

SW Engineering CSC648/848 Fall 2018

Project Title: Aquirium

Team Number: 08

Names of Students (Local Team):

Jianfei Zhao (jzhao11@mail.sfsu.edu) (Team Lead)

Feras Alazzeh (Back-end Lead)

Tommy Lik (Front-end Lead)

Lileana Wright

Edward Barajas

Alex Li

Jiawei Xu

Milestone: 2

Date: 10/16/2018

History Table:

Date	Revision
10/16/2018	Created as the 1st version

I. Data Definitions V2

1). User

- Users are of key importance in our project. The users will utilize our web application to purchase items they need, or post items they want to sell to others. Basically, there are 3 types of users, including unregistered, registered, and admin users, with different functions for them to take advantage of. Below are the desired components for each user.
- *id* (unique numeric value for each user)
- *username* (unique username in string format)
- *password* (encrypted for safety)
- *email* (email address ended with “@mail.sfsu.edu” to be validated as an sfsu user)
- *priority* (permission level: 0, registered user; 1, admin user)
- *avatar* (image of avatar)
- *created_at* (time that marks the creation of this user)
- *updated_at* (time that marks the last update)

2). Priority

- It describes the permission level for users. Among the 3 types of users mentioned above, unregistered users are of the lowest level, since they can only browse and search items on our website. Registered users have all the privileges that unregistered users have, plus the functions of contacting sellers, posting items, etc. Admin users have the highest priority, since they can delete illegal items and also delete users with inappropriate behavior.

3). Item

- Items can be considered as goods or products, referring to the objects which users will post or purchase on our website. Each item has a title, together with a short paragraph of description to introduce related parameters or details of the item. Price will be used for sorting the items in both ascending and descending order. Below are the desired components for each item.
- *id* (unique numeric value for each item)
- *user_id* (user who post/sell this item)
- *category_id* (category to which the item belongs)
- *title* (title of the item posted on website)
- *description* (short paragraph of introduction and related details for the item)
- *price* (numeric value, also used for sorting)
- *title_img* (title picture/image to show the item)
- *deteail_img* (detail picture/image to show the item)
- *created_at* (time that marks the creation of this item)
- *updated_at* (time that marks the last update)

4). **Category**

- There are multiple categories used to classify all the items into specific types. For example, sofas, tables, and chairs are taken into the “Furniture” category, whereas calculators, laptops belong to the “Electronic Device” category. Users may browse the list of all items at first and then narrow down the results by using different categories. Below are the desired components for each category.
- *id* (unique numeric value for each category)
- *title* (title/tag of the category)

5). **Img**

- Img refers to images and pictures, which are used as supportive media in our website. When a user is browsing *Aquirium*, s/he will see a list of items. In order to provide users with intuitive impression of the items on sale, a title image will be shown for each item. If a user is interested in purchasing a specific item, s/he may click the title image. Then the user will be redirected to the detail web page, with several other images to display the details of that item.

6). **Status**

- It describes the status for post request, to be more precise, the status notifying whether the posted item is approved by admin. For each post request created by a registered user, it will not be shown on the website immediately. Instead, the status will be marked as “pending” automatically. The “pending” status will then become “approved” and go live only when an admin user examines the details of that post request and approves it. If not approved due to some inappropriate information, that request will be deleted by the admin user.

7). **Message**

- Due to safety reasons, email systems are not implemented in our web application. Instead, in-site messaging system is designed for users to communicate with each other. For example, if a user is interested in an item, s/he may ask the seller for more details before the purchase. Registered users can send messages to the sellers via in-site web form, with the question they want to ask. After receiving messages from inquiring buyers, the sellers can also reply to them through in-site messaging.
- *id* (unique numeric value for each message)
- *item_id* (item to which the message is related)
- *from_user_id* (user who sends the message)
- *to_user_id* (user who receives the message)
- *content* (content of the message)

- *parent_id* (parent message to which the current message replies)
- *created_at* (time that marks the creation of this message)
- *updated_at* (time that marks the last update)

8). Dashboard

- It serves as the panel for user to manage posted items as well as messages, and is only visible after signing in. For each registered user, the dashboard should contain the history record of messages s/he has sent or received, and items s/he has posted, where the user can edit or delete items. For each admin user, the dashboard should contain the history record of all registered users, messages, and items. The admin user can delete any item with illegal information or any user with inappropriate behavior.

II. Functional Requirements V2

i). Group by User Type

The initial functional requirements have been listed in Milestone 1, and grouped by user types, including unregistered, registered, and admin user.

- The requirements for unregistered users are 1) ~ 6).
- The requirements for registered users are 7) ~ 11), plus 1) ~ 6).
- The requirements for admin users are 12) ~ 15), plus 1) ~ 6).

ii). Group by Development Priority

On the ground of Milestone 1, the functional requirements are explained further in Milestone 2, with more details expanded. To avoid confusion, the reference numbers for each requirement are kept the same as in Milestone 1.

Every single requirements are prioritized in three levels, namely **p1**, **p2**, and **p3**. The notion of three priorities are defined as: **p1 - necessary**; **p2 - desired**; **p3 - opportunistic**. Below are the requirements grouped by development priorities. The design of such groups are made after considering both the need from users and the resources of our team, including technical supports as well as delivery schedules.

Functional requirements with priority **p1**:

- 1). Unregistered users shall be able to browse the items on the website without being registered.

- 1.1 When browsing only, the user shall not be redirected to the web page for either sign-up or sign-in.
- 1.2 The sign-in or sign-up function shall be provided at the top-right corner, as a back-up choice for old users to log back in his/her account.

2). Unregistered users shall be able to search the items according to their categories, such as furniture, clothes, electronic device, etc.

- 2.1 The category tags shall be available as a drop-down menu in the top navigation area, next to the search bar.
- 2.2 The category tags shall also be available as a list at the left side of the web page, for users' convenience.
- 2.3 Every time by choosing a different category tag, the items on the web page shall be narrowed down to the ones belonging to that specific category.

4). Unregistered users shall be able to sort the list of items by price, in both ascending and descending orders.

- 4.1 The sorting of price shall be available as a drop-down menu, with the choices of both descending and ascending order.

5). Unregistered users shall be asked for registration only when they decide to contact the seller or post an item.

- 5.1 In such cases, the user shall be redirected to the web page for either sign-up or sign-in.
- 5.2 There shall be two options, either sign-in for old registered users or sign-up for new unregistered users.

6). During registration, the checkboxes for terms and conditions shall be unchecked by default. The registration form shall be successfully submitted only when all these boxes are checked and all the required fields (username, password, etc.) are filled in.

- 6.1 All the fields necessary for registration shall be marked with “* required”, as a prompt information for user.
- 6.2 From the perspective of html, the necessary fields shall also be labelled with the keyword “required”.
- 6.3 There shall be two text fields about password, with the first one simply for input, and the second one for confirmation.
- 6.4 The necessary fields shall include username, password, confirmation of password, email, and checkboxes for terms and conditions.
- 6.5 If a registered user fails during the sign-in, there shall be prompt information to notify what the error is.

- 6.6 The hint shall be “invalid username” (or something similar) for non-existing username, or “invalid password” for wrong password.

7). Registered users shall be able to contact sellers as well as post items.

- 7.1 For each item in the list page, or the item in its detail page, there shall be a button or link named “contact seller”.
- 7.2 There shall be a button or link visible to each user at the top navigation area, named “post” or “sell”.
- 7.3 When the user clicks “post”, s/he shall be redirected to a page to fill in the details about the item s/he wants to sell.

8). Registered users shall be able to access the dashboard with the list of items they have posted.

- 8.1 In the seller dashboard, there shall be a tab named “items”, with a list of items posted by that user.
- 8.2 Right after posting an item, the status of that post shall be automatically marked as “pending”.
- 8.3 Once an item has been approved by an admin user, the status of that post shall be marked as “approved”.
- 8.4 An item shall be available to all users on the website only when its status is “approved”.

9). Registered users shall have the privilege to edit the items they have posted, but Admin users shall not.

- 9.1 In the seller dashboard, there shall be a column in the “items” tab, with the operation named “edit/delete”.
- 9.2 The “edit” operation in “items” tab shall only be visible to registered users, but not to admin users.
- 9.3 By clicking “edit”, the user shall be redirected to the detail page of that item, with database fields listed.
- 9.4 Almost all the fields (e.g., *title*, *price*, *description*, etc.) shall be able to edit, except for the status.
- 9.5 By clicking “delete”, the user shall be able to remove that posted item from the dashboard and database.

10). Registered users shall be able to send messages to sellers via web form (in-site messaging), but not via email.

- 10.1 When the user clicks “contact seller”, a web form shall be popped up, with a text field for the user to fill in as content.

- 10.2 The seller who will receive this message shall be automatically shown to the sender, with no need for typing.
- 10.3 Each in-site message shall be successfully submitted only when the content is not empty.

11). Registered users shall be able to receive messages regarding any postings from inquiring buyers.

- 11.1 In the seller dashboard for each registered user, there shall be a tab named “messages”, with a list of messages sent to that user.
- 11.2 There shall be a column in the “messages” tab, with the operation named “view”.
- 11.3 By clicking “view”, the user shall be redirected to the detail page of that message, and shown with the content.
- 11.4 There shall be an empty text field below the content, for the user to reply to the original message.

12). Admin users shall have the privilege to view and delete inappropriate items.

- 12.1 In the admin dashboard, there shall be a tab named “items”, with a list of all the items in database.
- 12.2 There shall be a column in “items” tab, with the operation named “view/delete”.
- 12.3 The “view” operation in the “items” tab shall only be visible to admin users, but not to registered users.

13). Admin users shall have the privilege to view and delete users who post inappropriate items.

- 13.1 In the admin dashboard, there shall be a tab named “users”, with a list of all the users in database.
- 13.2 There shall be a column in “users” tab, with the operation named “view/delete”.
- 13.3 By clicking “view”, the admin shall be redirected to the detail page of that user, with the details listed.
- 13.4 By clicking “delete”, the admin shall be able to remove that user from the dashboard and database.

14). Admin users shall have the privilege to approve legal items before they are shown on the website.

- 14.1 By clicking “view”, the admin shall be redirected to the detail page of that item, with the details listed.
- 14.2 Almost all the fields shall not be able to edit, except for the status, where the options are “pending”, and “approved”.
- 14.3 By clicking “delete”, the admin shall be able to remove that item from the dashboard and database.

Functional requirements with priority **p2**:

15). Admin users shall have the privilege to add/edit search categories.

- 15.1 In the admin dashboard, there shall be an additional tab named “categories”, with a list of all the categories in it.
- 15.2 In the category list, the admin user shall be able to edit the name of each category, or add a new category.
- 15.3 Once a new category is added or an old category is edited, the change shall be reflected on the website immediately.

Functional requirements with priority **p3**:

3). Unregistered users shall be able to search the items by using text search bar. If there is matching, the number of results shall also be shown together.

- 3.1 The search bar shall be available as a text field in the top navigation area, next to the category tags.
- 3.2 If no records match the text search, a sorry message shall be delivered to the user, together with some newly posted items.

4). Unregistered users shall be able to sort the list of items by price, in both ascending and descending orders.

- 4.2 Another order for items shall be descending order of creation time, to make sure that newly posted items get a chance to be shown on the website.
- 4.3 The sorting shall be based on single keyword, namely either *created_at* or *price*, due to the schedule reason.

III. UI Mockups and Storyboards

1). Browsing Items

Use Case:

Jason, an unregistered user, browses the list of items on *Aquirium*. By choosing the “furniture” category, he looks through the dorm items that he wants. After browsing furniture, he searches for books relating to his courses. Jason switches into the “book” category, and types “Computer Science” in the search bar, and finds an interesting book about computer. He clicks the title image of that book, and the detail page is shown in a new tab.

Aquarium

filters

item 1	item name price location sold by:	item 1	item name price location sold by:
item 2	item name price location sold by:	item 2	item name price location sold by:
item 3	item name price location sold by:	item 3	item name price location sold by:
item 4	item name price location sold by:	item 4	item name price location sold by:
item 5	item name price location sold by:	item 5	item name price location sold by:

Aquarium

IMAGE

title of item

price
location
sold by:
posted date

extra item description

- User can browse the list of items in home page. In prototype, there will be 3 items in each row to occupy the web page space.
- By clicking the title image in list page, user will be redirected to the detail page, through which s/he can contact the seller.

Sign In to Continue

Email Address

Password

Create Your Account

Email Address

Username

Password

Confirm Password

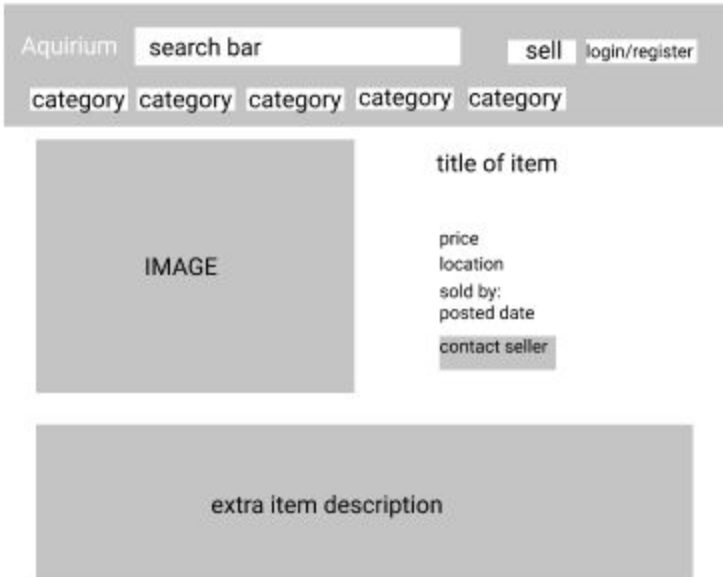
☐ Agree to terms and conditions

- When an unregistered user tries to contact the seller, s/he will be prompted to either register a new account or log in with a registered account.

2). Purchasing Items

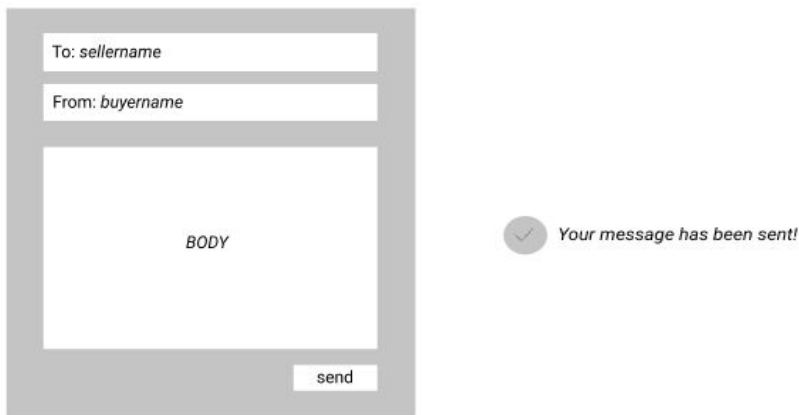
Use Case:

Jason views the detail page of a book about computer science. While reading the description of the book content, he thinks that this is exactly what he needs. Thus, Jason decides to contact the seller for more details and he is prompted to register. After that, he is able to send an in-site message to the seller. Then he is notified with a confirmation that the message has been sent.



The image shows a wireframe for an item detail page. At the top is a navigation bar with the text 'Aquarium' on the left, a 'search bar' in the center, and 'sell' and 'login/register' buttons on the right. Below the navigation bar is a row of five 'category' labels. The main content area is divided into two columns. The left column contains a large rectangular placeholder labeled 'IMAGE'. The right column contains the following text elements: 'title of item', 'price', 'location', 'sold by:', 'posted date', and a 'contact seller' button. Below these two columns is a wide rectangular placeholder labeled 'extra item description'.

- Users can view the detail page of an item before contact the seller.



The image shows a wireframe for a message sending interface and a confirmation message. On the left is a form with three main sections: a 'To: sellername' field, a 'From: buyername' field, and a large 'BODY' text area. A 'send' button is located at the bottom right of the form. To the right of the form is a confirmation message consisting of a circular icon with a checkmark and the text 'Your message has been sent!'.

- Registered users can send in-site messages to the seller. If the message has been successfully sent, it will show a confirmation message. For unregistered users, they will be prompted to register or login, as mentioned above.

3). Posting Items

Use Case:

Adam will graduate at the end of the semester. He has several pieces of furniture that are too large to bring with, including a table, a queen-size mattress, and two chairs. Moreover, he has books that are no longer needed. Thus, Adam logs in his *Aquirium* account and posts these items to sell them. After filling in the details of his mattress and clicking “post”, he is prompted with a confirmation message.

The screenshot shows the 'sell' form in the Aquirium application. At the top, there is a navigation bar with the 'Aquirium' logo, a search bar, and links for 'sell' and 'login/register'. Below the navigation bar, there are five category selection buttons, each labeled 'category'. The form fields include: 'item name' with a text input field; 'price' with a text input field; 'upload image' with a 'choose file' button; 'category' with a dropdown menu; and 'description' with a large text area. At the bottom of the form, there are 'cancel' and 'post' buttons.

- By clicking “sell” in the top navigation bar, user will be redirected to a detail page to post new item, where s/he will fill out all the required information and upload images (a title image at least, as well as optional detail images).

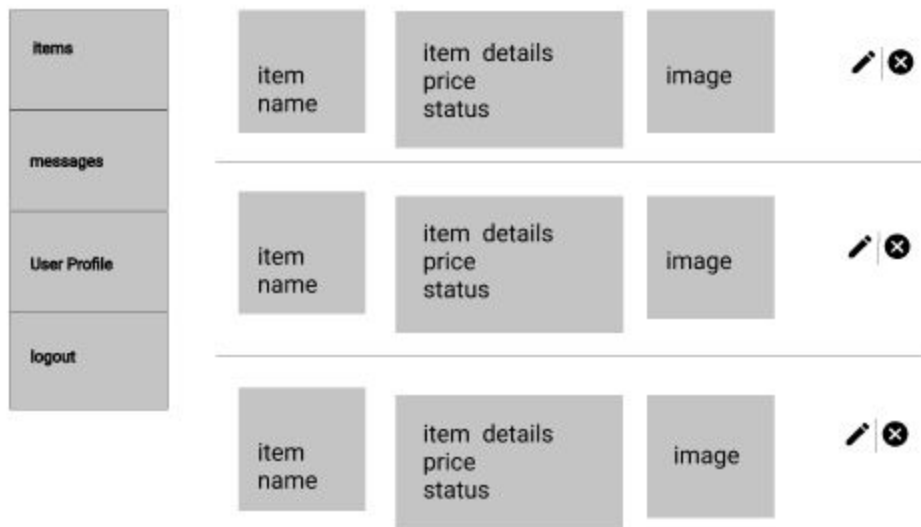


- After the new item has been submitted, the seller will be prompted with a confirmation message. Each new item will go live on the website only after it is approved by an admin.

4). Checking Dashboard

Use Case:

Adam logs back in his account 3 days later. When he checks the status of his posts in seller dashboard, he finds that they all have been approved. Besides, he receives messages from potential buyers who want to purchase his table and chairs. Having browsing other mattresses on *Aquirium*, Adam feels that the price he set for his mattress is too high for students. Therefore, he goes back to the dashboard and edits his mattress by lowering the price.



- Registered users can view the list of their own items, and also edit or delete items.



- In the dashboard, users can also check the messages they have received. Users also have a sidebar panel where they can edit their profile, and logout.

5). Monitoring Posts and Users

Use Case:

Susan is an admin user at *Aquirium* that has the permission to approve requests. She ensures that there is no inappropriate data so that the users have a good experience on *Aquirium*. One day she goes on to her admin dashboard, and notices that two new posts from sellers need to be approved. After reviewing the post requests, she feels that one of them meets our site's guideline and then approve the content to go live by editing its status. However, she finds that the other post contains illegal information, so she removes that post request and deletes that user.

items	item name	item details price status	image	 
Users				
messages	item name	item details price status	image	 
Admin Profile				
logout	item name	item details price status	image	 

- Admin user will approve appropriate posts or delete posts with illegal information.

items	user name	user details	avatar	
Users				
messages	user name	user details	avatar	
Admin Profile				
logout	user name	user details	avatar	

- Admin user can also delete any user with inappropriate behavior on the website.

IV. High level Architecture, Database Organization

i). DB Organization

Each table corresponds to a specific model in MVC pattern. Below each table, the database terms are listed in the format of

- *name_of_term* (data type, extra attribute)

The extra attribute includes some special property related to the data type. For example, “auto-increment” is only available for integer, while “utf8 unicode” is the encoding format only for string. It also notifies whether this DB term has a key attribute, such as primary key, foreign key, and index.

1). Table **User**

- *id* (int, auto-increment, primary key)
- *username* (varchar, utf8 unicode)
- *email* (varchar, utf8 unicode)
- *avatar* (varchar, utf8 unicode)
- *priority* (short int)
- *password* (varchar, utf8 unicode)
- *created_at* (timestamp)
- *updated_at* (timestamp)

2). Table **Item**

- *id* (int, auto-increment, primary key)
- *user_id* (int, foreign key)
- *category_id* (int, foreign key)
- *title* (varchar, utf8 unicode)
- *description* (text, utf8 unicode)
- *price* (int)
- *unit* (varchar, utf8 unicode)
- *title_img* (varchar, utf8 unicode)
- *detail_img0* (varchar, utf8 unicode)
- *detail_img1* (varchar, utf8 unicode)
- *detail_img2* (varchar, utf8 unicode)
- *detail_img3* (varchar, utf8 unicode)

- *status* (short int)
- *created_at* (timestamp)
- *updated_at* (timestamp)

3). Table **Category**

- *id* (int, auto-increment, primary key)
- *title* (varchar, utf8 unicode)

4). Table **Message**

- *id* (int, auto-increment, primary key)
- *item_id* (int, foreign key)
- *from_user_id* (int, foreign key)
- *to_user_id* (int, foreign key)
- *content* (text, utf8 unicode)
- *parent_id* (int, index)
- *created_at* (timestamp)
- *updated_at* (timestamp)

ii). Media Storage

We have made the decision that all the images will be stored in file system. To make the web application portable and runnable both on our local machines and on the remote server, relative path of each image file (.png, .jpg, etc.) will be saved in the MySQL database, with each file path in string format.

To avoid any duplicate filenames and unexpected overwriting to the existing image files, the back-end engineers are planning to implement our own uploading method. For example, we will design functions to generate unique name for each image file, by using the combination of timestamp and random number. Instead of choosing video, we will take image as the primary supportive media in our project. Since images generally have smaller file size than videos, we are able to reduce the waiting time for each web page.

iii). Search/Filter Architecture and Implementation

1). What will be the algorithm/SW for search?

- With the help of MySQL database provided by *phpmyadmin*, we do not have to design low-level searching algorithm by ourselves. Instead, we will directly take advantage of the SQL language such as “select * from table xxx ... where ... like ‘%...%’ and ...”.

- The *phpmyadmin* database is part of the WAMP development environment, where “M” is short for MySQL and has been successfully installed in previous Milestone.
- Both simple search and inner-joint search are used. Description about the algorithm will be further introduced (in section **IV-v**).

2). How will the search items be organized for the user?

- Normally, the search results are displayed to the user in a descending order of time, in which the time refers to the DB term *created_at*.
- In addition, the items can also be sorted by the DB term *price*, in both descending and ascending order.
- Due to the schedule reason, every time the items can be sorted by only one keyword, either in time or in price.

3). What DB terms will be searched?

- The table **Item** and **Category** in database (in section **IV-i**) will be involved in search. For table **Item**, the DB terms used for searching are *category_id*, *title*, and *description*, where *category_id* is the foreign key to connect with the primary key *id* in table **Category**. For table **Category**, the DB term *title* is only shown to users in text format, but not included in searching process.

4). How will it be coded and organized in the DB?

- The function of searching and sorting will be organized by using SQL, during which %like is used for string matching. Three of the most basic examples are provided to show SQL coding:

```
SELECT * FROM Model ORDER BY term DESC
SELECT * FROM Model WHERE cond1 AND cond2
SELECT * FROM Model WHERE id=val
```

- With the help of framework, the above coding of SQL can be translated into:

```
Model::orderBy("term", "desc")->get();
Model::where(cond1)->where(cond2)->get();
Model::where("id", val)->first();
```

In these examples, “*Model*” should exactly match the table name in database, and all the other parts following the “*::*” are functions, with parameters passing in between “()”. Herein “*where()*” and “*orderBy()*” correspond to the “WHERE” and “ORDER BY” in SQL, respectively. The function “*get()*” retrieves multiple records, and “*first()*” retrieves a single record.

iv). APIs

Below are some general application programming interface (API) that will be implemented by back-end engineers. Some general features can be explained as follows:

- The “**xxx**” ahead of each API notifies the corresponding database model, and therefore will be replaced by real model name (e.g., user, item, etc.) during the development process.
- Each API is completely in lower-case format, without any white spaces in between.
- “/” separates the name of API and the required parameter.
- “**id**” is always numeric value, to reduce the time complexity of searching algorithm within database.

Herein only high-level explanation for these APIs is provided, regardless of which specific model it refers to.

1). **xxxretrieve**

- This API is responsible for selecting multiple records from a specific table to view them as a list.

2). **xxxcreate**

- This API is responsible for saving a single record to a specific table, where the record is newly created (not existing).

3). **xxxupdate/id**

- This API is responsible for saving a single record to a specific table, where the record is updated (currently existing).

4). **xxxretrievedetail/id**

- This API is responsible for selecting a single record from a specific table to view, by using “**id**” as a parameter.
- “**id**” refers to the primary key that is necessary to retrieve the detail of this single record, and cannot be omitted.

5). **xxxcreatedetail**

- This API is responsible for redirecting to the detail page to create a new record in a specific table.
- When creating a new record, we do not need the parameter “**id**” since the primary key in every table is automatically incremented. Such maintenance is provided by MySQL database, and it guarantees the uniqueness of every single “**id**”. As a result, no more parameter passing is required.

6). **xxxupdatedetail/id**

- This API is responsible for redirecting to the detail page to update an existing record in a specific table.
- “**id**” refers to the primary key that is necessary to retrieve the detail of this single record, and cannot be omitted.

7). **xxxdelete/id**

- This API is responsible for selecting a single record from a specific table to delete, by using “**id**” as a parameter.
- “**id**” refers to the primary key that is necessary to delete this record, and cannot be omitted.

v). **Description of Non-trivial Algorithm for Searching the Items**

1). **Simple Search**

- When there are a huge amount of records in database, scanning the entire table is definitely not a smart choice. Such kind of brute force algorithm would lead to $O(n)$ time complexity. Instead of searching all the records, we may take advantage of the index attribute, which is provided by MySQL database (also available in other popular database).
- With the help of tree data structure, a DB term of integer type can be indexed, where the key values are stored as a tree. When we try to search data records by using the indexed DB term, an efficient algorithm similar to binary search will be applied. As a result, the time complexity can be reduced to $O(\log n)$. It is noteworthy that all the primary keys are automatically indexed.

2). **Inner-Joint Search**

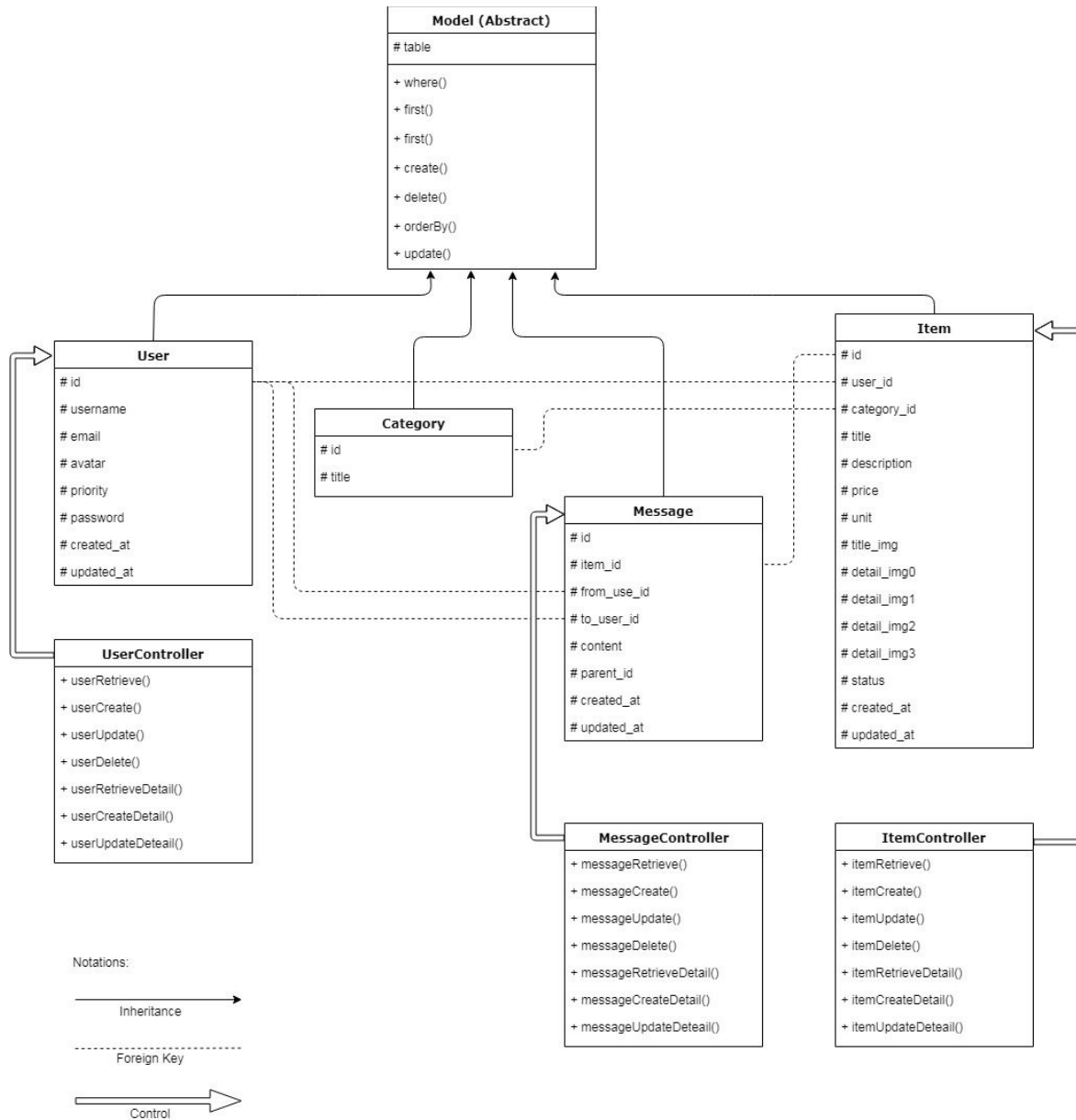
- By using foreign keys, we may connect a table to other tables. In this way, we can combine the information from multiple (usually 2) tables together as a whole, without any nested loop algorithm.
- Another advantage is that the foreign keys are integer-type DB terms, and we always prefer integer-matching to string-matching.

vi). **SW Tools and Frameworks**

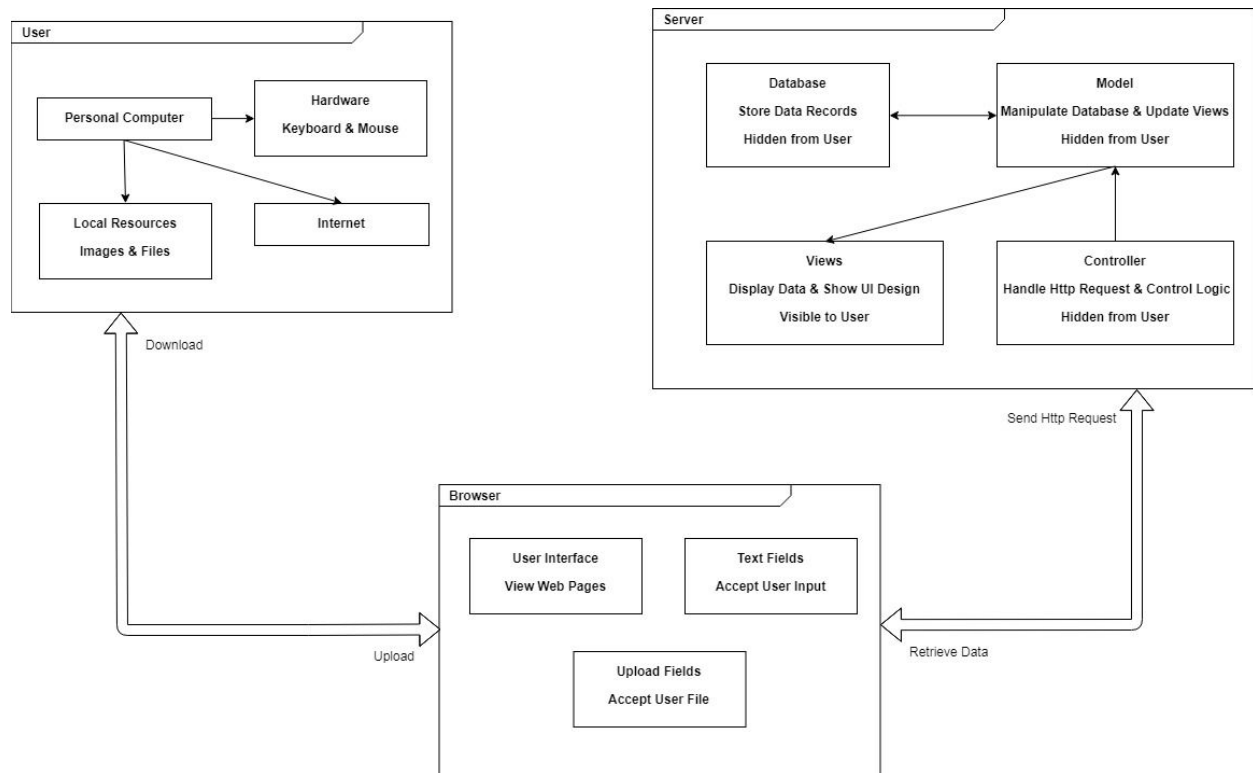
Currently, we do not have any changes about the tools and frameworks that we are using. If we do need new tools to help with development, we will send an email to Anthony (the CTO) to acquire his approval before we use it.

V. High Level UML Diagrams

1). UML Class Diagrams



2). UML Component and Deployment Diagrams



VI. Identify Actual Key Risks at This Time

- 1). Skill Risks: None
- 2). Schedule Risks: None
- 3). Technical Risks: None
- 4). Teamwork Risks: None
- 5). Legal/Content Risks: None