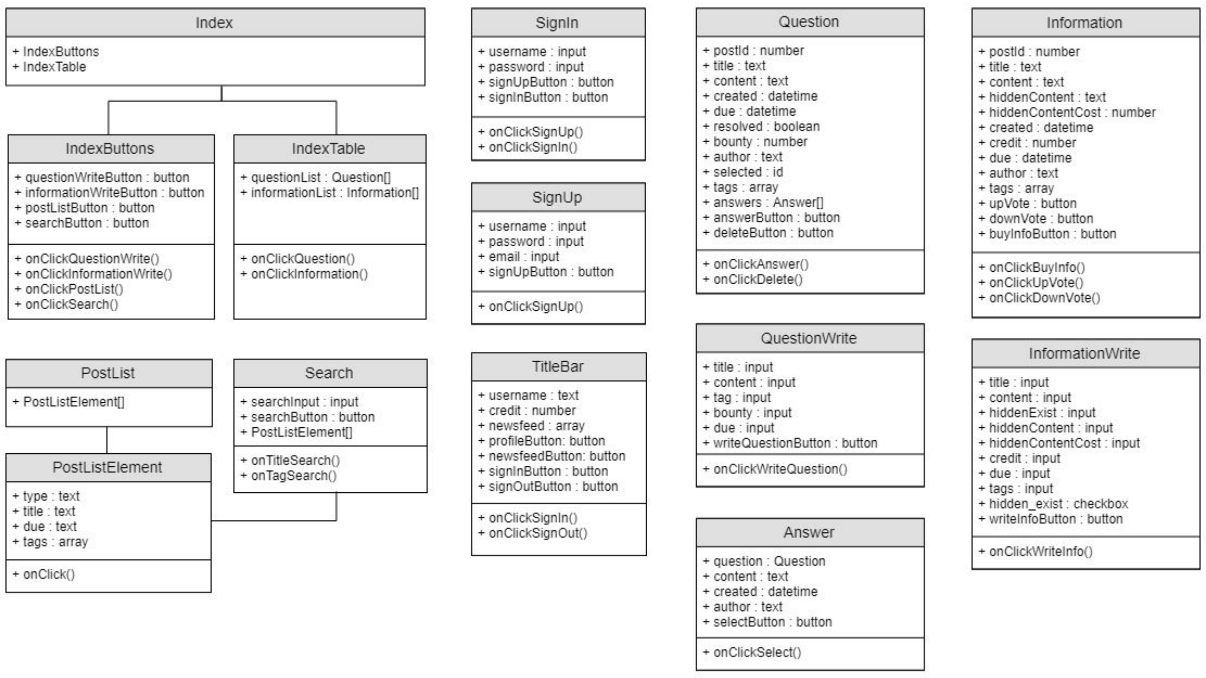
TYPE	URL	About
GET	/api/list/questions	Get list of Questions
GET	/api/list/questions/tag/:tag	Get list of Questions with specific tag
GET	/api/list/questions/title/:title	Get list of Questions with specific title
GET	/api/list/information	Get list of Informations
GET	/api/list/informations/tag/:tag	Get list of Informations with specific tag
GET	/api/list/informations/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/title/:title	Get list of Questions and Informations with specific title

TYPE	URL	About
GET	/api/list/tags	Get every existing tags
GET	/api/notifications	Get list of new notifications
GET	/api/newsfeed	Get list of unread newsfeed
PUT	/api/newsfeed	Mark a newsfeed as read
POST	/api/watchtags	Setup a user's watch tag list
GET	/api/vote/:id	Get votes corresponding to an Information
POST	/api/vote	Add a vote to an Information

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

2.2. Frontend

2.2.1. Frontend Components



We followed Brad Frost's atomic design pattern[1] on composing frontend components. Since it is too complicated and tedious to list all components in atomic design hierarchy, the component diagram above shows only higher level components.

Components

1. Index (IndexButtons, IndexTable)
    - onClickQuestionWrite() : change route to Question Write page
    - onClickInformationWrite() : change route to Information Write page
    - onClickPostList(): change route to Post List page
    - onClickSearch(): chagne route to Search page
    - onClickQuestion(postId): change route to Question page which shows the question having postId.
    - onClickInformation(postId) : change route to Information page which show the information having postId.

2. SignIn
    - onClickSignIn(username, password) : try signing in, if successful, change route to Index page.
    - onClickSignUp() : change route to Sign Up page.

[1] http://bradfrost.com/blog/post/atomic-web-design/

2. SignUp
    - onClickSignUp(username, password, email) : try signing up, if successful, change route to Sign In page.

3. TitleBar
    - onClickSignIn() : change route to Sign In page.
    - onClickSignOut() : try signing out, if successful, change route to Index page.

4. Question
    - onClickAnswer(content) : try writing an answer to the question, if successful, update view including answer.
    - onClickDelete() : try deleting the question, if successful, change route to Index page.

5. QuestionWrite
    - onClickWriteQuestion(title, content, due, bounty) : try writing a question, if successful, change route to Question page which shows the question written.

5. Answer
    - onClickSelect(answerId) : try selecting the answer, if successful, update view with selected answer.

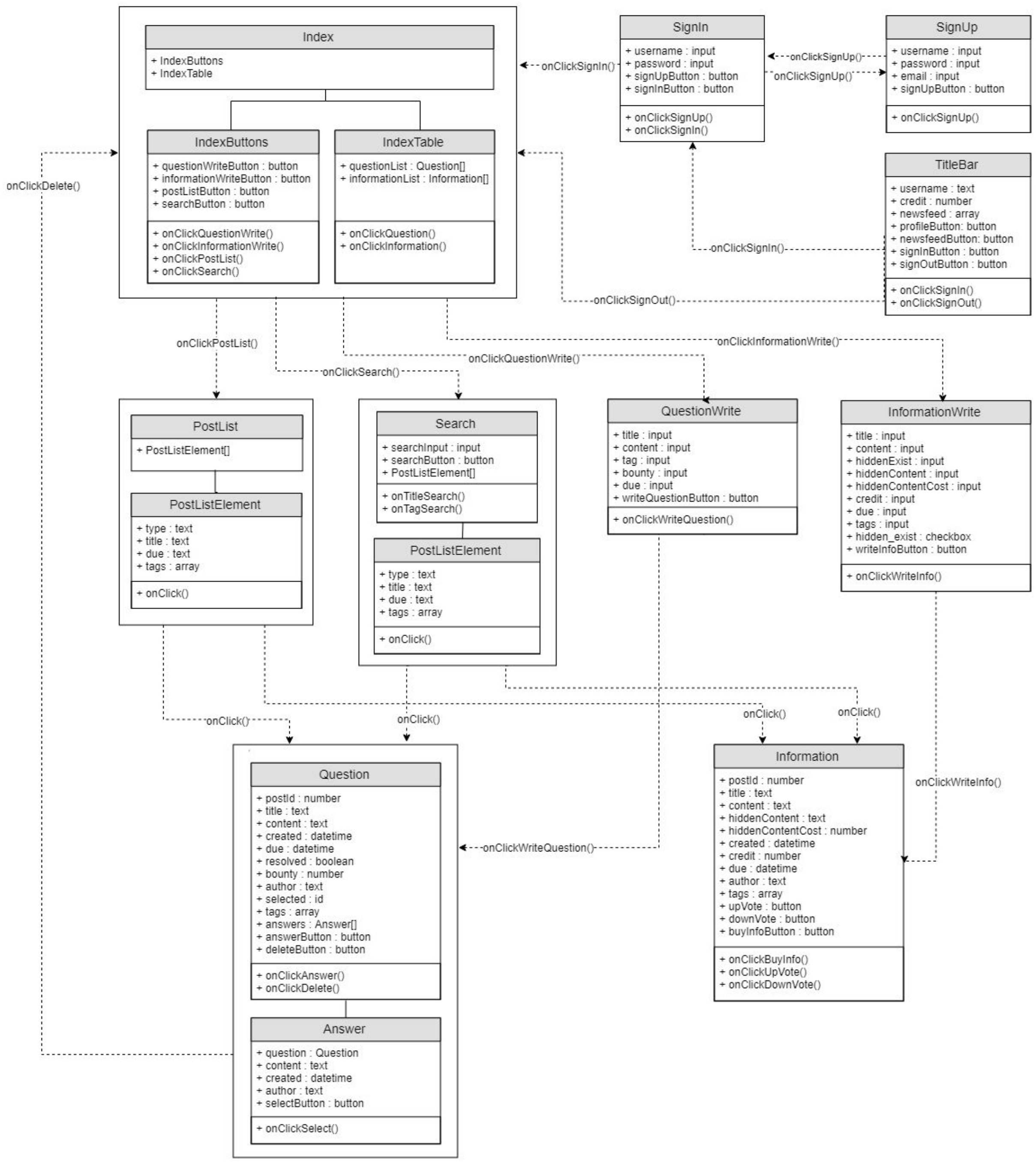
6. Information
    - onClickBuyInfo(postId) : try revealing the hidden content of the information, if successful, update view including hidden content.
    - onClickUpVote(postid) : try upvoting an information, if successful, update view with increased up vote count.
    - onClickDownVote(postId) : try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

6. InformationWrite
    - onClickWriteInfo(title, content, hiddenExist, hiddenContent, hiddenContentCost, due, credit) : try writing an information, if successful, change route to Information page which shows the information written.

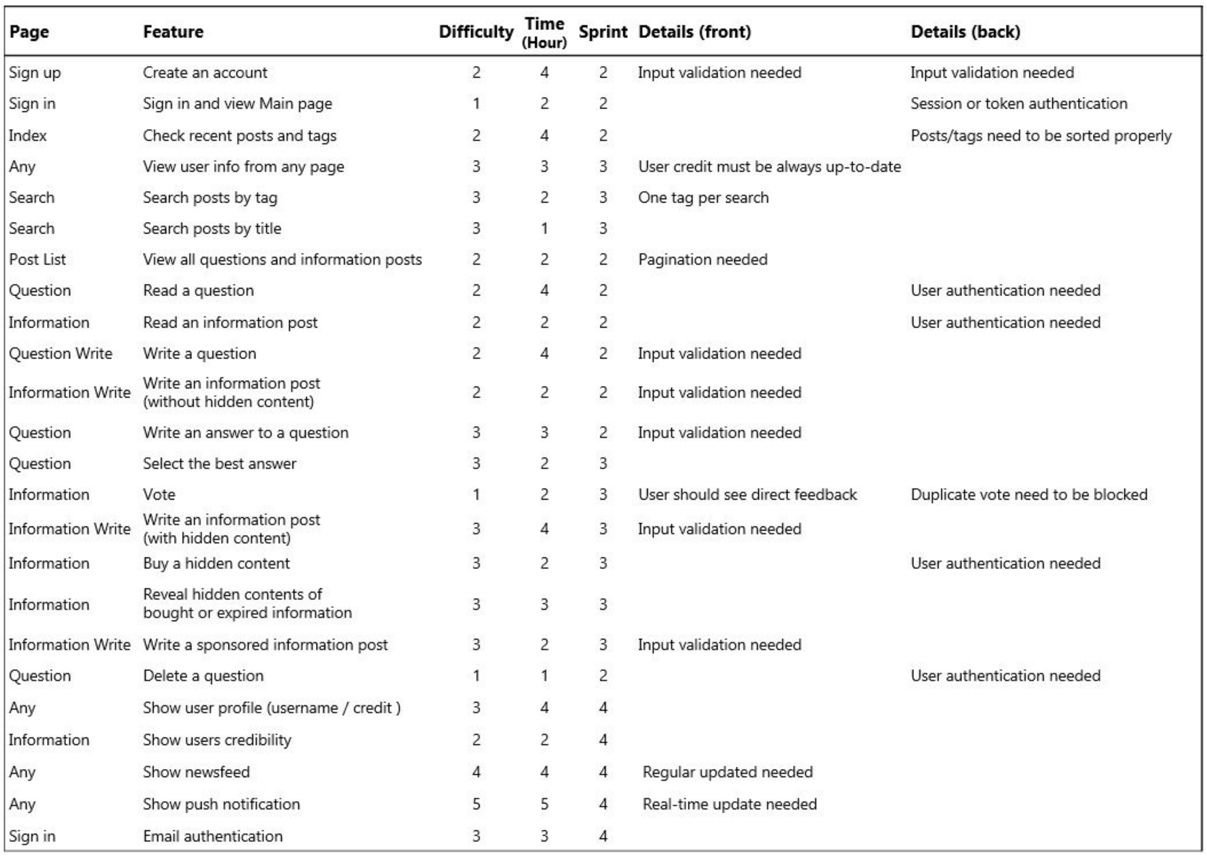
10. PostList (PostListElement)
    - onClick(type, postId) : change route to Question or Information page according to type and postId.

11. Search
    - onTitleSearch(query) : try searching posts which contain query in its title, if successful, update view with search results.
    - onTagSearch(query) : try searching posts which contain query in its tags, if successful, update view with search results.

2.2.2. Frontend Components Relationships



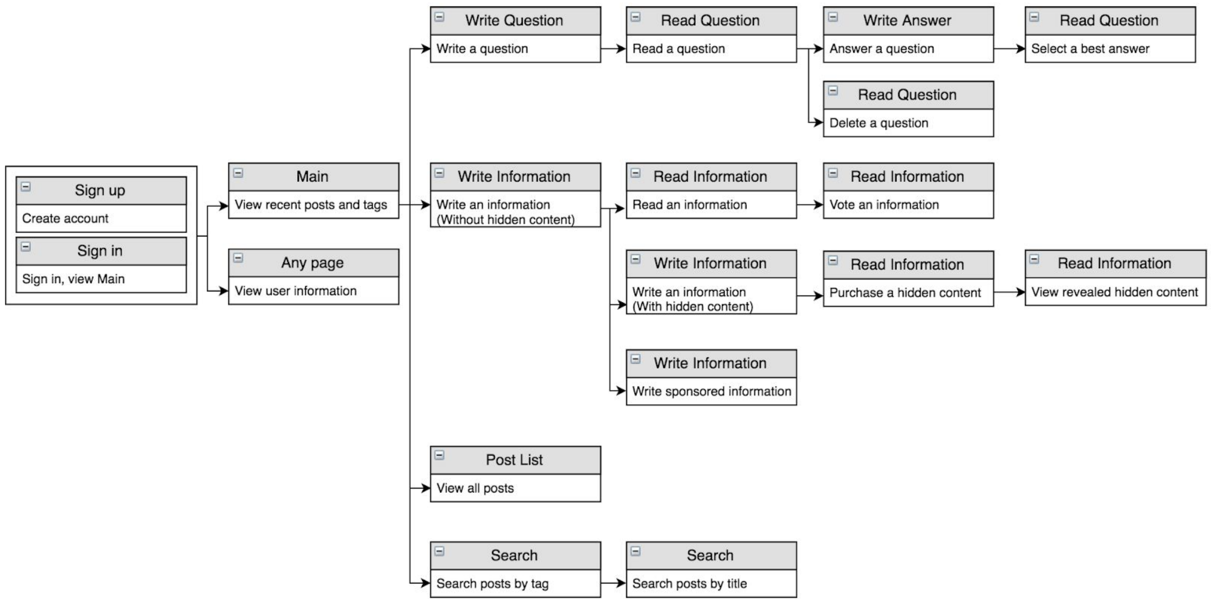
3. Implementation Plan



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
Index	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	
Question	Read a question	2	4	2		User authentication needed
Information	Read an information post	2	2	2		User authentication needed
Question Write	Write a question	2	4	2	Input validation needed	
Information Write	Write an information post (without hidden content)	2	2	2	Input validation needed	
Question	Write an answer to a question	3	3	2	Input validation needed	
Question	Select the best answer	3	2	3		
Information	Vote	1	2	3	User should see direct feedback	Duplicate vote need to be blocked
Information Write	Write an information post (with hidden content)	3	4	3	Input validation needed	
Information	Buy a hidden content	3	2	3		User authentication needed
Information	Reveal hidden contents of bought or expired information	3	3	3		
Information Write	Write a sponsored information post	3	2	3	Input validation needed	
Question	Delete a question	1	1	2		User authentication needed
Any	Show user profile (username / credit)	3	4	4		
Information	Show users credibility	2	2	4		
Any	Show newsfeed	4	4	4	Regular updated needed	
Any	Show push notification	5	5	4	Real-time update needed	
Sign in	Email authentication	3	3	4		

The table above shows the total implementation plan of this project. 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. In sprint 4, we will implement additional features which are slightly more challenging.

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.

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.

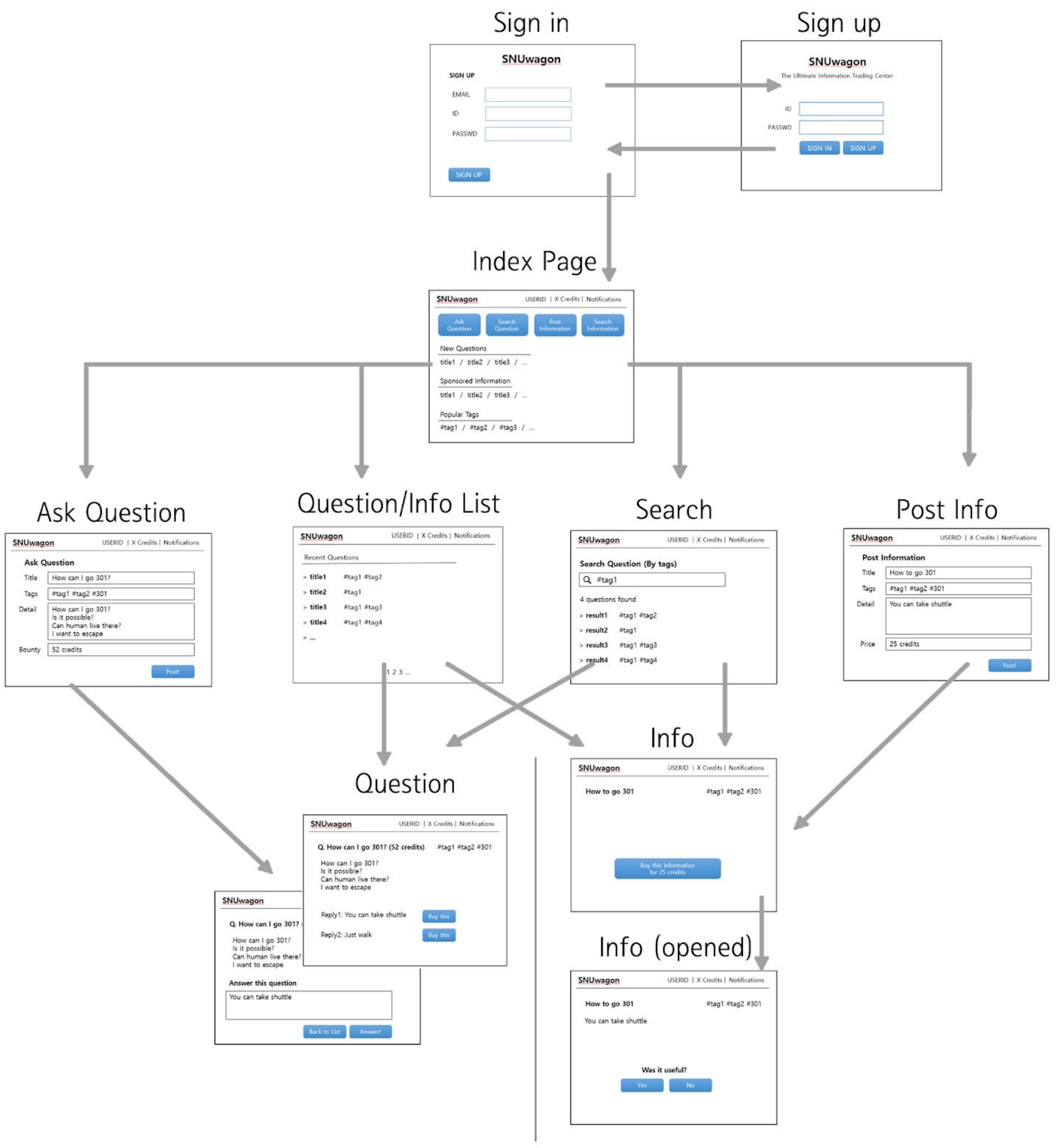
4.3. Acceptance & Integration Test

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


Sign in
SNUwagon
SIGN UP
EMAIL
ID
PASSWD
SIGN UP
Sign up
SNUwagon
The Ultimate Information Trading Center
ID
PASSWD
SIGN IN
SIGN UP
Index Page
SNUwagon
USERID | X Credits | Notifications
Ask Question
Search Question
Post Information
Search Information
New Questions
title1 / title2 / title3 / ...
Sponsored Information
title1 / title2 / title3 / ...
Popular Tags
#tag1 / #tag2 / #tag3 / ...
Ask Question
Ask Question
Title
How can I go 301?
Tags
#tag1 #tag2 #301
Detail
How can I go 301? Is it possible? Can human live there? I want to escape
Bounty
52 credits
Post!
Question/Info List
Recent Questions
title1 #tag1 #tag2
title2 #tag1
title3 #tag1 #tag3
title4 #tag1 #tag4
...
1 2 3 ...
Search
Search Question (By tags)
#tag1
4 questions found
result1 #tag1 #tag2
result2 #tag1
result3 #tag1 #tag3
result4 #tag1 #tag4
Post Info
Post Information
Title
How to go 301
Tags
#tag1 #tag2 #301
Detail
You can take shuttle
Price
25 credits
Post!
Question
Q. How can I go 301? (52 credits)
#tag1 #tag2 #301
How can I go 301? Is it possible? Can human live there? I want to escape
Reply1: You can take shuttle
Buy this
Reply2: Just walk
Buy this
Answer this question
You can take shuttle
Back to List
Answer!
Info
How to go 301
#tag1 #tag2 #301
Buy this information for 25 credits
Info (opened)
How to go 301
#tag1 #tag2 #301
You can take shuttle
Was it useful?
Yes
No