The design of our Riki application is built with the idea that we are going to be using python with a Flask micro-framework to implement the design. Python’s capabilities and limitations were considered when we planned what features we could implement.

The application starts in \_\_init\_\_.py which is in the wiki/web directory. Here we define our app’s configuration in the create\_app method: A screen shot of a computer

Description automatically generated

This create\_app method is called on the actual startup of the app which occurs in Riki.py here:

A black background with white text

Description automatically generated

Once the application has been started, the app waits for an API call (by a user typing in the URL of the website). The Flask framework handles these calls in the routes.py file inside wiki/web. Using the decorator pattern, Flask allows us to define API endpoints such as /index/ and then define behavior that we want to occur when this endpoint is routed to. Here is an example:

A computer screen shot of a program code

Description automatically generated

In this example, we create a model of our form SignUpForm if /signup/ is triggered. Then, if the user successfully fills out the form, we can automatically redirect them to the multi-factor authentication part of our application. Flask’s ability to seamlessly use routes to change between parts of our application makes it very easy to add new features.

One more thing to examine from the route is the final line with the return statement:



The render\_template method passes in an html file that utilizes the Jinja2 templating engine. Jinja allows us to use standard HTML5 in addition to Jinja’s unique templating features that allow us to use simple logic operations to display our code.

In summary, the use of routes in Flask is the key feature that allows us to use URLs as function calls to render the proper web page. This is where the majority of our time was spent. There is also important functionality to this application such as the core.py that defines important processing and page behavior of our application. However, this was in place before our project started, and we did not have to modify any of this preexisting code.

For our individual features, the design pattern typically followed defining a new Flask route as an endpoint to be hit. Then, in the route’s method for handling the incoming call, we render a new .html template displaying the page of our new feature using HTML5 and Jinja2. Therefore, we tried to maintain conceptual integrity across our features by designing them using the same pattern of creation.