**CS 410 Final Project (Intelligent Browsing) Team SDK Documentation**

**1) An overview of the function of the code (i.e., what it does and what it can be used for).**

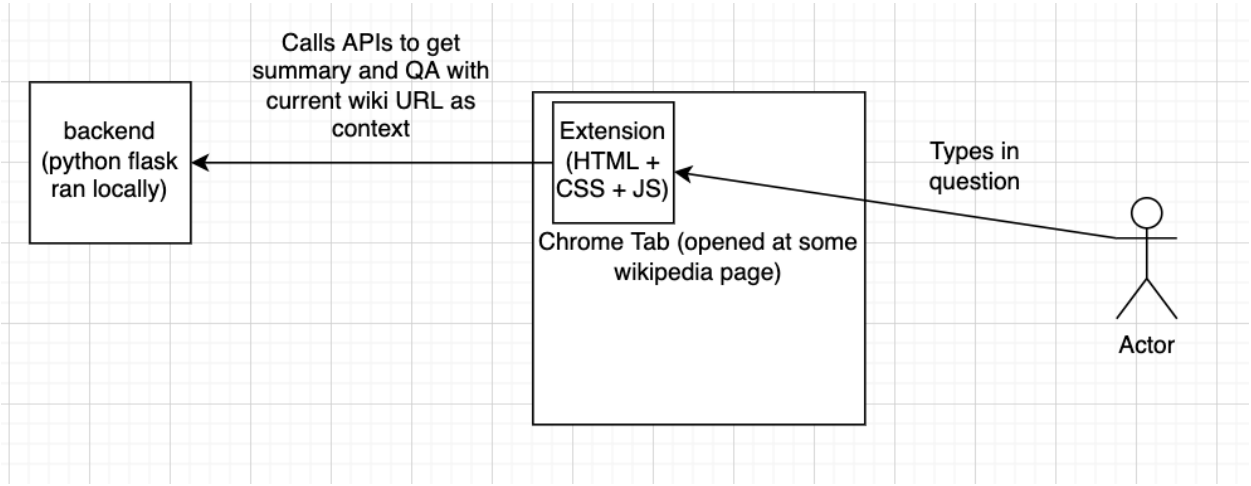
For users of wikipedia for information retrieval we produced a chrome extension product to assist in their task. Our extension is capable of scraping the details of any wikipedia page and offering 3 features. Feature one is a question answering tool where users can ask questions specific to the page and receive answers that are informed by the page content. Feature 2 provides a summary of the page contents and feature 3 provides useful images extracted from the page for quick lookup.

**2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.**

We have 2 main components on this project:

- Backend = serves the API used by the extension to display summary and/or get answers for the Extension. The backend code is written in python using Flask (The code for this is located inside the backend folder). The backend also makes use of NLP libraries to make the predictions.
    - Major Libraries used in the backend and some of the uses (details are located in backend/requirements.txt)
        - wikipedia - library used to help retrieve parsed contents from wikipedia servers
        - Flask - backend framework
        - nltk - used for the tokenizers and WordNetLemmatizer in QA
        - scikit-learn - used for QA to calculate TF-IDF and cosine vectors
        - sumy - for summarization using LexRank algorithm
- Extension = shows the UI and accepts input from the actor/user. It will also make calls to the backend
    - JavaScript, HTML, CSS, JQuery, FontAwesome, Bootstrap, Chrome Extension APIs

### High level overview of application flow




### Backend Details

High level overview of some of the files in the backend. Instructions to setup is on the README.md

#### flaskr/__init__.py

This file is used to setup the initial flask application and also to hook a function that automatically pad responses with **CORS** friendly headers. It is also used to register the necessary API routes needed using flasks' blueprint class.

#### flaskr/questionanswer/blueprint.py

This file is used to set up the routes needed for the API; it just acts as a router/controller. This file will also be the entrypoint in our application layer whenever an API is hit.

#### flaskr/questionanswer/qamodel.py

This file contains the logic for the NLP related stuff, also including getting the wikipedia article using the library. This is called by the blueprint.py file.

**Model Context:**

**Question Answering:**
Our Question Answering implementation leverages a combination of services from the nltk and scikit-learn package to accomplish the task. Nltk lemmatizes and tokenizes our page content for scikit learn input. Now with the altered input we use scikit-learn to calculate TF-IDF and cosine similarities in order to find the most applicable answer to the user's questions. In our testing we also tried BERT-large, Albert, and TinyBERT. We found BERT-large to perform the best however it would be an immense time commitment to the user with the user first downloading the pretrained model and then caching it into their systems then from there each prediction through the extension would take several minutes. AlBERT had decreased performance from large BERT and although the predictions were received quicker overall the time spent was still large. TinyBERT performed the worst compared to all of our implementations tested which is why we decided to move forward with the Nltk + scikit-learn implementation.

**Summarization:**
Our summarization implementation consists of preprocessing text data retrieved from wikipedia pages using the wikipedia package in python. Then we use sumy to parse and produce the summary. In our testing we tried manual tf-idf summary extraction using nltk, summarization offered by the gensim package as well as LexRank, Luhn, LSA, KL-sum, and Textrank from the sumy package. LexRank performed the best amongst all of these summarization techniques. We also tried a BART implementation but similar to our issues with the BERT-large implementation for question answering, its predictions were too slow when used with our extension.

**3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.**

**GET /api/qa**

It expects two query parameters =

- q = the question that the user is asking
- wiki_url = the wiki url as a context that will be used to answer the question

It expects the header to also accept content type of JSON

This endpoint will respond with a JSON consisting of the answer to the question. This API uses sentence similarity between the question and every sentence in the question to get the answer of the question, it will have the score which currently will always be 1. More detail is described in the Model Context section.

**GET /api/summary**

It expects a query parameter =

- wiki_url = the wiki url as a context that will be used to answer the question

This endpoint will respond with a JSON consisting the summary for the wiki page and a list of images in the wiki page.

**Backend Installation Setup**

You can also refer to the **backend/README.md** in order to setup the backend

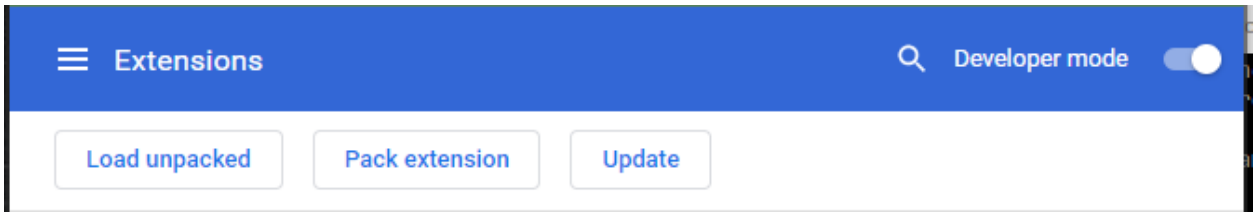
Make sure that you have no other processes running on port 5000 since we currently hardcode the port to 5000

1. Make sure you have python 3.8 setup (you can use conda to have multiple versions of python running in your computer https://docs.conda.io/projects/conda/en/latest/commands/install.html )
2. Clone the repository listed in github **https://github.com/sdgs72/CourseProject**
3. Go to the backend folder and run the following command (pip3 install -r requirements.txt) also see below image
    a. Installing scikit might be problematic since it depends on the OS version and some other global python packages you might have, please use google to find out on how to fix it
4. Run the python setup script to install nltk data packages using (python3 setup.py)
    a. In windows you might just use python instead of python3, but make sure that you are using the same python 3.8 environment that is running as in the previous step
5. Run the run script (./run)
    a. Please use windows powershell if you are in windows to run the run script
6. You can quickly test by opening the link on a browser(please use chrome)
    a. Your browser should hit the backend and render the JSON response

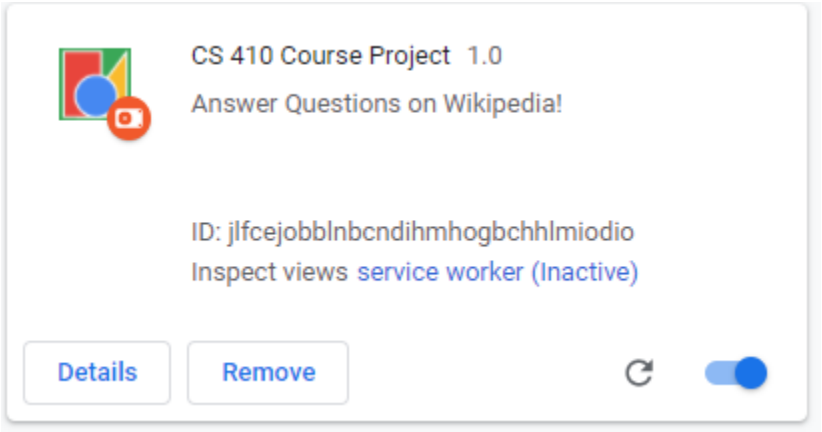
**Chrome Extension Setup**

Make sure that the backend is up and running before running the extension!

1. Install google chrome on your computer if you have none
2. Open the Extension Management page by navigating to **chrome://extensions** in your chrome browser. You will then be brought up to a window containing chrome extensions.
3. Make sure that the Developer mode option is enabled (located in the top right)
4. Click the load unpacked button and select the path to our project's **extension** folder. (IE …../CourseProject/extension )




5. The plugin should be loaded at this point ( it should show CS410: CourseProject)




**Please follow this page for more details on how to load a custom extension https://developer.chrome.com/docs/extensions/mv3/getstarted/**

Intuition of how the extension works

- It will only work on one single page of wikipedia
    - (i.e. if you open another tab in wikipedia, it will use that url instead)
- Once you are on the Wikipedia page, the summary will get loaded automatically
- While summary is loading, question asking will be disabled
- Once the summary has finished loading, the questions can be asked
- Answers will be retrieved and highlighted on the Wikipedia page

**Main Files Included:**

Manifest.json

- This file declares all the chrome extension apis that we will be using. For example, storage, tabs, webRequest, webNavigation. It also declares the name of the extension and the version of it.

Content.js

- This JavaScript file mainly controls the url that you are currently on your **actual** browser (where the wikipedia page is). Main function here is to collect the summary upon your visit to the wikipedia page. It also listens to changes in the storage in order to highlight the wikipedia paragraph with pink.

Popup.js

- This JavaScript file controls the extension UI (the frontend) and listens to events on the popup.html itself. Examples include updating the UI with answers, summary, images when retrieved. It also interacts with the storage to clear and refresh it.

Background.js

- This JavaScript controls the background requests that are made between visiting different urls. On change of the storage for the current question being asked, it makes a request to the backend and stores the answers in the storage to be accessed by the frontend.

Popup.html

- The frontend of the extension as you see it when you click on the extension

4) **Brief description of contribution of each team member in case of a multi-person team.**

Kaung Yang (kaungky2@illinois.edu)

- Developed the front-end interface for processing wikipedia pages, and used chrome extension APIs
- Added animations and improved on user experience of the extension, worked on some images retrieval to Backend API

Shyam Sridharan (shyams4@illinois.edu)

- Researched and developed both question answering system and summarization
- Assisted with integration to backend

Dion Hiananto (hianant2@illinois.edu)

- Developed backend APIs with Flask and wikipedia library, solved some FE <> BE related issue regarding CORS
- Assisted with research of question answering system and integration of summarization to backend