

Project Name: WAILA

Team Members

Nihal Dhamani (nnd365)

Vignesh Kumar (vk4433)



Contact Emails

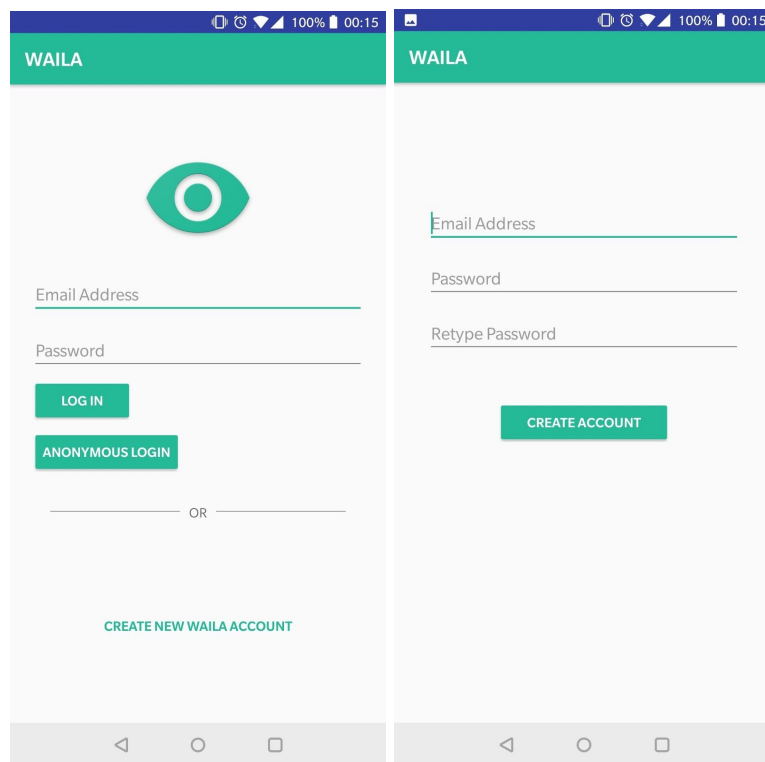
nihaldhamani@gmail.com

vignesh.1996@utexas.edu

Title: WAILA (What am I looking At)

Description: A real-time object detector app that will use a convolutional neural network trained using TensorFlow. WAILA provides functionality to learn more about the object and save it as a memory. Each memory can be stored on a cloud database with the timestamp and the location at which the memory was lived.

Authentication:



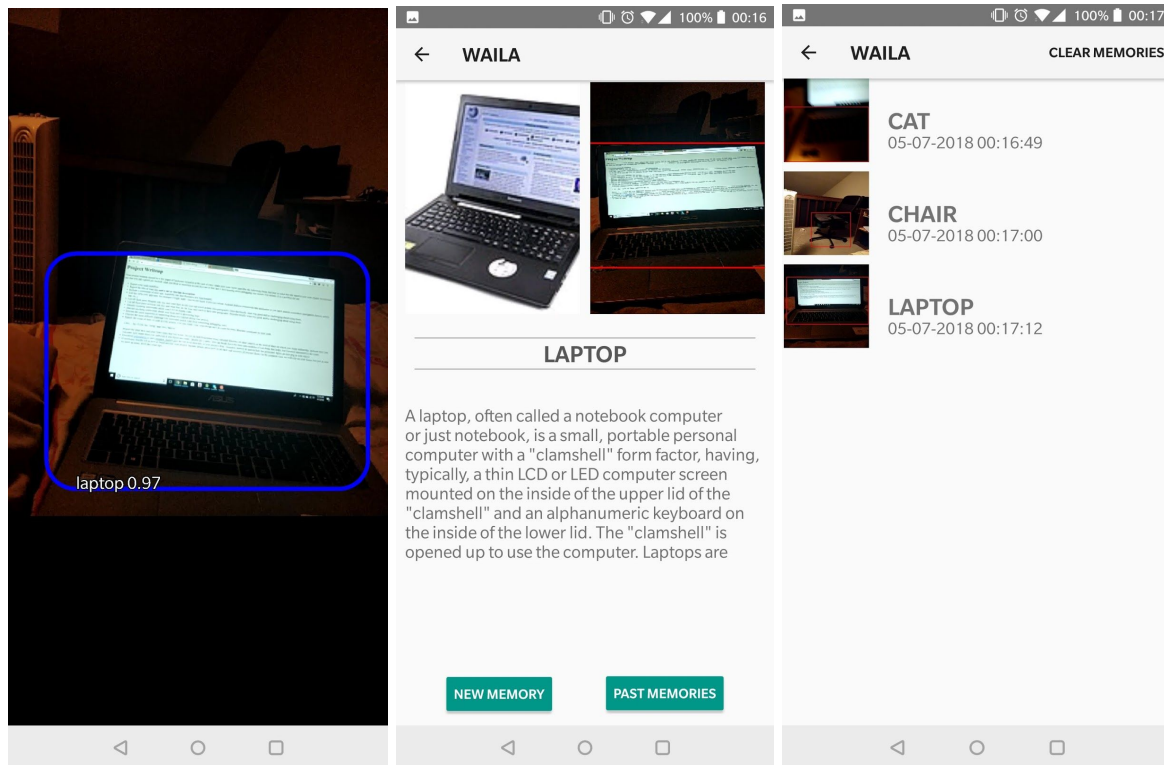
The image displays two side-by-side screenshots of the WAILA mobile application's authentication interface. Both screens have a green header bar with the text "WAILA".

The left screenshot shows the login screen. It features a large green eye icon at the top. Below it are two input fields: "Email Address" and "Password". There are two buttons: "LOG IN" and "ANONYMOUS LOGIN". Below these is a horizontal line with "OR" in the center. At the bottom, there is a link that says "CREATE NEW WAILA ACCOUNT".

The right screenshot shows the registration screen. It features three input fields: "Email Address", "Password", and "Retype Password". Below these fields is a button that says "CREATE ACCOUNT".

Here the user chooses to either login if he/she has an existing account. If the user does not want to login with credentials he/she can log in anonymously. If the user does not have an account he can choose to create an account which opens the create account fragment.

TensorFlow Object Detection:

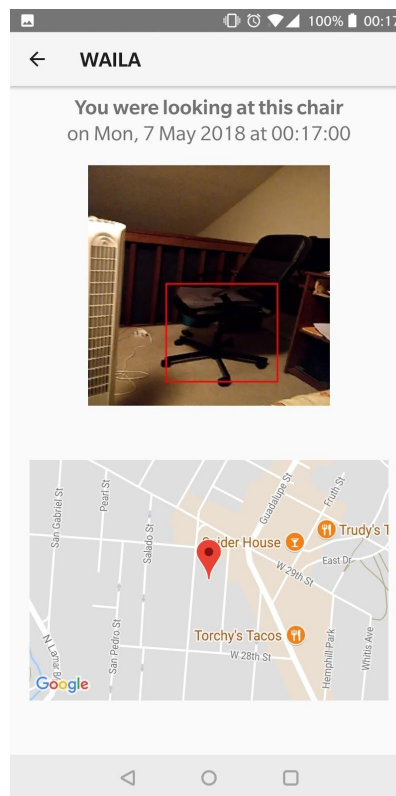


After camera permissions are granted, the TensorFlow object detection intent is started. Here the user sees a live video feed from the phones camera and they can point it at any object. If its an object TensorFlow detects, a bounding box is drawn around the object and a prediction is given. The user can learn more about the object by clicking anywhere within the bounding box of the object.

Once the bounding box for the object is clicked, using Wikipedia's API, WAILA return the details about what the object detected. It also returns another image from Wikipedia so that the user can relate the object they have seen with the picture. There are 2 buttons on the screen; "New Memory" allows the user to store this memory to their database/public database. If the user logs in with his/her credentials, they will store the memory in their private memory database. If the user logs in anonymously, they will store the memory in the open public database. "Past Memories" takes the user to the next page where he/she can go look at his/her stored past memories.

This is the list of memories in the database. If the user signs in anonymously, the list of memories is the public database. If the user signs in with their credentials, the the list of memories is their private database. The "Clear Memories" functionality only works for the private

databases, as no anonymous user has the permissions to clear memories from the public database. Clicking on any of these memories takes the user to the page with details of each memory.



This is the final activity that displays the details of the selected memory. This uses the google maps API to show where the memory was taken.

APIs/Third Party Services:

- 1) Wikipedia
- 2) Google Maps
- 3) Firebase (Database backend)

Third-Party Library: TensorFlow demo, we used this for our object detection activity. The most challenging part about this was integration and scaling to all screens. We initially had a lot of problems because this was an android app by itself and we needed to ensure the activity lifecycle is handled perfectly during integration between our Login Fragment and Wiki Post. Getting the On touch listener on the canvas was also quite challenging as we had to scale to work on different screen-sizes. Key part of this was testing on multiple devices and learning from our mistakes.

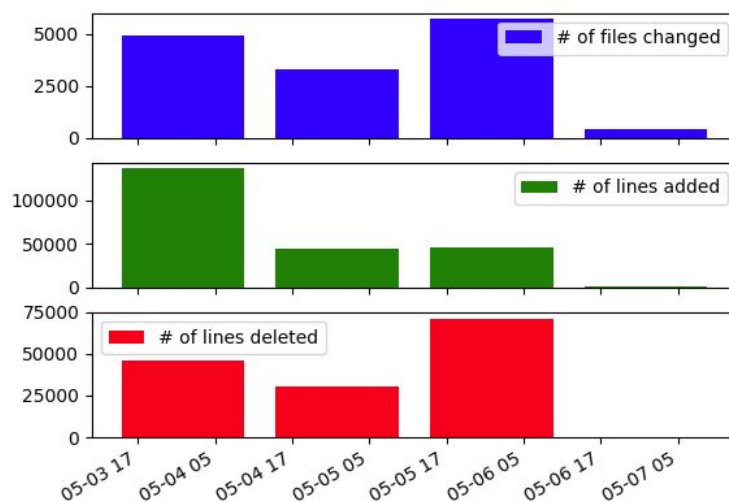
UI/UX: Our goal was to make the app very intuitive and sleek. Additionally, we really wanted the app to work with multiple screen sizes. As a result we mainly used constraint layouts in our XML files so that everything would work across multiple screens. Additionally, the tensorflow object

detector scales to screen sizes as well, so we made sure to convert clicked touch events and scale them according to the screen size of the phone so that the bounding box click works across multiple screen sizes.

Backend Processing: I think the most noteworthy thing about the way we deal with our database across our 5 different activities. We have a Java class called PhotoManager (misleading name) that controls every interaction from our database. Since we want the same database reference across all the activities, we used a singleton design pattern to ensure that we always have the same database inference. PhotoManager has a lot of useful functions such as updating the current user, uploading a photo object to firebase, helper functions that convert to and from different data types and also a function that gets all the photo objects for a given user. Additionally, PhotoManager defines an interface getDataListener, that serves as a callback from when dbView (our recyclerView activity) wants to query the database.

Most important thing we learned: The most important technical thing we learned from the project was how to get consistent data and database references across multiple activities. Additionally, the most important non-technical thing we learned was catering our app towards certain audiences. Currently, we have designed it for a target consumer that enjoys recollecting memories about things they've seen. As such, we designed it to be sleek and modern. However, we also considered other use cases for the application. One such use case would be targeting towards older people with memory problems. This app would serve as a way to recollect their memories; however, the layout and interactions would definitely have to be changed in order to target the audience.

Visualization:



Just a note, we started off doing this project in another repo and only created this repo a few days ago so we don't have much data as we deleted the older repo.

Most Difficult Challenge: Integrating the tensorflow demo application with our application was very challenging as we ran into several issues of maintaining the activity lifecycle. At the end we had 5 activities, all but one being a child of another. But the hardest challenge, was ensuring only one application was built. When we brought over the tensorflow application it had its own build and removing this build and adding it as an activity was tedious. Finally, after removing the intent-filters and manifests, we were able to build one clean application with a bug-free activity lifecycle.

Cloc ran over res and main directories (including demo code):

Language	files	blank	comment	code
XML	32	168	232	737
Java	35	1035	998	4467

Lines of code we wrote (calculated manually):

XML: 426

Java: ~850