

User Stories

This document is intended to outline the initial user stories, website template, front-end and back-end models, and basic database concepts.

User Story: Login/Sign-Up

“As a user, I would like to sign-up to this website, and login when desired with my email and password”

The user requires a login and sign-up system. For the initial phase, this login and sign-up will be very basic in order to follow the paradigm of incremental development. The login/sign-up forms will be on the same page, where the left-hand side includes the login system of email and password, and the right-hand side will include the sign-up system of email and password. The databases will store these values, along with other user properties that will be added to the sign-up in a future user story.

During sign-up and login, the email should be validated via simple *html based validation*, no JS at this point as it is unnecessary.

In the sign-up system, once the email is validated, the password should be validated via simple character limit requirements. If both email and password are valid, the email is checked against the user-login database to see if the email already exists. If it doesn't, then a new account is added to the database, and the user is notified of this with an appropriate message. If the email is already taken, then the user is notified of this and nothing occurs.

For the login system, the email is checked for validity, and if it is valid, the email is used to query the database. If the email exists in the database, it's password is checked against one provided by the user. If the email and password provided by the user match an equivalent pair in the database, the user is logged in and notified of this; else the user is notified of the incorrectly provided data.

User Story: Post a Question

“As a user, I want to be able to post a question once logged in to my own personal account”

The question page should have a *title* text box with a 100-character limit, and a question *body* with a 1000-character limit. No submissions will go through unless both boxes contain content. These requirements will be refined as the project moves ahead.

If the user is not logged in, the page should display the option for the user to login/sign-up before posting. If the user is logged in, he/she will be able to send a question. Once a valid question is submitted by a logged-in user, the title and body will be sent to the server side via a *post*. Along with the title and body of the question, several other auto-generated metrics (date,time,user_id) will be appended to the entry and stored in an appropriate database. A message is returned telling the user the question was successfully submitted.

User Story: Search for Question

“As a user, I want to be able to search for a question in hopes of having it already answered”

Almost every website has a search bar for the users to look up things which interest them. In the navigation bar – which will be present in every single page – there will be a search bar.

When the user enters a search query, it is sent to the server where it is matched via a custom-made algorithm with question tiles pulled from the appropriate database. All matching results will be sent back to the client via the server and listed on a single page (many in the future) where the user may click on any entry to visit its Question Forum (where question, answers, and ability to answer resides). The initial search method will be fairly simple, but will be tweaked in the future based on user requirements.

User Story: Read the Question Forum

“As a user, I want to be able to view a question and all of its answers”

The user will be able to read the entire question (including metrics such as date and time of post), along with all answers provided by other users. There are 3 different divisions on the page. From top-to-bottom, it starts with the question (title and body), followed by the answers, and finally a text box for new answers including a button to submit. The submit button is only available for logged-in users. If the user is not logged-in, there will be a message which asks the user to sign-up or sign-up via a button which redirects the user to the log-in/sign-up page

The server needs to pull all data related to the question at hand including: the title, the body, the user id, the date and time. It also needs to pull data of all answers linked to the question as objects including the following fields: the answer, the user id of the person who answered the question, the date, time and score of the answer.

There will be a text area for the user to write new answers and a submit button. The actual submission of the answer will be covered in the next user story below.

User Story: Answer a Question

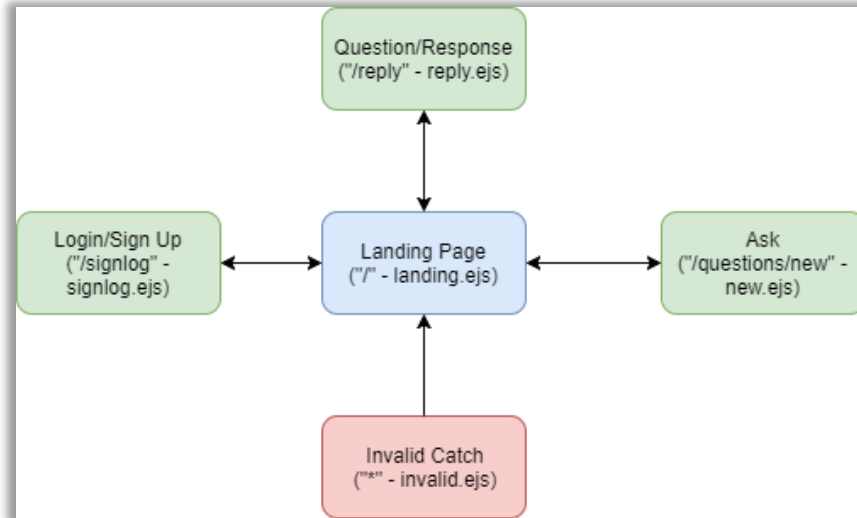
“As a user, once I find an appropriate question through a search, I want to be able to answer the question”

Each question will have its own dedicated page that lists the question title and body, and all previous answers to that question below it, and at the bottom of it all, a text area of a logged-in user to submit his/her own answer. The answer should be a maximum of 1000-characters, and an empty answer will not be submitted. If the user is not logged on, there should be an appropriate redirect button to allow the user to quickly login/signup otherwise they will not be able to submit an answer.

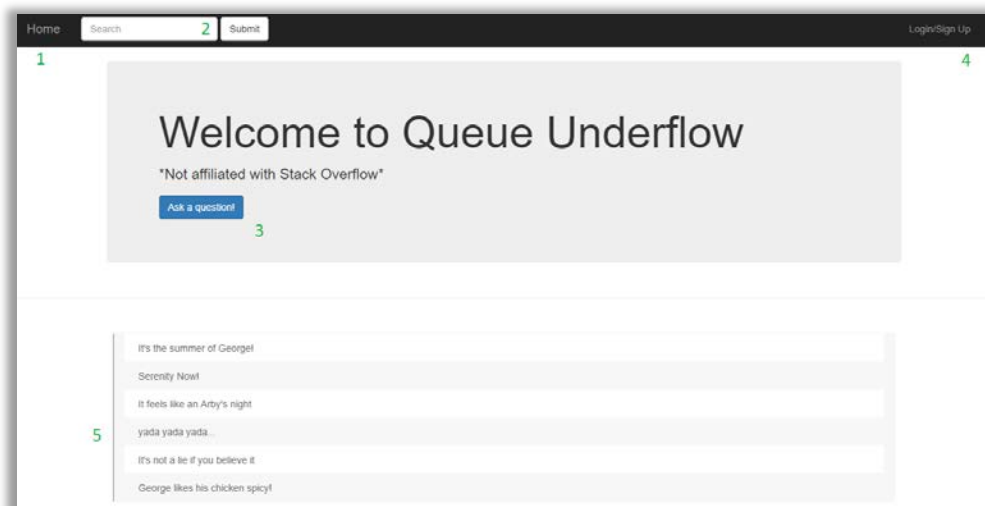
Once a valid answer is submitted, it will be stored in an appropriate server-side database with a few auto-generated fields (date, time, user_id, score...). When the data is finally stored on the server, the server returns the same page updated with the newly submitted answer.

Front End Design

The front end of the website is based around the home landing page. From there, all other pages of the site can be accessed.



Landing Page (“/”):

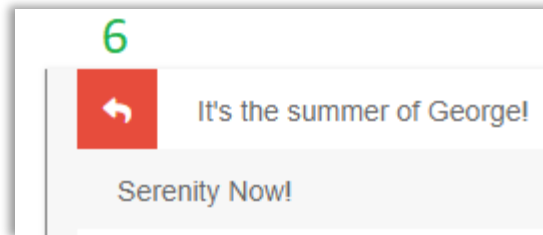


When the user accesses the home landing page he/she will be presented with a list of the most popular/newest questions that have been asked. From the landing page, the user is able to access all other pages of the site.

Referring to the image above, the landing page consists of:

- 1) Home button – Allows the user to return to the landing page from anywhere else on the site
- 2) Search button – Allows the user to search the question database for up to 100 characters
- 3) Ask button – Redirects to the Ask page (“/questions/new”) in order to ask a new question
- 4) Login/Sign Up button – Redirects to the Login/Sign Up page (“/signlog”)
- 5) A list of the titles of previously posed questions
- 6) Reply button (below) – Redirects to a page where the user can reply to a selected question

The reply button can be accessed by simply hovering the mouse over the question and clicking on the reply icon, as seen in the image below.



Login/Sign Up Page (“/signlog”):

Referring to the image above, the Login/Sign Up page consists of:

- 1) Login form
 - a. Email input – HTML validation to ensure proper email format
 - b. Password input – No validation needed here
- 2) Sign Up form
 - a. Email input – HTML validation to ensure proper email format
 - b. Password input – HTML validation to ensure it is between 6 – 12 characters

To ensure that only one action can be done at a time, the page will immediately redirect to the landing page (“/”) as soon as either form is submitted.

Ask Page (“/questions/new”):



The screenshot shows the 'Ask Page' with a dark header bar containing 'Home', a search bar, a 'Submit' button, and a 'Login/Sign Up' link. The main content area has the heading 'What is your question?'. Below this is a 'Title:' label followed by a text input box with a green '1' to its left and placeholder text 'This is where your question goes (100 character max)'. Below the title box is a 'Question:' label followed by a larger text area with a green '2' to its left and placeholder text 'This is where your question goes (1000 character max)'. At the bottom left of the form is an 'Ask!' button.

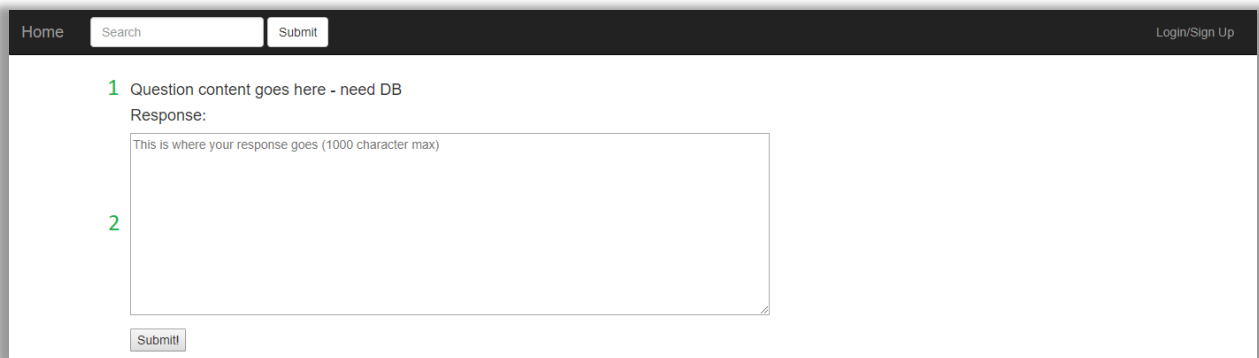
Upon entering the page, there is an immediate javascript check to ensure that the user is currently logged on. If no login is detected, the page automatically redirects to the Login/Sign Up page (“/signlog”). If the user is logged in, the javascript check grants access to the page contents.

Referring to the image above, the Ask page consists of:

- 1) Question Title text box
 - a. HTML validation to require between 1-100 characters
- 2) Question Content text box
 - a. HTML validation to require between 1-1000 characters

When the question is submitted, the user is redirected to the landing page (“/”) where the new question title will appear in the list.

Question/Response Page (“/reply”):



The screenshot shows the 'Question/Response Page' with the same dark header bar. The main content area displays '1 Question content goes here - need DB' in green. Below this is a 'Response:' label followed by a text area with a green '2' to its left and placeholder text 'This is where your response goes (1000 character max)'. At the bottom left of the form is a 'Submit!' button.

Upon selecting to reply to a question, the user is taken to the Question/Reply page. This page lists the full body of the question and allows the user to add a reply. There will be javascript validation to ensure that the option to reply is only available if the user is logged in.

Referring to the image above, the Question/Response Page consists of:

- 1) The content (title, body, and username) of the original question
 - a. Currently placeholder
- 2) Response textbox
 - a. HTML validation to require between 1-1000 characters

When a response is submitted, the page refreshed and displays the newly added response.

Invalid Page (“*”):



If the user enters a custom URL, our system will detect that the page does not exist and display an appropriate message. This page is only accessible if an error occurs and cannot be accessed from any of the links on the listed pages.

HTML/CSS Tools

The front end was created with the help the Bootstrap library.

Bootstrap:

Bootstrap allows the uses of specific class definitions to refer to a large imported stylesheet. It was used to easily create the navbar, and to ensure that all items of the page resize appropriately as the browser window changes size.

- Bootstrap v.3.37 is available at <http://getbootstrap.com/docs/3.3/>
 - Imported into our code via the CDN

Server Request/Response

Server requests and responses are handled via Express and NodeJS by using the “get” and “post” functions.

Get:

The “get” command instructs the server to listen in on a designated page and when accessed, it responds with text, a console log, or a web page to be rendered by the browser. It is also able to pass javascript variables to the associated embedded javascript file (.ejs), which allows for dynamic changes to be embedded in the web page’s HTML code.

Below is an example of the landing page “get” command, which responds to the request by rendering the landing page .ejs file and passing in an array of question titles.

```
// Sends back the landing page
app.get("/", function(req, res) {
  res.render("landing.ejs", {titles: titles});
});
```

Post:

The “post” command is used to gather data that has been entered on the webpage by the user. The command instructs the server to listen in on a specified page, and to capture any data that is submitted from that page. The data is transmitted over the server via the request (req) and can be accessed using the body-parser package, which converts the request body into a string. The “post” command can also be used to send a response whenever data is submitted, like to a redirected page, for example.

Below is an example of the user login “post” command. It receives the information filled out in the login form, parses it from the request body, and saves it to local variables. It then logs the action and redirects to the landing page.

```
// Gets the data from the login form
app.post("/login", function(req, res) {
  var email = req.body.email;
  var pass = req.body.pass;
  currentUser.email = email;
  currentUser.pass = pass;
  console.log(email + " has logged in with password: " + pass);
  res.redirect("/");
});
```

All “post” commands in the website’s code have a corresponding console log to ensure that data is properly transferred and received.

```
C:\Users\Tom\Desktop\S0EN341\Website>node app.js
Current Directory: C:\Users\Tom\Desktop\S0EN341\Website
Server Running on PORT: 3000
New Question Submitted
Test@Hi.com has logged in with password: password
New user added - email: Newbie@Hi.com, password: hellothere
```

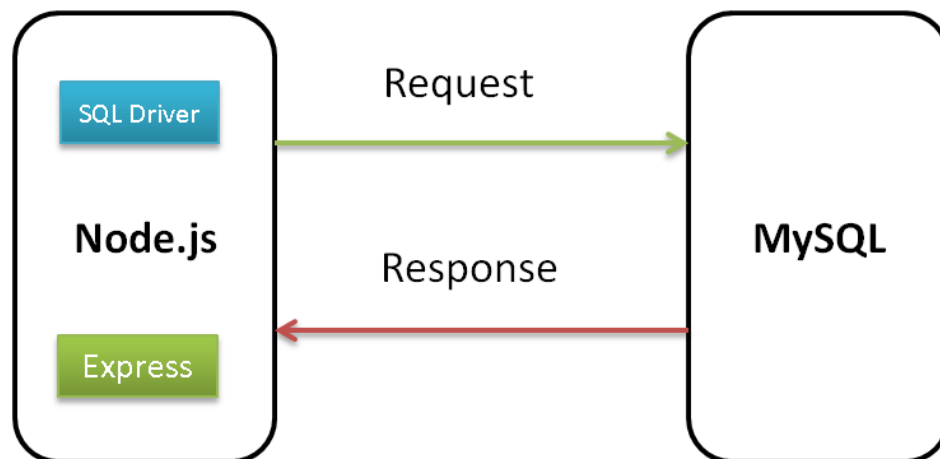

DATABASE DOCUMENTATION

A critical point in our project is to link Node.js (JavaScript) with many databases (SQL). In order to do so, we need to use some features provided by Node.js and few other tools for ergonomic and productivity purposes. Testing is crucial in this part of the project since the order of steps matters.

Installation instructions are not included in this document for simplicity.

❖ Required Tools:

- **Node.js**
- **Npm : Node package manager**
- **Text editor : Atom + packages (example : atom-runner, git-plus)**
- **Express : a node.js module and a web framework used for the server side**
- **MySQL driver : installation through node.js**
- **Nodemon : allows to automatically restart a server**
- **XAMPP : to start a server and have access to phpMyAdmin to easily create tables for the database**



➤ Specifications

1. Connection to the database :

The first requirement is to link the database to Node.js. Pre-defined methods like `createConnection()`, `connect()` and `require()` are used to reach this goal. The code implementation is standard; an example it is shown below for reference:

In our text editor (Atom), we create a new file test.js.

```
var mysql      = require('mysql');
var connection = mysql.createConnection({
  host      : 'localhost',
  user      : 'me',
  password  : 'secret',
  database  : 'my_db'
});

connection.connect();

connection.query('SELECT 1 + 1 AS solution', function
(error, results, fields) {
  if (error) throw error;
  console.log('The solution is: ', results[0].solution);
});

connection.end();
```

* The name of the database is not mentioned since it is not created yet.

➤ **Test connection**

2. Create the database and its tables :

In order to create the database, the keyword CREATE DATABASE is used. To create a table, the keyword CREATE TABLE is used.

- **Entities of the database has to be unique**
- **Whenever a table is created, a primary key column must also be created and incremented automatically.**
- **Check if the table already exists**
- **Constraints on the tables (# of characters, conditions > or < ...)**

3. Manipulate the database :

The data is inserted into a table either one record at a time or many all at once via an array. Retrieving data from the table involves selecting the desired attribute(s) (column) and conditions to attach to them in order to get an accurate return of information. Select information from multiple tables requires joins and conditions (similar to set theory logic).

- Sort tables when necessary or apply filters (advantage : start at a certain index and offset)
- Check if values can be updated correctly
- Delete values, attributes.
- Drop a table.
- Use joins in the searching algorithm
- Limit the numbers of results returned by a query of the database (also for search algorithm)

Example of Tables Created for The Project :

USER

Field	Type	Null	Key	Default	Extra
first_name	varchar(30)	NO		NULL	
last_name	varchar(30)	NO		NULL	
username	varchar(30)	YES		NULL	
email	varchar(60)	NO		NULL	
password	varchar(60)	NO		NULL	
Country	varchar(40)	YES		NULL	
birth_date	date	YES		NULL	
sex	enum('M','F','O')	YES		NULL	
date_entered	date	YES		NULL	
user_id	int(10) unsigned	NO	PRI	NULL	auto_increment

10 rows in set (0.00 sec)

QUESTION

Field	Type	Null	Key	Default	Extra
question_title	varchar(100)	NO		NULL	
question_body	varchar(1000)	NO		NULL	
user_id	int(10) unsigned	NO		NULL	
date_asked	date	YES		NULL	
question_id	int(10) unsigned	NO	PRI	NULL	auto_increment

5 rows in set (0.00 sec)