Progress Report - Increment 2 -
Group Streamster

1) Team Members

Isaac Wolfe- iww15- IsaacWolfe

Laura Sarmiento - lss16b - Lulairu

Carlos Alfonso - caa16f -

Vanessa Myron - vim18 - Vanessa Myron

Ryan Diaz - rd18d- ryandiaz1090

2) Project Title and Description

Streamsters is a Chrome extension that is made to allow users to add both Twitch and Mixer accounts to view streamers they follow and get an extension that notifies them when the streamer is live. We support them having the ability to change their game immediately, get a view count, get length of stream, send out custom messages to followers, and most importantly accurately notify when streamers start streaming from both websites.

3) Accomplishments and overall project status during this increment

In this Iteration we focused on backend support for the variety of front end elements we hope to have displayed in Iteration 3. We added support to grab and store necessary data from the user. We increased the amount of information displayed to the user as well as set up the backend to be able to store stream information in the user's browser cache.

4) Challenges, changes in the plan and scope of the project and things that went wrong during this increment

One of the challenges that we faced during the design stage of the project was getting the user icon.

Another issue that we discovered was getting "404" with respect to implementing the profile picture.

One large issue that we are having to work around now and try to fix for the next iteration

is consistency across the site and on closing the webpage. We need to use cookies to keep streamers that have been added and this will bring a new challenge as none of us have worked previously with using cookies.

Due to Covid-19 working remotely in a team has been a challenge for communication as well as keeping track of time schedules. We have started using Zoom to work as a group for big deadlines and discussions and still use GroupMe and Discord although overall remote teamwork has been harder.

5) Team Member Contribution for this increment *,*

including the sections they wrote or contributed to

a)

i) *Ryan* - Working on handling permanent storage of following using the google chrome storage api.

ii) *Isaac-* I have been working with the front and backend of replacing my addStreamers webpage with the popup page for better use and styling that page as well as adding more functionality.

iii) *Laura* - I worked in the background themes and colors of the webpage by putting a JQuery plugin in settings.html

iv) *Vanessa* - Trying to implement getting the followers and following count

v) *Carlos* - Contributed to the backend of the project, adding changes to the ability to add streamers to the list.

b) *the **requirements and design document**, including the sections they wrote or contributed to*

i) Ryan - Filled out the basic classes information and functional requirements for the project.

ii) Isaac - Created base web design layout document that was later replaced.

iii) Laura - I write the Sequence Diagram and help Issac with the web page base design

iv) Vanessa - Implemented the design interface for followers count

v) Carlos- I wrote the case Diagram and helped Laura write the Sequence Diagram. Wrote Some of the non-functional requirements. Contributed where was needed.

*c) the **implementation and testing document**, including the sections they wrote or contributed to*

i) Ryan - Tested APIs using execution-based testing.

ii) Isaac - Testing adding streamers page on popup.html for functionality and multiple instances of streamers.

iii) Laura - Tested background color change.

iv) Vanessa - Tested the followers count section

v) Carlos- Worked on the execution based non-functional testing section. Not much has changed in this section compared to the last iteration.

*d) the **source code** (be detailed about **which** parts of the system each team member contributed to and **how**)*

i) Ryan - Wrote changes to popup.js and background.js on storage branch

ii) Isaac - Added to the popup.html and replaced addStreamers with popup

iii) Laura - I wrote changes in settings.html and jquery.themes.css

iv) Vanessa - Wrote followers count function

v) Carlos- Wrote backend support to fix errors we were having in our table. Also added support for offline streamers, so they can be added to the users followed streams. Added other minor cosmetic and syntax changes.

*e) the **video or presentation***

i) Ryan - Walk through the parts I worked on for iteration 2.
ii) Isaac - Walked through reason for change to popup.html being base add streamer page now along with functionality and eventual layout plan
iii) Laura -Walked through jquery.themes.css and background color change
iv) Vanessa - Walked through what I have so far with the followers count
v) Carlos- Had speaking role and edited and compiled video.

6) Plans for the next increment

We need to focus on creating a system for maintaining selections made on the web page with theming and adding streamers so it is consistent and does not reset upon exiting out. As of now we also need to implement more of a default theme as well as cleaning up much of the

website to look cleaner and more consistent across the entire webpage.

7) Link to video *Paste here the link to your video*
https://youtu.be/E1EKHloMwdA

R&D

Overview (5 points)

We are proposing a chrome extension to manage your favorite Twitch and Mixer streamers. Users will receive notifications when their followed streamer goes online, what platform they're on, and even what game they're playing. Streamster will consolidate your favorite games and streamers into one easy to use button.

Functional Requirements (10 points)

1. Background Themes

2. Setting pane on extensions homepage

4. Notifications for when streamers go live

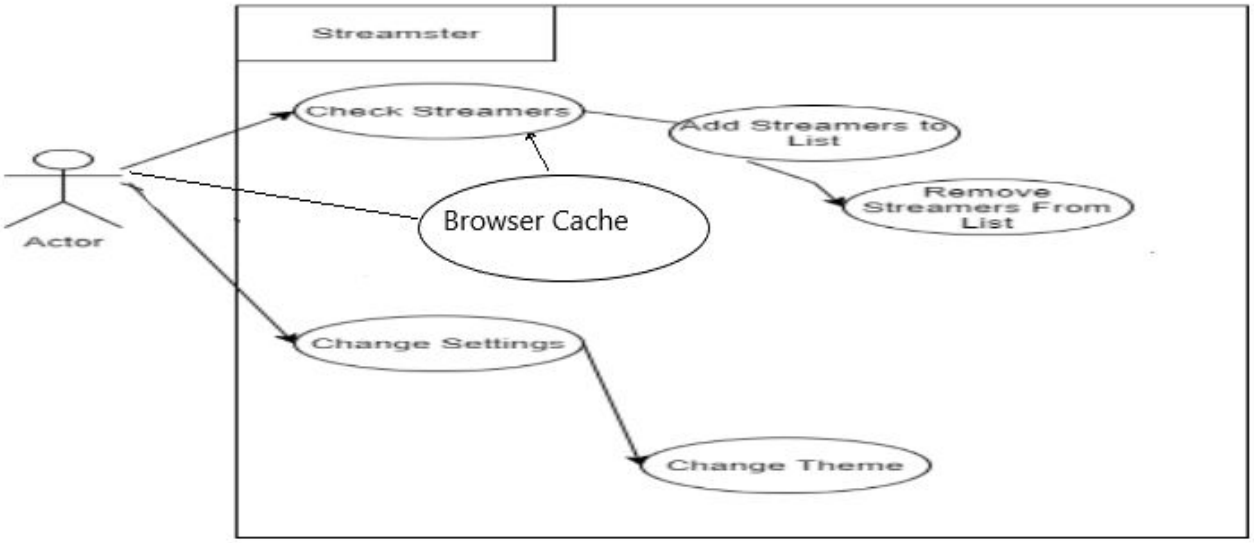
5. Clickable window from extension to view current streams with data such as viewers, length of stream, game, and streamer

6. Custom sort methods of how one wants their list of active streams sorted (such as length of stream, viewer count, game, or streamer name)

Non-functional Requirements (10 points) Perfectly optimized design, the app doesn't require much security besides hiding our own API client key. Webpage

1. Reduce number of API calls through cache evaluation or block get calls
2. Make theming consistent across web pages

Use Case Diagram (10 points)



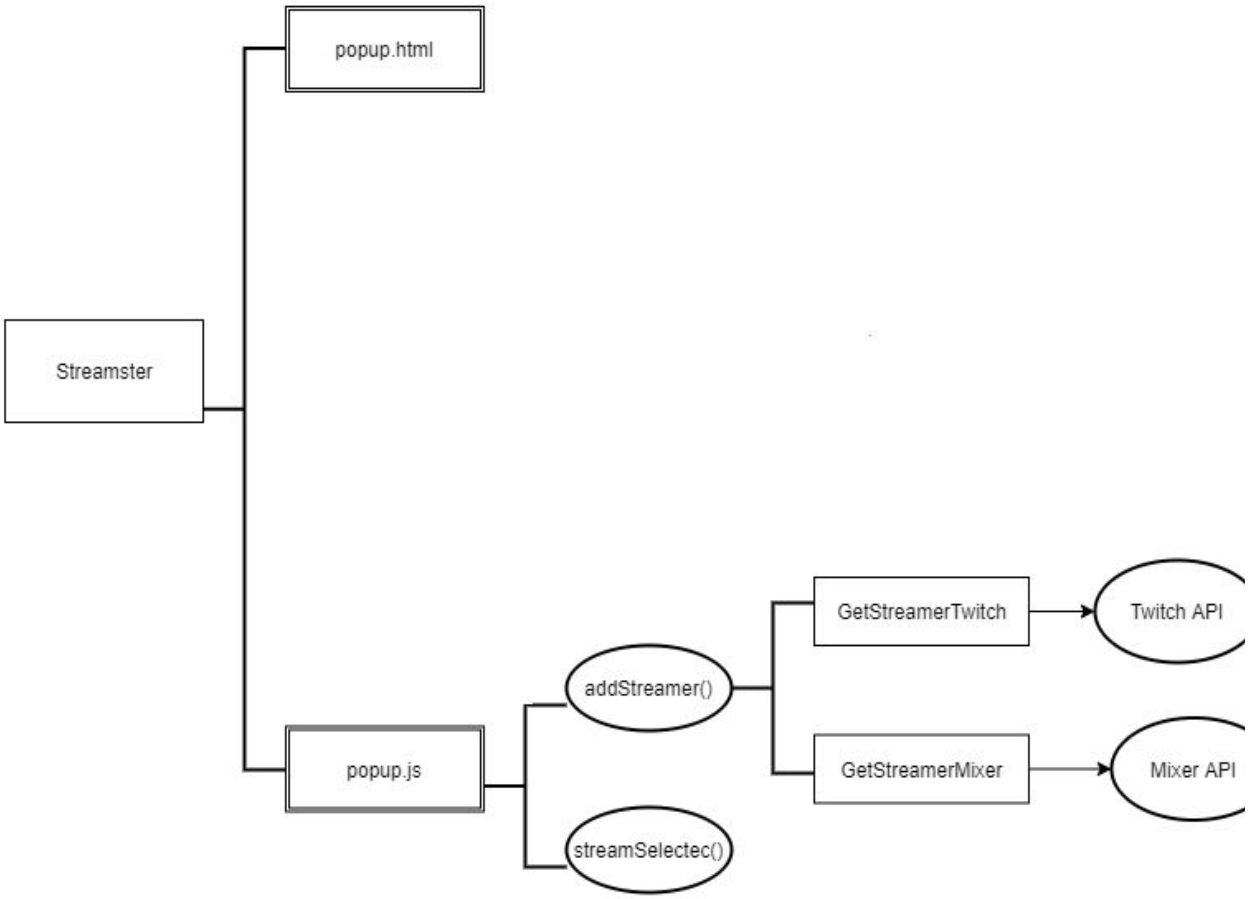
Class Diagram and/or Sequence Diagrams (15 points) ***Classes:***
Settings *- Handles user settings such as themes, adding their account names to get their follows*

Popup *- The html page that pops up when the chrome extension icon is clicked. Will show a minified version of the following class*

Following *- A full webpage that will display more in depth information about who the user is following*

Background *- Place to store user data either through the chome.storage api or through our own backend. This will handle continuity to the program.*

A ***Sequence Diagram*** *simply depicts* ***interaction between objects*** *(or* ***functions*** *- in our case - for non-OOP systems) in a sequential order, i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.*



Operating Environment (5 points) The software will run within the Google Chrome browser for each user that installs it. All user interaction will be through the Google Chrome browser.

Assumptions and Dependencies (5 points) Assumed factors -> We assume that the API will not change in the future. We also assume that none of the libraries will depreciate. We also assume that our operating system (in this case Google Chrome) can handle all the requirements set forth in our project. We are also using jQuery Themes to handle making themes.

Software Implementation and Testing

Programming Languages (5 points)

List the programming languages use in your project, where you use them (what components of your project) and your reason for choosing them (whatever that may

be).

Javascript, CSS , HTML are our primary languages. We will be using Javascript for the back end and CSS + HTML to handle our web applications. Our reasons for choosing these languages stems from our need to use certain API's like Twitch and Mixer

Platforms, APIs, Databases, and other technologies used (5 points) *List all the platforms, APIs, Databases, and any other technologies you use in your project and where you use them (in what components of your project).*

We will be using the Twitch and Mixer API to make calls so that we can get up to date stream information. We will consider using a Database most likely MongoDB if the need arises.

Execution-based Functional Testing (10 points) *Describe how/if you performed functional testing for your project (i.e., tested for the* ***functional requirements*** *listed in your RD).*

We tested API calls to both Twitch and Mixer and made sure that we could deliver on our most important feature to show a streamers name, if they are online, and how many viewers they currently have.

Execution-based Non-Functional Testing (10 points) *Describe how/if you performed non-functional testing for your project (i.e., tested for the* ***non-functional requirements*** *listed in your RD).*

One way we could perform non-functional testing is to exclusively test the calls and commands we are doing from the API. We consider the API to be our weakest link in the stability of our program, therefore Testing the security and speed of our API uses will allow us a more concrete idea of the overall efficiency of our project.

Non-Execution-based Testing (10 points) *Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).*

We review the code posted on github to check for invalid packaging or other possible errors having to do with how our code fits together. We also demonstrate any new code so that we are all up to date on it.