User Stories and Set-up Assignment

Brendan Byers, Ryan Sisco, Iliana J, Aidan Grimshaw, Yufei Zeng
byersbr, siscor, javieri, grimshaa, zengyu

Contents

1 User Stories

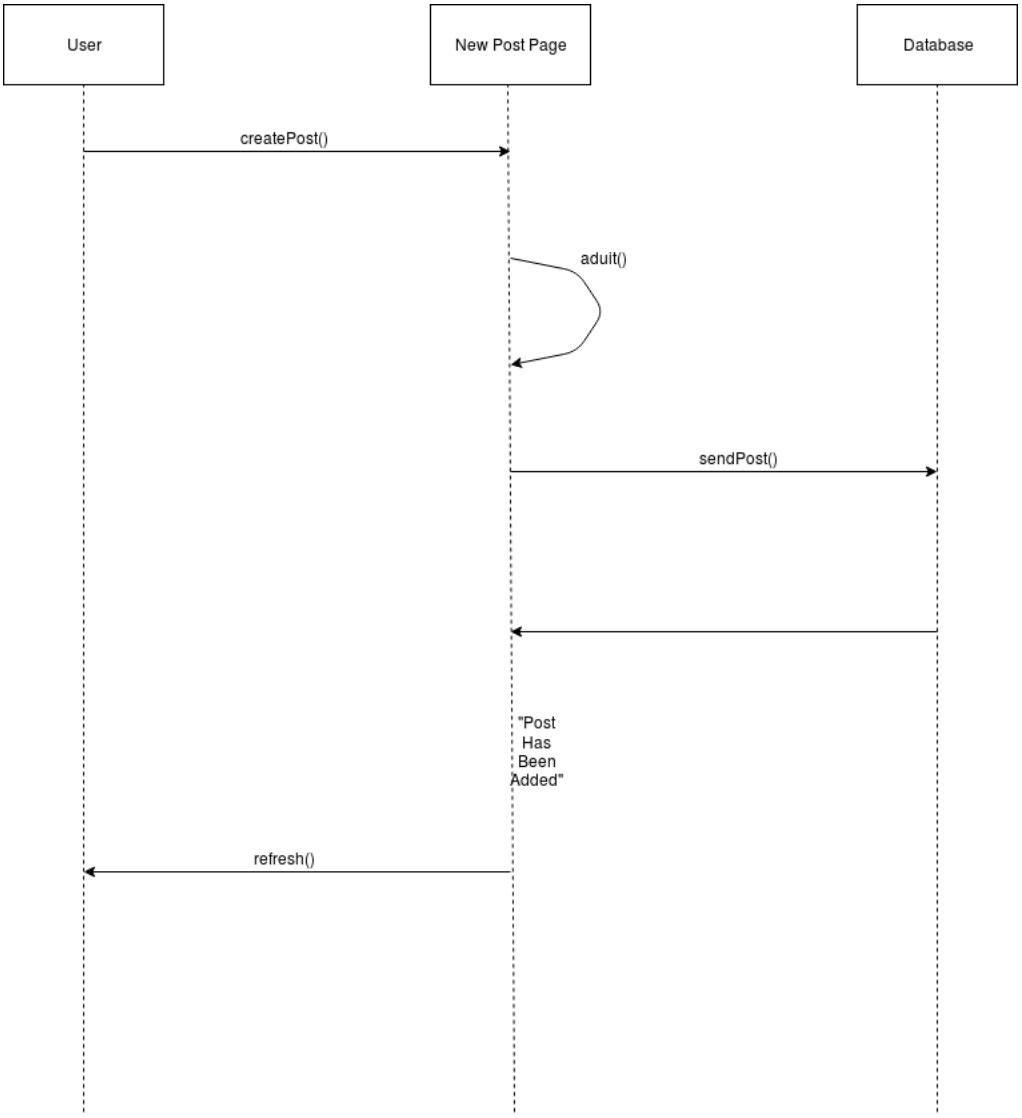
1.1 Add a post

As the user adds a post, his post body is checked for any illegal characters or length, where it is then sent to the server and published.

1.1.1 Corresponding Tasks

This user story depends on getting an initial feed working. Once the initial feed is complete, we can add the capability to add a post. It will be due early on, as many other things depend on it. Adding a post should be reasonably simple. It will consist of a post create screen where a user writes their post. When a button is clicked, the post will be verified and then sent to the server and added to the database. Other users will then be able to view and react to it. Paired programming time: 4 hours.

1.1.2 UML Sequence Diagram/Spike



1.2 Add a comment

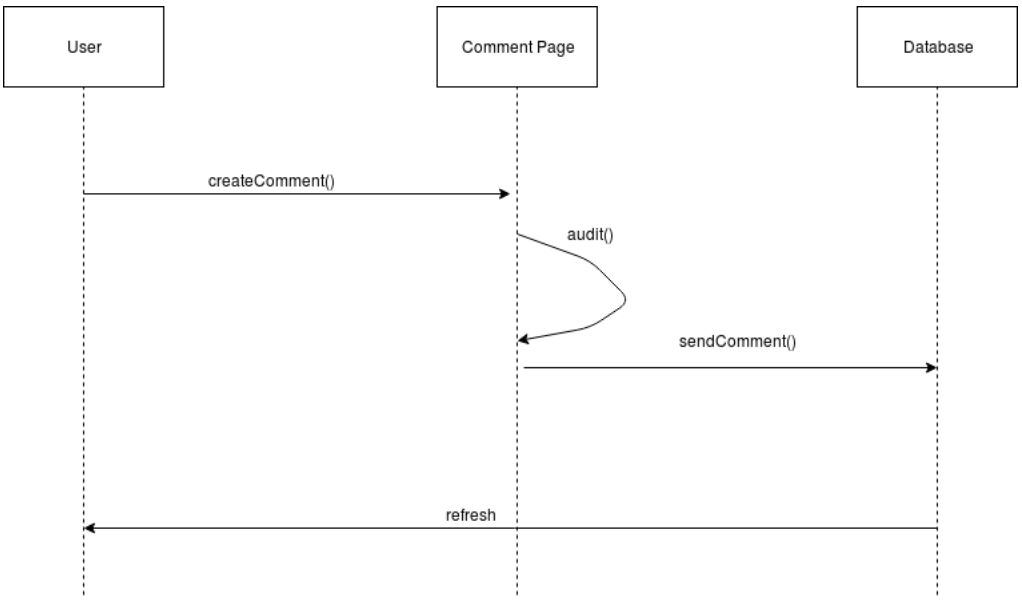
When the user is on the post page, they can add a comment where it is checked for any illegal characters or length, and then it is sent to the server and published.

- Response will be tied to the original post
- Responses arent viewable from the main feed
- Response can be voted on/shared/favorited

1.2.1 Corresponding Tasks

This depends on the ability to create a post. The process will be exactly the same, except when the comment is created it will be tied to the parent post. Comments will behave exactly the same as posts. This user story will be due after the ability to add a post is implemented. Paired programming time: 2 hours.

1.2.2 UML Sequence Diagram/Spike



1.3 Like/Dislike

When a user votes, their +1 or -1 is sent to the server, as long as they have not voted on that post already, then the server is updated.

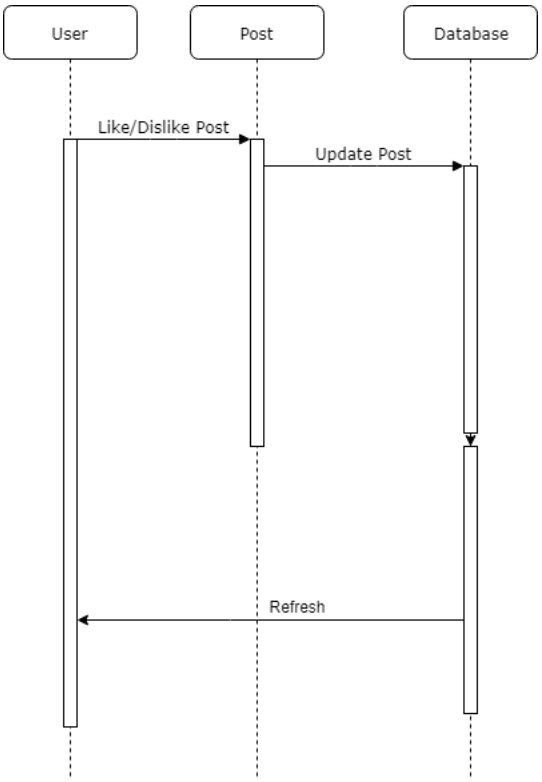
- Posts can receive one vote from the user
- The users vote can be changed
- If a post is upvoted:
  - If upvote is clicked the vote is removed
  - If downvote is clicked the vote becomes a downvote

1.3.1 Corresponding Tasks

This user story depends on the Add a Post user story. Once that is implemented we can begin work on this user story. Buttons will be present on each post that when clicked enable to user to vote on a post. The user can either like or dislike a post. If a user previously liked a post and click like again, the vote is removed. If dislike is clicked the post will be marked

as so. The total votes for a post should be stored server side. Paired programming time: 6 hours.

1.3.2 UML Sequence Diagram/Spike



1.4 Report

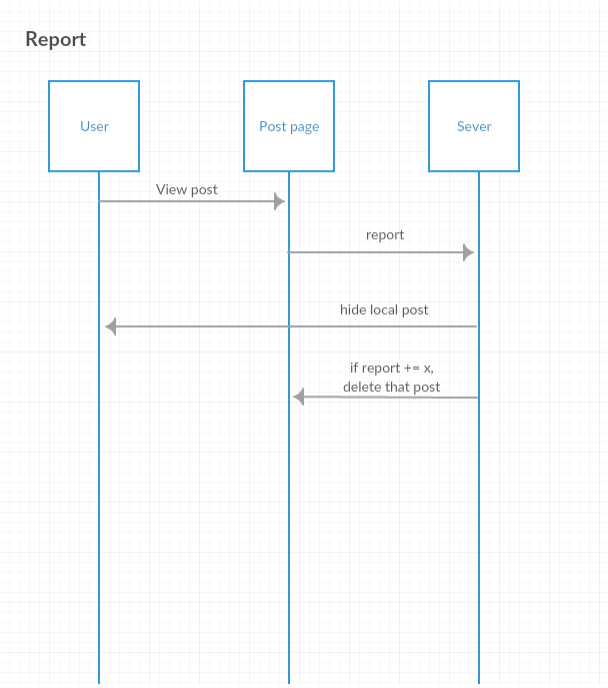
When a user clicks the report post button, the post is flagged for review. After X amount of reports, the post is automatically taken down while waiting review.

- When a user reports a post, the post wont appear on the feed for that user

1.4.1 Corresponding Tasks

This will be able to be completed once the post object has been created. Users will click a button and a window will appear. They will be able to choose or type in a reason for the reporting, then send the report. When a post is reported by the user the post will be hidden from the users view indefinitely. This depends on the functionality of the feed and refreshing posts. Paired programming time: 6 hours.

1.4.2 UML Sequence Diagram/Spike



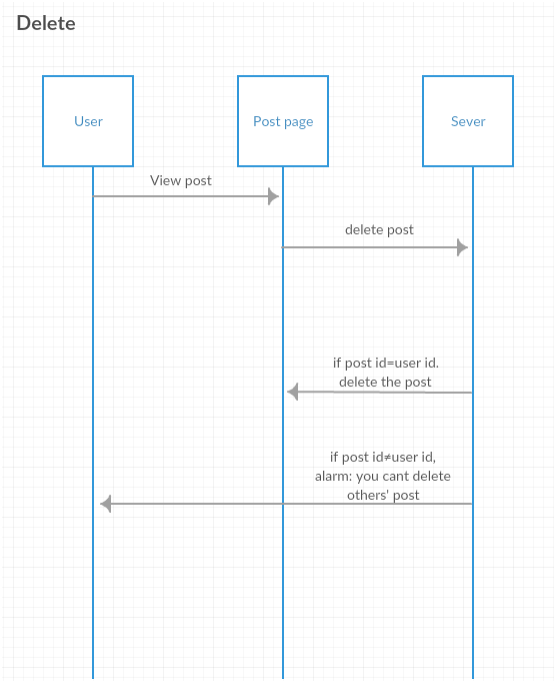
1.5 Delete

When a user wants to delete a post, it is checked if it is their own post, if it is, it is deleted. If it is not, it is checked if they are an admin, if they are, it is deleted. The server will then update based on the above.

1.5.1 Corresponding Tasks

This user story depends on the feed, posts, and refresh being implemented. A button will be present on user created posts that allows them to delete their own posts. This will allow people to delete what they posted before. When a post is deleted it will no longer appear on any devices when the users refresh things. Paired programming time: 5 hours.

1.5.2 UML Sequence Diagram/Spike



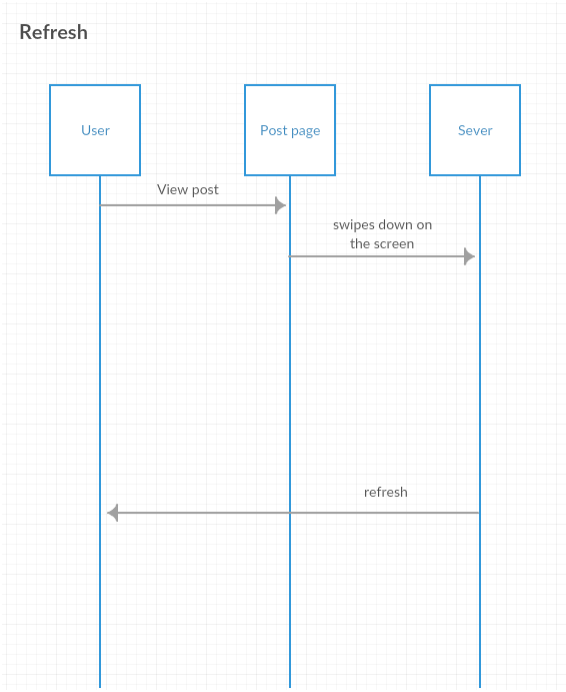
1.6 Refresh

If the user swipes down on the screen, the page reacts with an animation and then refreshes the page.

1.6.1 Corresponding Tasks

This will be due after the implementation of posts and the feed. This will refresh the feed and pull new posts from the database. It wont be a full refresh of the page. IT will simply refresh the data within the page, giving a better user experience. This will be a function of the service worker. The new content will be fetched and then served to the user. Paired programming time: 5 hours.

1.6.2 UML Sequence Diagram/Spike



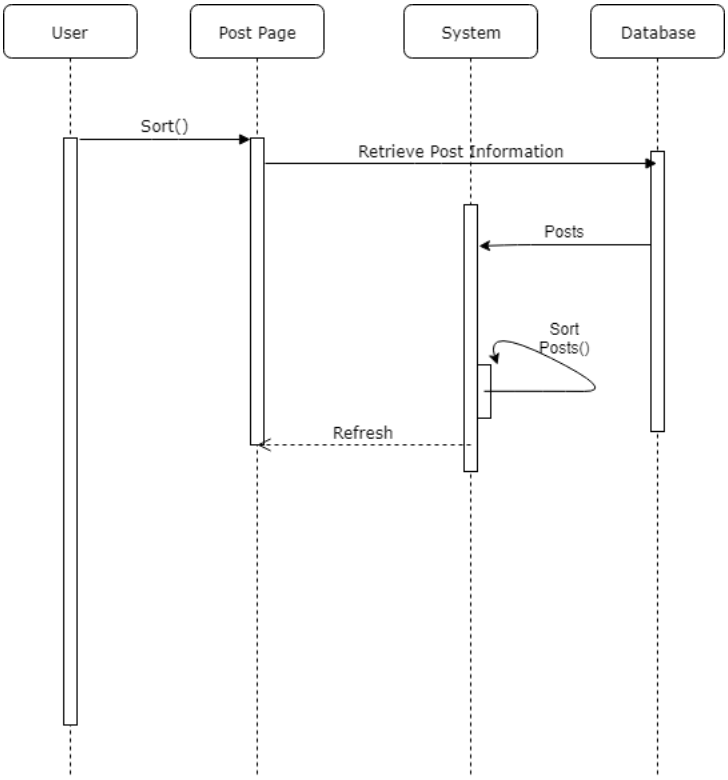
1.7 Sort

when clicked, the page alerts the user of the sorting type and then changes the sorting on the next click. They will be able to cycle through all of the different types.

1.7.1 Corresponding Tasks

Posts on the front page will be able to be sorted based on certain criteria. Users will be able to sort based on top posts, new posts, and possibly hot posts. This user story depends on the feed, post objects, and refreshing. It will be due after refresh is implemented. Paired Programming time: 3 hours.

1.7.2 UML Sequence Diagram/Spike



1.8 Add to Favorites

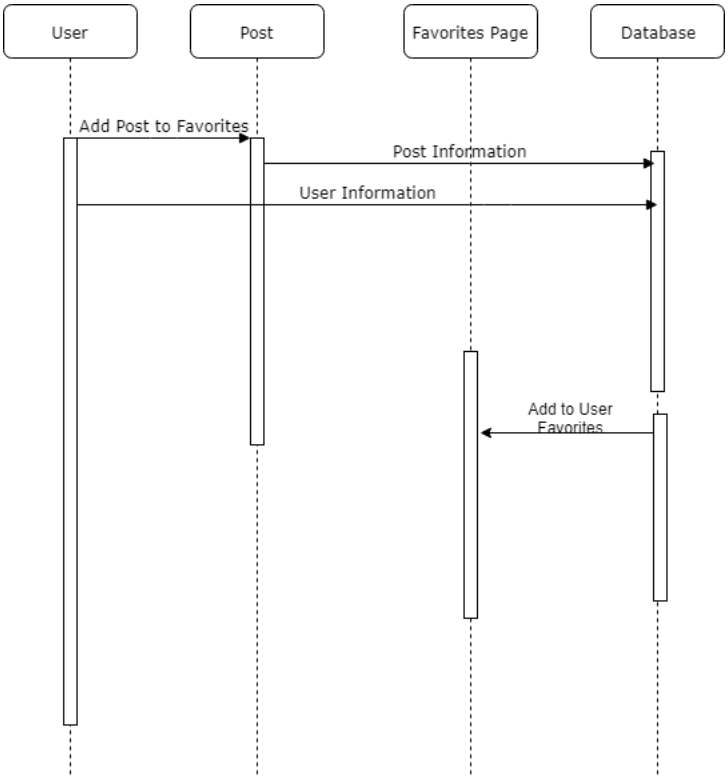
All favorite posts are shown to the user by date added only.

- After a post is added to favorited, the post on the main feed should show that it is currently in the users favorites list

1.8.1 Corresponding Tasks

This will enable a user to add a post to their favorites list. It depends on the feed, posts, and favorites. There will be a button on each post that allows the user to add that post to their favorites list. When the button is clicked, the post will be saved in storage for viewing later. The favorites list will not be stored server side. Paired Programming time: 5 hours.

1.8.2 UML Sequence Diagram/Spike



1.9 Offline Favorites

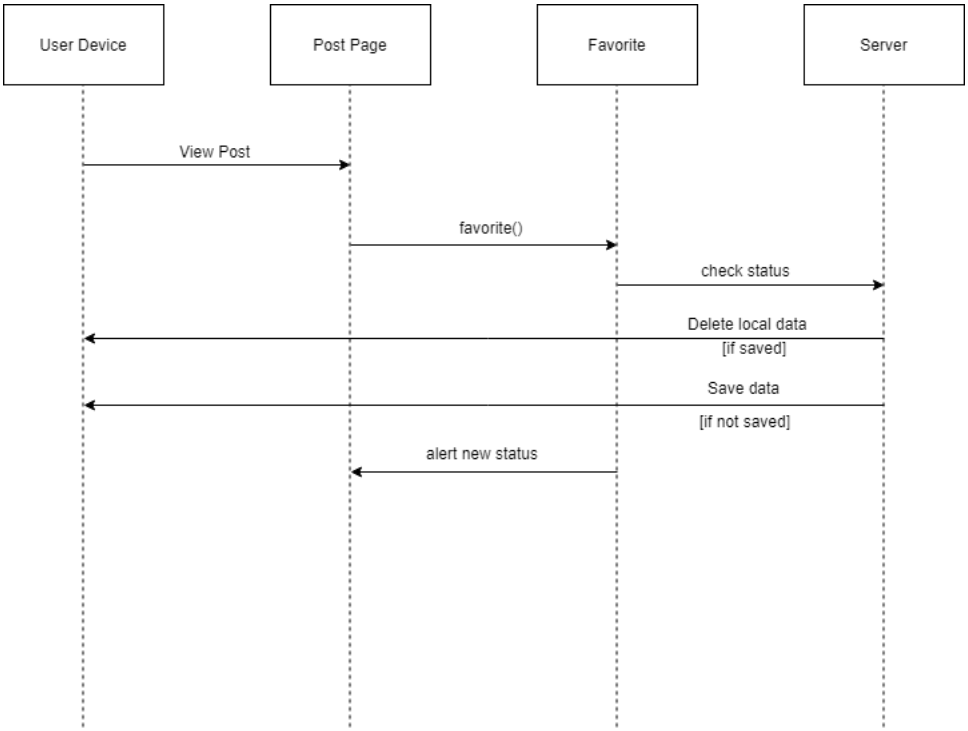
When a user doesnt have an internet connection, they should still be able to see posts that they have favorited.

- Store them in memory
- Posts will show instead of the no internet page

1.9.1 Corresponding Tasks

This will enable the user to view their saved favorites while they are offline. This will be due after offline capability and favorites are implemented. The posts within the favorites list will be saved locally and viewed there. Paired Programming time: 5 hours.

1.9.2 UML Sequence Diagram/Spike



1.10 Offline Capability

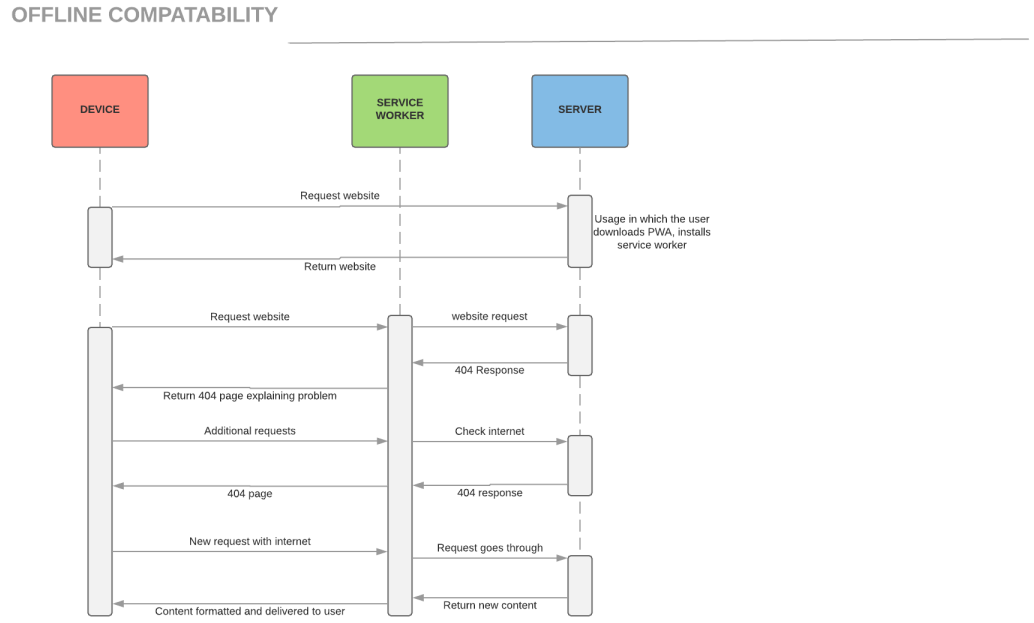
When a user is offline, they should be served a page telling them they have no internet. All urls should load.

- Use a service worker to monitor connections and requests
- If theres no connection show a catch all page (exception is favorites page)

1.10.1 Corresponding Tasks

This will enable to user to use the website when they have no internet connection. IT will serve to simply show the user that they arent connected to the internet. This is pretty easy to implement. It depends on the ability to save websites as an app and the service worker. Paired Programming time: 8 hours.

1.10.2 UML Sequence Diagram/Spike



1.11 Browser Compatibility

Web app should work in both chrome and safari.

1.11.1 Corresponding Tasks

The PWA will work across different web browsers. Our primary targets are safari on iOS and chrome on android. Because this depends on the entire app, it will be due late in the development process once nearly everything is implemented. This allows us to make things compatible all at once instead of continually needing to update things. Paired Programming time: 4 hours.

1.11.2 UML Sequence Diagram/Spike

Talktatiel should be designed to be compatible with modern browsers, meaning that it will display and behave the same on targeted browsers. Talkatiel is meant to be a PWA, so we dont need to worry about traditional desktop browsers. We are dealing with mobile devices, so we want to make our PWA work on mobile browsers. The primary browsers on the mobile market are Chrome, Safari and Firefox. In order to properly implement compatibility, we must focus on two things: programming patterns and testing. We must establish a set

of programming practices that lend themselves to writing compatible code. Two things that will help with this are doctyping and conditional comments. Doctyping signals to the browser what sort of document it is dealing with. Different documents have different ways of displaying, so making it clear that the PWA is a .aspx and not .asp will result in browsers correctly handling our site. Conditional comments are sections of code that are conditionally commented. For example, say we have this section of code:

¡link type="text/css" href="style.css" /¿ ¡!–[If IE]¿ ¡link type="text/css" href="IEHacks.css" /¿ ¡![endif]–¿ ¡!–[if !IE]¿ ¡link type="text/css" href="NonIEHacks.css" /¿ ¡![endif]–¿

sections This section changes the style sheet link depending on if the browser is internet explorer or not. If it is internet explorer, is loads a CSS file that has hacks that make internet explorer work. If the browser is not internet explorer, it uses a cross-browser CSS file that behaves better. We will not have as much of an issue with compatibility because we do not need to cater to Internet Explorer or Edge.

Also, we need to place an emphasis on testing. We need to go beyond just running it and seeing what happens. We will need to test on a variety of devices and browsers, along with testing tools such as Lighthouse. We will use tools online where you provide a URL and it tests the site on selected devices and browsers. This will help us make sure that the user experience of our PWA is the same across the board. Tools like Lighthouse will test the PWA and make sure that the website behaves as a website should and follows the website best practices set out by W3.

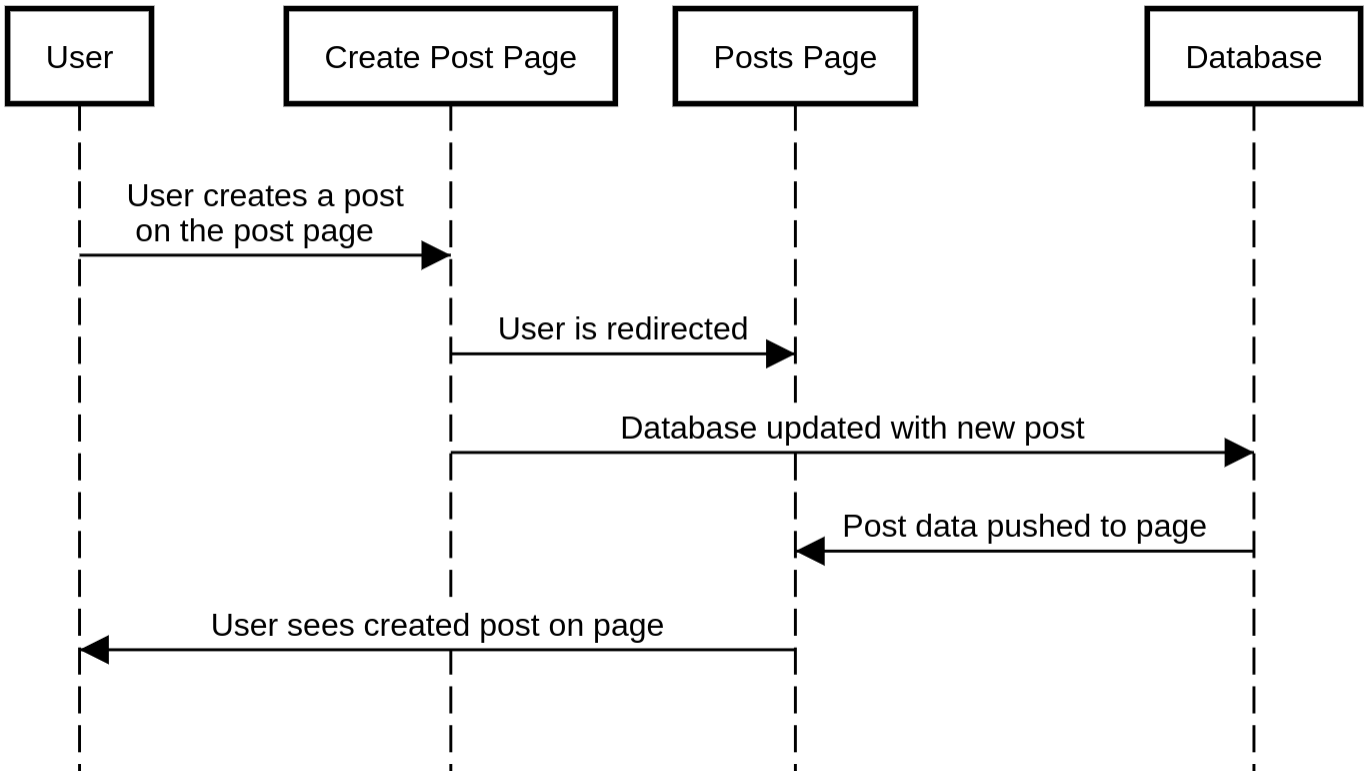
1.12 View newly created Post

When a post is created by a user, when they go back to the feed they should see it immediately if sorted by new.

1.12.1 Corresponding Tasks

When a user creates a post, they will be able to immediately see it at the top of their feed if it is sorted by new. This user story depends on adding a post, adding a comment, refreshing, the feed, and sorting. This is a low priority feature. It will be due at the end, if at all. Paired Programming time: 3 hours.

1.12.2 UML Sequence Diagram/Spike



1.13 Collapse Post Responses

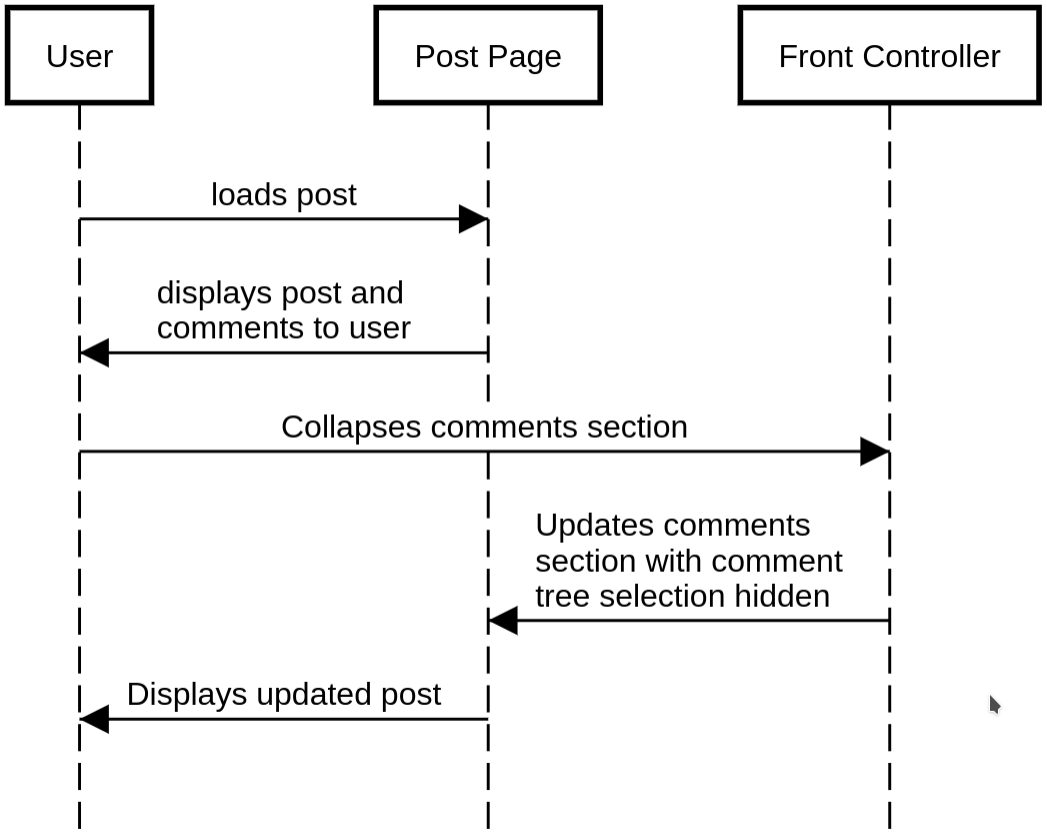
When viewing responses to a post, user should be able to collapse a response, which will collapse that post and all the children.

- Collapsing can be toggled by clicking on the response again

1.13.1 Corresponding Tasks

When a user clicks on a post response/comment, the post should be able to collapse. This will hide all the child comment to the response and make the clicked on response smaller. When it is clicked on again it will return to original state. This will serve to save space on the screen and hide what isnt needed. This user story depends on adding a post, adding a comment, refreshing, and the feed. IT will be due pretty late as it is an enhancement. Paired Programming time: 4 hours.

1.13.2 UML Sequence Diagram/Spike



1.14 Secure HTTPS

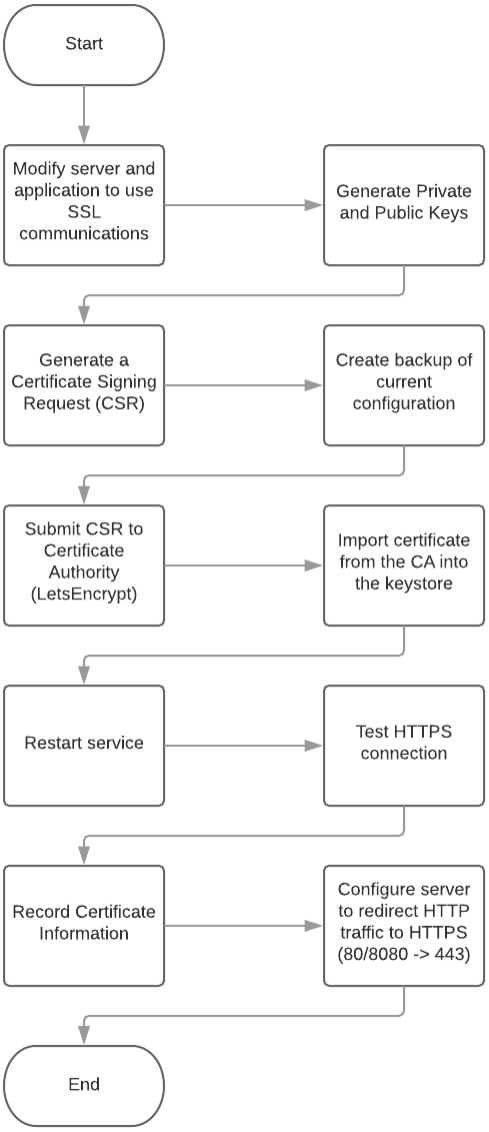
All communication should be through an ssl connection.

- Need a certificate from https://letsencrypt.org

1.14.1 Corresponding Tasks

This consists of setting up the server that delivers content with a security certificate. The certificate can be gotten from https://letsencrypt.org . It will enable our server to deliver all content over https, improving security and following the guidelines of a PWA. This will depend on the server side of things being implemented, which depends on the Refresh user story. Paired Programming time: 4 hours.

1.14.2 UML Sequence Diagram/Spike



This is the spike description for the implementation of secure HTTPS communication of Talkatiel. In order to communicate to users over an SSL connection, we must obtain a CA certificate for the application. In order to get this we will user the free service Lets Encrypt. They provide free SSL certificates and signing for websites. To get a certificate, first we must modify the server and application to utilize HTTPS traffic. Next we must generate public and private PGP keys for the server itself. Then we generate a signing request certificate and send it to LetEncrypt. They will validate and sign it before returning it to us. Then we link the application to the certificate and restart the application. Here we test the HTTPS connection to make sure it was implemented correctly. Finally, we save the certificate information and setup a redirect to redirect traffic on ports 80 and 8080 to 443. This will restrict our server to HTTPS traffic only, which is preferred and more secure for our users.

1.15 Save Website as App on Phone

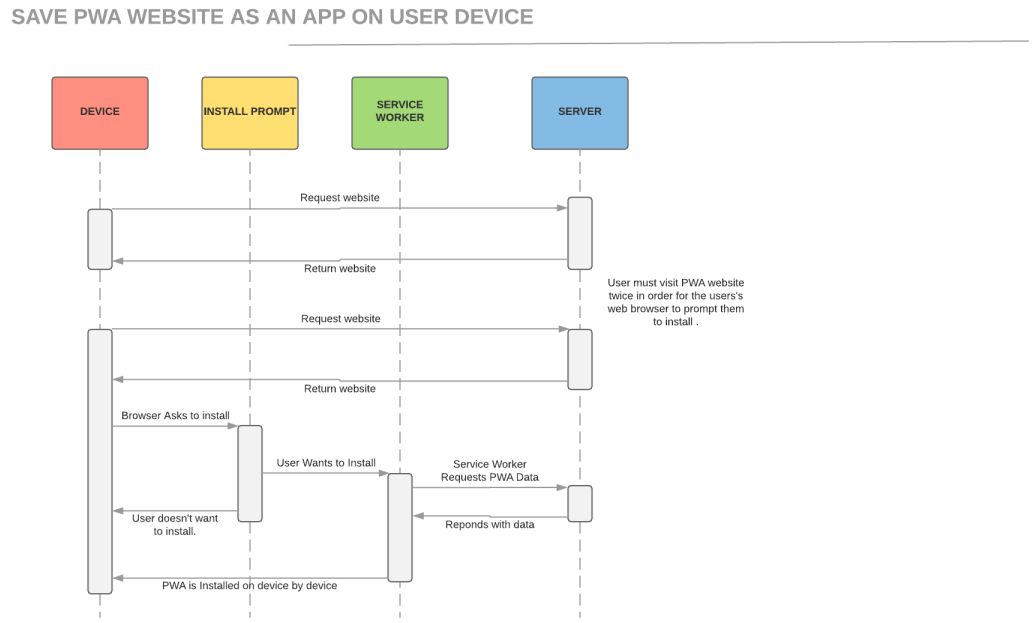
Website should behave like a pwa and be able to be saved to a phones home screen.

- Add a web app manifest
- Setup unique icons for the screen etc.

1.15.1 Corresponding Tasks

This user story depends on the app manifest. This is a document served with the website that tells the browser what the website is. Ours will signal that it is a pwa and is able to be saved to the device. This will depend on the service worker. It will be due directly after the service worker is completed. Paired Programming time: 10 hours.

1.15.2 UML Sequence Diagram/Spike



1.16 Responsive Web Page

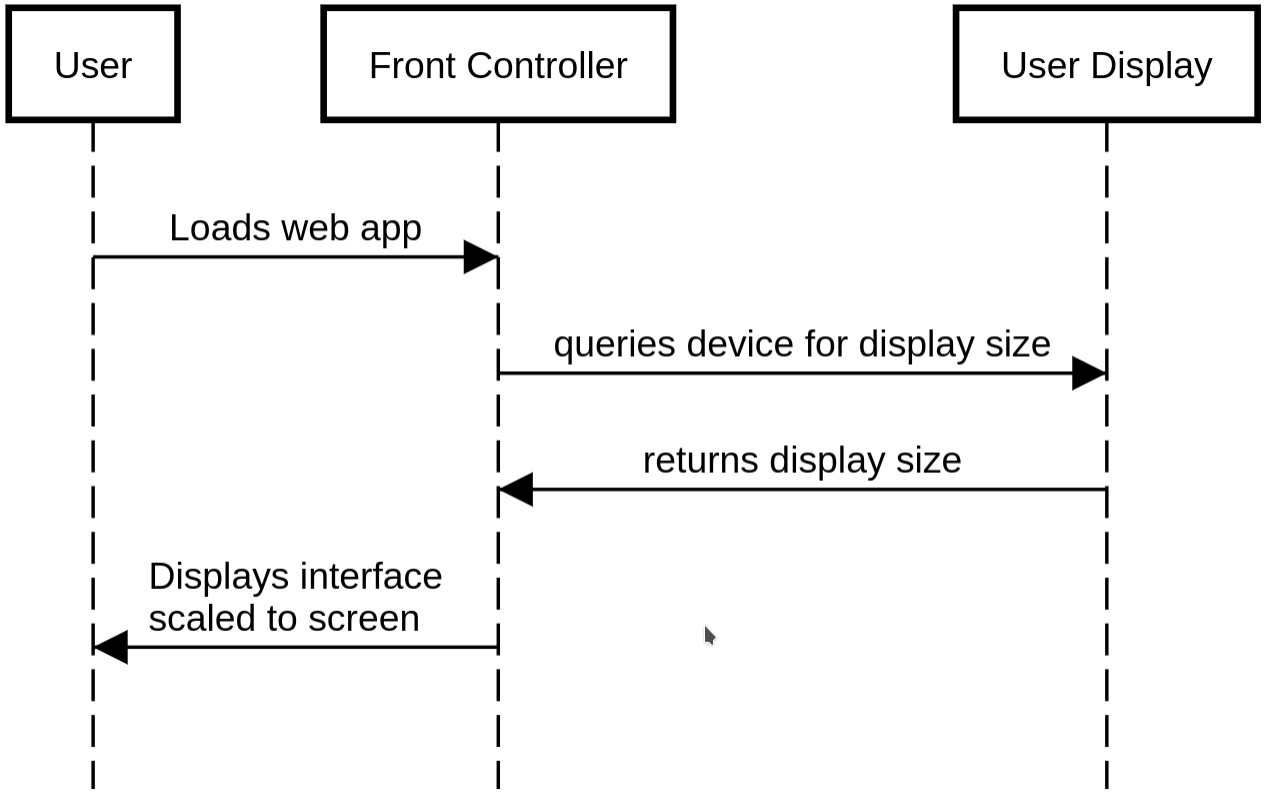
Content should scale with the size of the viewport/screen

- Default viewport size should be defined in the app manifest

1.16.1 Corresponding Tasks

The delivered web page should be responsive. This means that as the screen size changes the content scales to best fit. This can include removing flair or other things to save space. This should only be needed for the desktop view, which will display a page saying to use a mobile device. IT will depend on most of the rest of the project being complete. Paired Programming time: 6 hours.

1.16.2 UML Sequence Diagram/Spike



2 The Stories due Next Week

Creating the web app to send and receive data into the server is the most important part of the project. We have two groups, one working on frontend (Aidan Grimshaw, Yufei Zeng, Ryan Sisco), and one on backend (Iliana Javier, Brendan Byers). Communication is very important between the two teams, and getting both parts to work together is the most important part.

The user stories that have the most priority, according to the customer, is Add a Post, Add a Comment, and Like/Dislike. These are needed in order to get content onto the server, and get a starting point for the app. The rest of the features should not be implemented until these 3 are completed. The frontend team will develop the javascript and css according

to our mockups. The backend team will help handle importing the data and exporting the data. Error handling will also be done according to the mockups.

The secondary tasks will be completed after the connection between the server and user is more grounded and bug-free. Once that is true, development on the tasks will begin. The backend team will handle receiving data from the user, and the frontend team will develop the different pages needed for the tasks. Error handling this data will not be as difficult, because there is no raw user input to check. Security should be easiest on these as well, with little risk for a dangerous bug here.

The estimated time till completion on the frontend is one week, on 3/1/18. This should give lots of time to create a UI that looks very similar to the mockups. The backend estimated time is 3/1/18 for proper communication between user and server. Security will take longer, with no estimated time due to the long process of investigated and auditing of our system. This means that the app will have some functionality and be considered completed while security is still being updated throughout the project. Getting our system working with data is our top priority, and before all of these user stories can be done, this needs to happen. This will be done with a simple webpage for testing with the database. Security will have the task of checking user text and possible attacks. This beginning code is the most vulnerable to malicious users, so it is important that it works early on. After the communication is established, the teams will be able to break off and continue development on their tasks. By working this way, we believe that our project will work extremely well, and have a complete backend and frontend that can be swapped if one or the other does not work out, or if hosting becomes an issue.

3 Meeting Report

3.1 Progress Made this Week

This week we worked on gathering User Stories and planning their implementation. We prioritized the 16 user stories into three categories: Due this week, due next week, and reach goals. Each story was further partitioned into a list of corresponding tasks to clarify how it will be implemented. To tackle implementation we created Front End and Back End teams and divided the tasks between them. We also divided the user story UML Sequence diagram creation equally among the team members.

3.2 Plans for Next Week

Next week we will begin to code the highest priority stories, such as Add a Post, Add a Comment, and Like/Dislike. We will be using the tools and coding style we have documented in the past weeks to do so. Communication between the Front and Back End teams will be emphasized to ensure all team members are on the same page and code implementation can be as efficient as possible.

3.3 Team Member Contributions

- User Stories - Brendan Byers, Ryan Sisco, Iliana Javier Aiden Grimshaw

- Corresponding Tasks - Brendan Byers
- Stories Due Next Week - Ryan Sisco
- UML Sequence Diagrams - Ryan Sisco, Aiden Grimshaw, Yufei Zeng, Iliana Javier, Brendan Byers
- Meeting Report - Iliana Javier