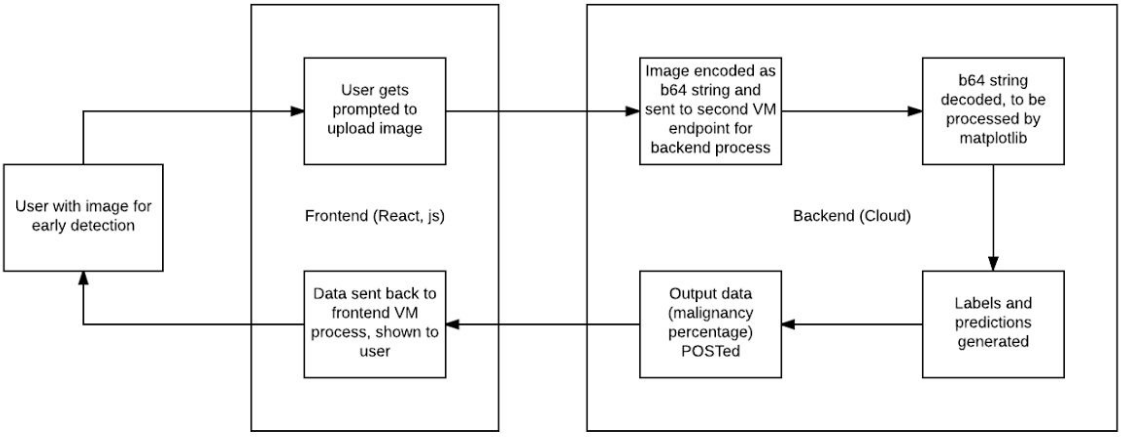
https://github.com/jasontrue24/BMEFinal-Project-WEBAPP
The working version of the final project runs at http://vcm-1838.vm.duke.edu/

The overall workflow of this webapp is described in the following figure and is more thoroughly described in the RFC document:



Front-end:

The front-end was developed using react.js and react-bootstrap. It allows a user to upload a photo of his/her skin into the upload zone - we used a component called UploadField from @navjobs/upload - and the website will generate the calculated percentage of the skin having developed melanoma. This percentage value is displayed next to the Results button as soon as the button is clicked. Other than the melanoma detecting function, the website intends to give the users a basic overview of what melanoma is and how to prevent it through some daily prevention guidelines anyone can follow.

For the basic framework set-up and the communication with the back-end to post and get the data, react.js was used. On top of the working code, react-bootstrap was applied to display an interactive, better-looking website; most of the modifications were made in index.html, upload.js, and index.js, because the original starting code provided implemented the muiTheme features from material-ui, which were not compatible with react-bootstrap features and modifications were necessary. Basically, the user can click on one of the three tabs at the top of the website that will take the user through automatic scrolling to the corresponding page. With the implementation of jQuery (please refer to smoothscrolling.js in the components folder of the real_frontend_Gina branch), a slightly smoother scrolling effect would have been achieved.

The completed front-end mentioned above functions correctly itself, but for some reason it currently cannot communicate with the back-end to get the data. The working version with the

link provided above unfortunately looks pretty much the same as the provided starting code, but has the front-end to back-end communication working successfully.

Back-end:

With the front-end displaying a website that allows users to upload melanoma photos, the back-end processes the image and uses a deep learning mechanism to analyze the data and give a prediction to users about the percentage of having developed melanoma. In the second virtual machine, Docker ran as a container that holds the tensorflow which is a library for data flow and processing. Tensorflow was used as the foundation for deep learning.

The back-end process runs the Dockerfile and docker-compose.yml described in the GitHub repository master branch. The front-end process uses Docker described in the frontend_Blake branch. We could not merge them due to the name requirement of the Docker files. One issue that often arose with Docker is the :8888 tree not updating as it should when multiple Docker processes were running. At first, both Docker processes were attempted on the same VM, using the screen software. This did not end up working properly due to failed communication between the two Docker processes. To resolve this issue, two VMs are used (each a different group member's).

Essentially, the back-end runs on a separate VM than the front-end process which allows for proper communication of data and modularity of our project. While this enables simplified work roles for our group, this is quite clunky and generally is slow due to the fact that the VM machines are not perfect. However, the front-end webpage is able to properly prompt the user for an image, send the decoded image as a base 64 string to the second VM's /classify endpoint, and receive POSTed data back from the second VM's /classify. More architecture involving the back-end can be found in the RFC description. With this basic setup, the webpage at http://vcm-1838.vm.duke.edu/ is able to properly report deep-learned predictions for possibly melanocytic images.