Document your life's adventures!

App Link: https://cse134b-25420.firebaseapp.com/
Welcome to the Memento Github directory!

Introduction: This directory contains folders of the weekly progress on Memento. The initial weeks mostly comprised of abstract and higher-level implementation, whereas the later weeks consisted on programming the details and functionality.

Idea: The main goal for our app is to help user's easily document their memories. What we want to create is in essence a virtual photo album. During our initial need-finding stage, we came to the conclusion that there was online service that allowed users to document their travels. Given the increasing mobile nature of society, we realized that physical photo albums were items of the past. With Memento, we aim to provide a way for people to be able to document their memories on the go!

Contributors: Matthew Wen, Minh Vu, Scott Lim Homework repo for CSE134B

Memento

**Authentication:**
For our backend we used the built in Firebase authentication service that allows the user to create an account using their own email address and password, along with Firebase's integration with Google sign-in, allowing user to also be able to sign in with a Google account.

**Database and Storage:**
To store the user's information, we also used Firebase's real-time database and storage api and server. The real-time database is used to store user's profile information along with their memories in json format on the server side. Then our application would query the database and populate the page with user specific information to display to the user. The real-time database is only used to store text-based information, so in order to allow the user to upload their own images, we used firebase storage to accommodate for this feature. Inside the real-time database, we store the path that the image is stored in the the Firebase storage as one of the fields of the memory object in the JSON tree, so that once we have access to that memory we can simply query the storage using that stored path.

To keep information stored by every user only accessible for that user, we use the current user's ID as first level children of the root "users" node. Also, in order to make the inputted memories stored in a consistent format rather than the default lexicological order the Firebase real-time database works in, we use push() to generate a unique key as the child of the current user's ID. This allows the database to be ordered in the order that users input information, so that when the current window of memories are full and the user adds another memory, no refresh is needed on the page after the memory is added.

**File Organization:**
We decided to organize our application into several different HTML pages based on their logical functionality, so that we can encapsulate the page's behavior with their functionality, making our application and code more organized:

1. LogInPage.html
    a. Responsible for handling authentication, based on email/password login or Google's login.
    b. Without correct authentication, the user will not be able to go to our HomePage.html and ProfilePage.html.
    c. Users can access RegisterPage.html from here in order to create an account using their email address and password.
2. RegisterPage.html
    a. Responsible for handling the creation of an account using the user's email address and password.
    b. Error checking and authentication will also be handled here to ensure that the user's email address is not already in used.

    c. Users that pass the authentication when signing up will be directly logged into their account and will be redirected to HomePage.html.
    d. Users can also go back to the LogInPage.html from here.
3. HomePage.html
    a. Responsible for the CRUD part of our application
        i. Create - Allows user to create a memory and add it into their “My Memories” section by pressing “Make Memories” button.
        ii. Read - Allows the user to pull up any memory in a popup modal that they have created or memories populated by the application by clicking any picture in the “My Memories” or “Future Memories” section.
        iii. Update - Allows the user to update any of their memories by first calling Read on our application and then editing any of the information besides the name of the place and pressing “Save Changes”.
        iv. Delete - Allows the user to delete any of their memories by first calling Read on the memories they wish to delete and then pressing “Delete”.
    b. Users can access ProfilePage.html and pressing “My Profile”.
    c. Users can refresh the current page by pressing “Home”.
    d. Users can logout of their account by pressing “Logout”, which will redirect them back to the LoginPage.html
4. ProfilePage.html
    a. Responsible for storing user’s information such as their name, description, and a profile picture.
    b. Users can access HomePage.html by pressing “Home”.
5. Images Directory
    a. Holds the local copy of the background picture or assets for our application.
6. Future Memories Pic Directory
    a. Holds the local copy of the future memories picture or assets that we pre-load into our “Future Memories” section.
7. futureMemories.json
    a. Holds the information that we use to populate the “Future Memories” section. The data is stored in a json format.

**Code Architecture:**

On our HomePage is where all the CRUD functionality for adding memories or visiting a future memory is placed. We have two panels of 9 images each, one for memories the user has personally added or visited from our pre-filled list, and one for future memories where we have pre-filled a list of suggested popular tourist locations in California. Upon loading the page, we populate both image panels by querying our real-time database for the current user, and if it is their first time on the app we will populate their real-time database with our pre-selected future memories. Then, we go through the list of memories and future memories for the user, and use their path fields to change the src of the images to the ones that are stored in the database. We save the unique key generated by the push() API call inside the “data-key” attribute of each image in my memories, so that when the user clicks on that image we use that unique key to

again query the database to fill our modal with the right fields associated with the image the user clicked. This way, we only have one modal for my memories and one modal for future memories that we fill with the right fields after a user click, and not have 9 modals for each panel attached to each image for a total of 18, which would greatly decrease performance. We also picked modals to display memory field information rather than directing the users to a new page so that we save load time and that the user's flow isn't interrupted. The arrows in the image panels allow users to go to the next window of 9 images when there are more than 9 memories in the database, and we called functions that made sure the user clicking those buttons when there were no additional images to fit onto the next window would do nothing, for example, if there were 9 images in the database, 9 images in the first window in the page, and the user tried to click the right arrow for the next page. Lastly, we wanted the users to be able to smoothly transition one of the future memories we suggested for them into one of their own, after visiting the place in real life. We added a visited button to the future memory modal, which when clicked will remove the location from the bottom future memories panel, create a copy of that location in the user's database as one of their memories, popping up a modal to allow them to edit the rest of the fields such as description and rating before saving it. Upon saving, the image and associated memory would disappear from the bottom panel and appear in the top my memories panel.

**Front-end Presentation:**
For our front end presentation, we used Bootstrap and their framework to design, present, and layout our HTML pages. In addition, we used some vanilla CSS in order to style some parts of the pages the way we wanted to that Bootstrap didn't allow.

**Front-end Scripting:**
We decided to use vanilla javascript to implement the majority of our client-side application with only some small jQuery code used.

**Concerns, and Limitations:**
One major concern for our application is that we decided to implement the majority of it on the client-side. This completely exposes the organization and architectural design and structure behind our database and storage implementation, making it extremely easy for hackers or anyone with decent web knowledge to malignantly access and destroy with our application and database. The tradeoff we got from deploying our application on the client-side is speed versus security. However, because security is such an important issue when the user's personal data is at risk, in the future as our TODO we plan to move the client-side code that queries the database to the server-side so that our implementation is not exposed to the user. Then, we would just need to send HTTP requests to our server from the client-side to ask for the information without needing to worry about the underlying implementation of how to get that data, and the server would just send back the data for the client-side to populate to the user. The downside of this is that we will lose some speed on how fast the page can load to the user, but the application's security will be better, which as an end goal is more important in an application that contains user sensitive data like our application. In addition, our code is not as

readable as it can be because all of the javascript, css are in the same file as our HTML code. However, because we wanted the page to load as fast as it can without making unnecessary HTTP requests, this was the design decision that we came up with. Another major limitation of our application is that we do not have code that will compress images uploaded by the user. So if the user decides to upload a really big image, our application's speed will degrade. Currently, we do not have the option that allows the user to input more fields than what's been provided, so if the user wants a more customized experience to describe their memories, it is not possible right now. However, this is a feature that we are looking to roll out with the next update. Also, we are limiting the user to only viewing 9 of their memories at a time, instead of allowing the user to decide how many they want to see in a single window view. However, this was the decision we had to make due to performance concerns of fitting too many images on one screen. Indeed, our design decision to focus on a image-based scrapbook of sorts can be a severe detraction to our performance, since we have to load up to a total of 18 images on the screen. If we did a different design where the user views a list of locations they visited and clicks the text to then populate the screen with only 1 image at a time, we would gain a lot of performance. However, we felt that as an online scrapbook of photo memories, this would defeat the purpose of our app. Also, another small limitation of our application is that some of the intended functionality does not work on Firefox but it will work on Chrome. For example, the modal will close if the user makes a memory without inputting in any of the required field, but chrome will prompt the user a warning and not close the modal. Finally, the last limitation is that the pre-populated list of "Future Memories" is populated by us, so if a place that they have in mind is not on the list, they will have to input it themselves. And since we are responsible for updating the "Future Memories" section with new places, the speed of our release might not coincide with what the user expects.

**Performance Testing:**
We tested our web app with Regular 3G throttling and a Nexus 4 without caching:

LogInPage: 5.95s load
RegisterPage: 5.01s load
HomePage: 35.10s load
ProfilePage: 8.33s load

One big impacting factor of the page loading time has to do with how big the images that the user decides to upload. So if they upload images that are small in size, our HomePage will load significantly faster. The average image size that we performed this test under was ~170 KB, with 18 images total being loaded.

We tested our web app with Regular 3G throttling and a Nexus 4 with caching:

LogInPage: 0.24s load
RegisterPage: 0.20s load
HomePage: 1.60s load

ProfilePage: 0.80s load

Caching can greatly improve the performance of our app, since it is heavily based on images. After doing any CRUD operation on the HomePage, most images will be cached after initial load and the load time decreases by many factors.

**Demo**
**1) How to create an account**
**2) How to make a memory**
**3) How to edit a memory**
**4) How to mark a memory as visited**

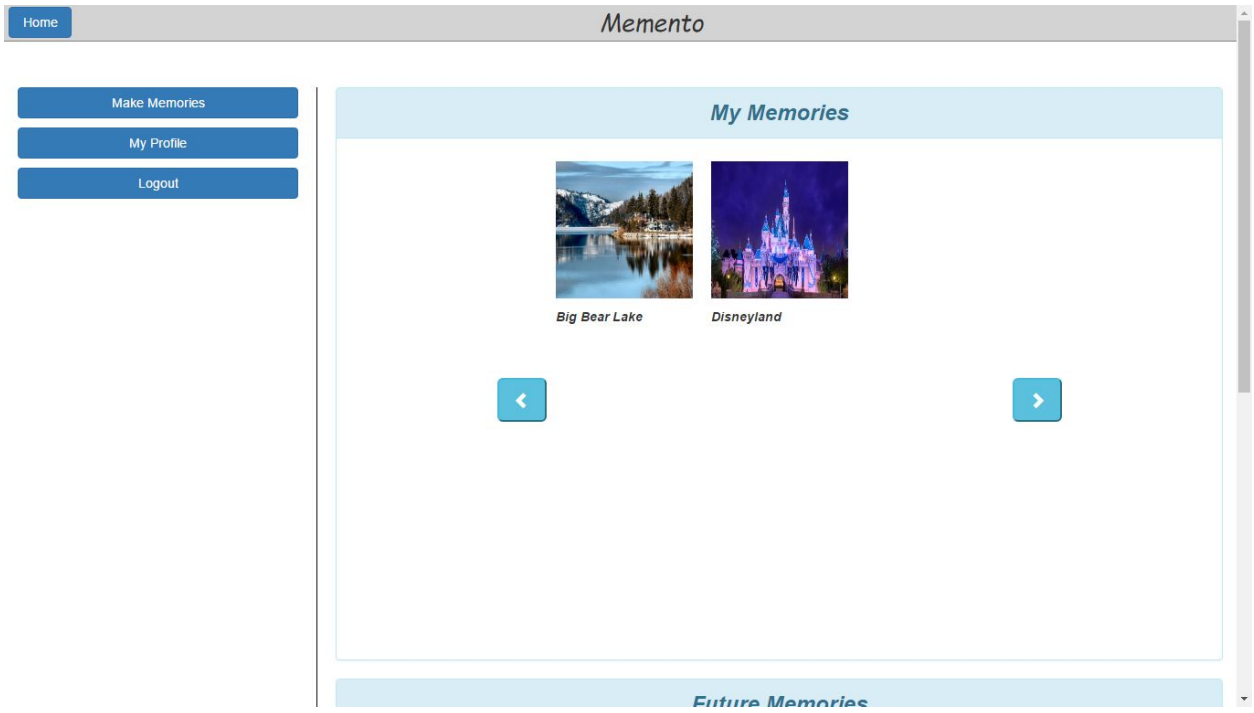
**How to create an account**

1) **Click on ‘Create New Account’**

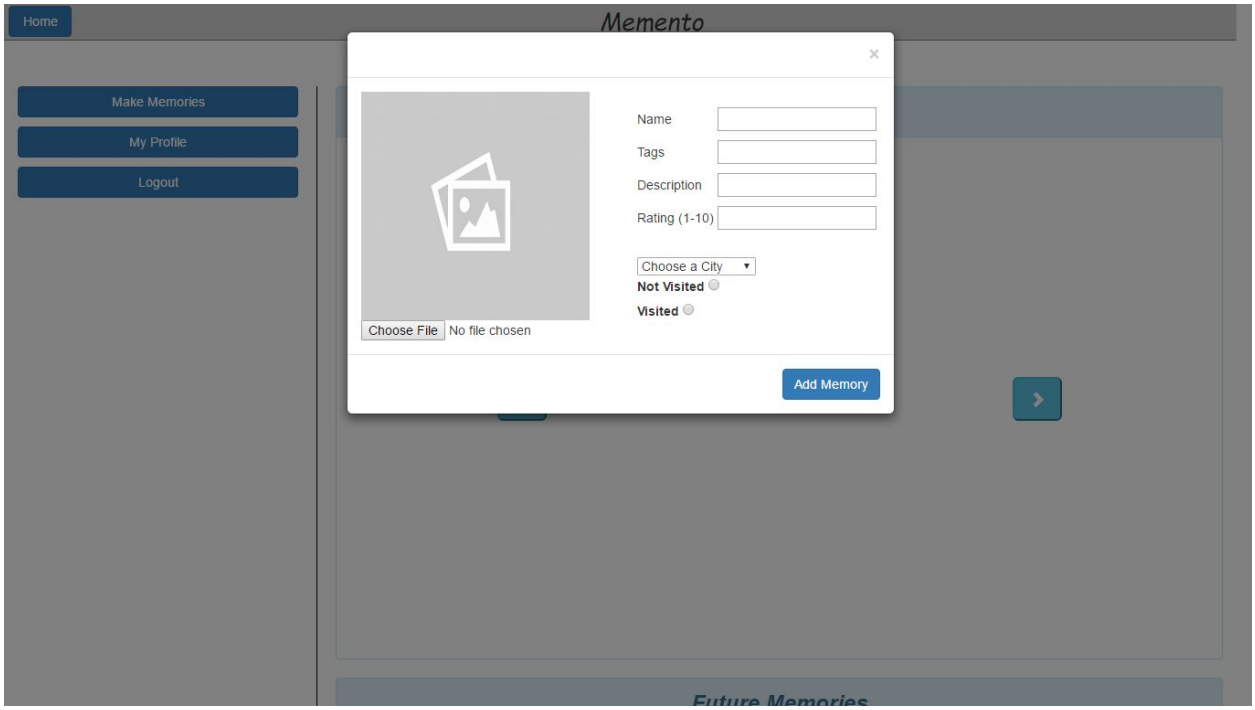
2) **Type in information and click Register**


**How to create a Memory**

1. **Click on 'Make Memories' on the home page**



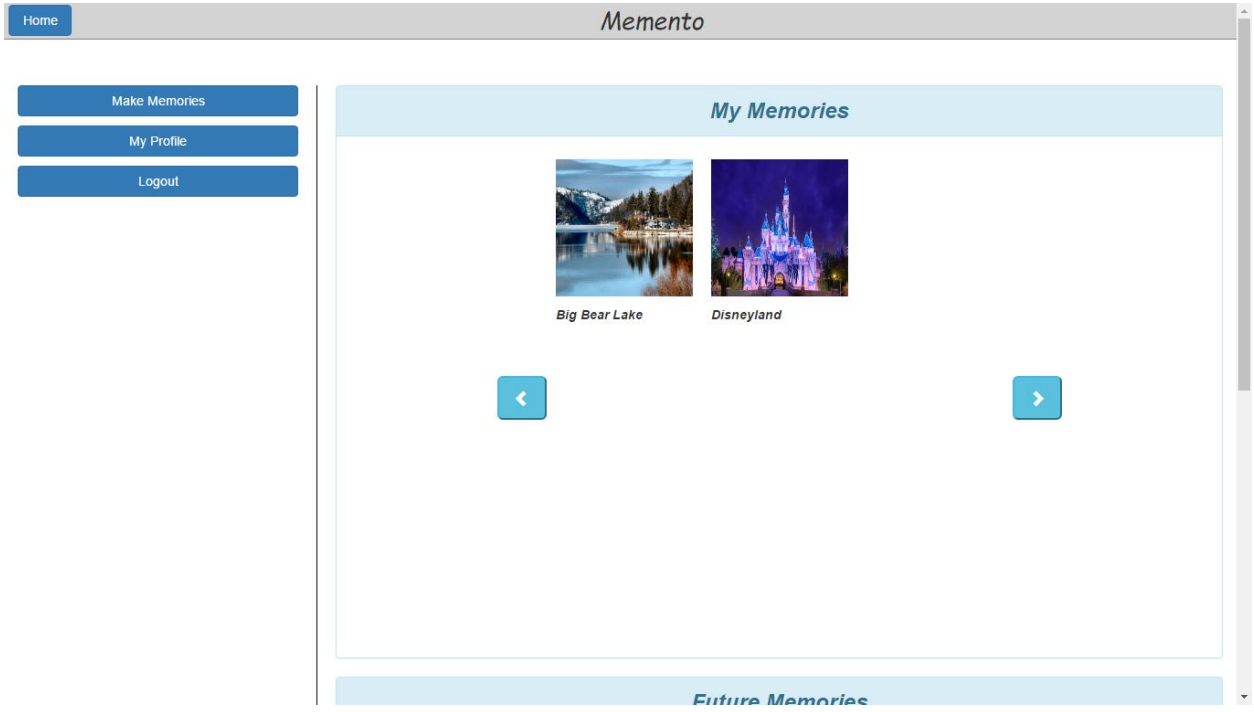
2. **Type in required information and click ‘Add Memory’**



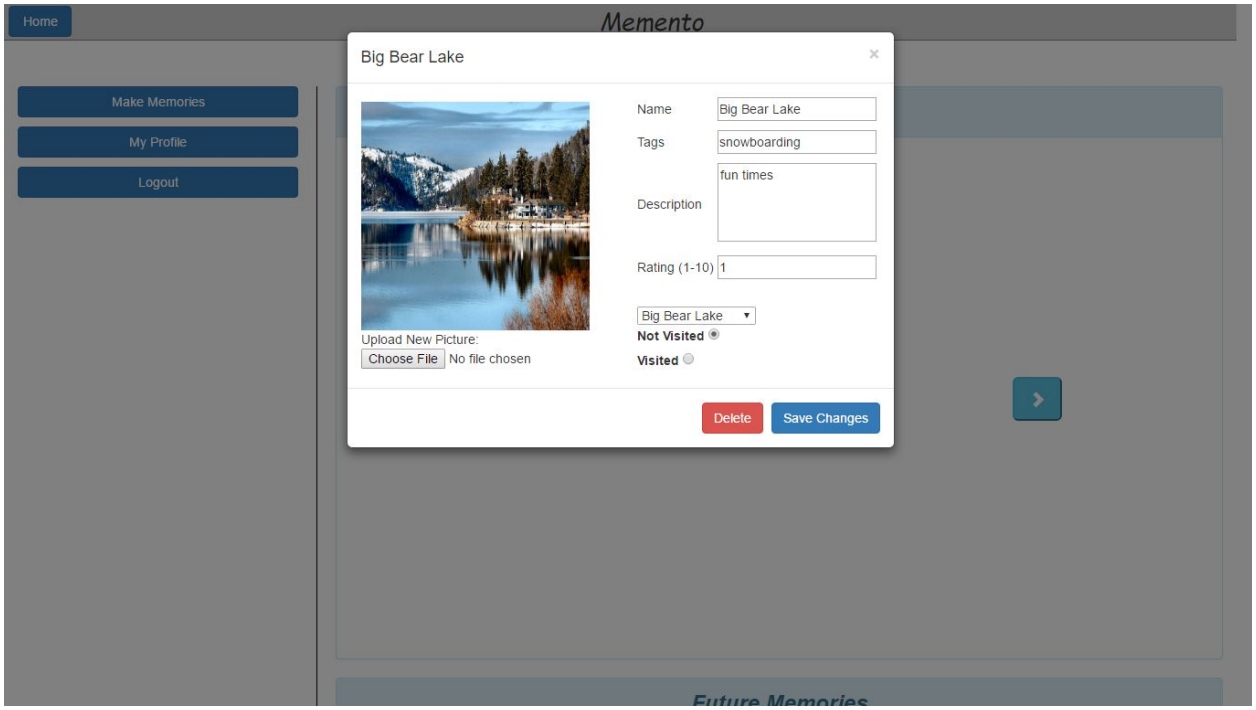

3. **The new memory will show up in ‘My Memories’ section**

## How to Edit a Memory

1. **Click on an existing memory via it’s image**



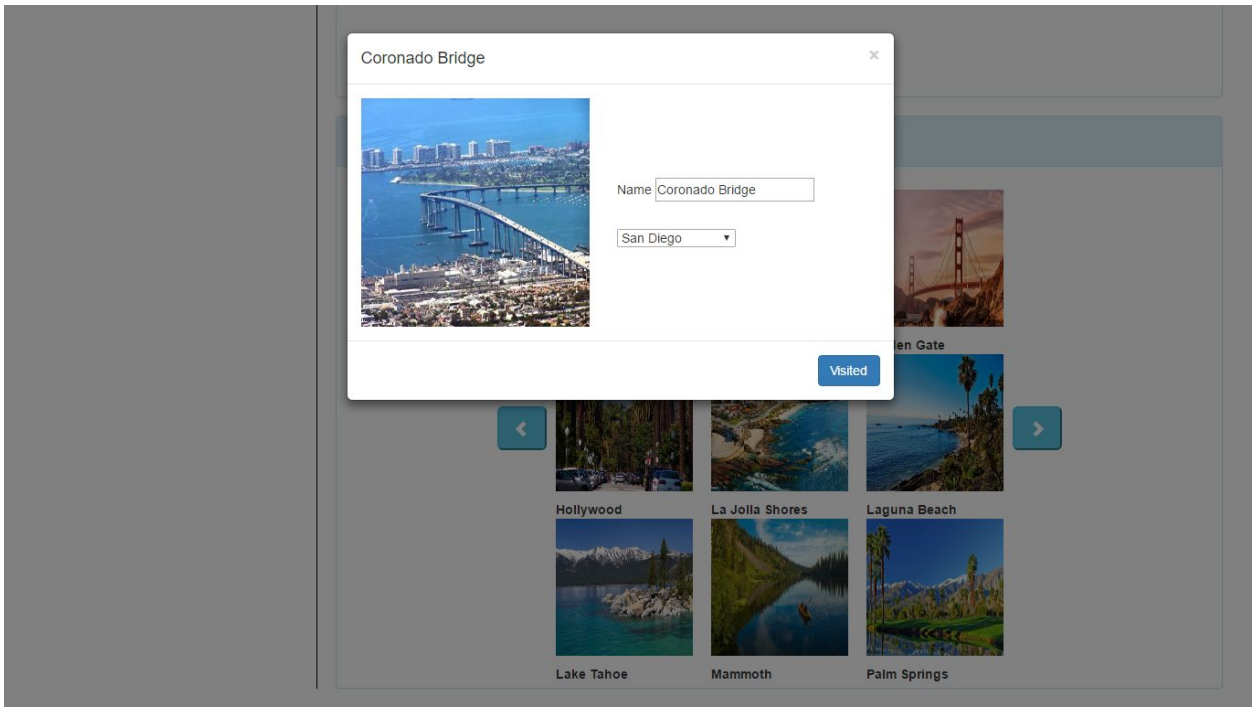
2. **Update the memories information and click ‘Save Changes’ to update information**




3. **The information will be updated for the user to see.**

**How to mark a memory as visited**

1. **Click on a memory in the ‘Future Memories’ section**



2. **If you have visited this place, then click ‘Visited’.**
3. **The memory will now be in the ‘My Memories’ section**

**ToDo**

**Even though we accomplished a good amount during these 10 weeks, we feel that there can still be improvements made. Specifically, we feel that users should be able to filter their memories by the tags so they can find specific ones easily. We would have also liked to allow users to input their own city upon creating a memory. We were unable to allow users to do this because we found it difficult to dynamically add entries to the database.**

**We also would like to improve on the User Experience of our app. We feel that a major factor in our app’s success would be it’s aesthetics because of the crowd that it caters to. A future improvement to be made is to create a color scheme and theme that incites wanderlust in the users. That way, they would be more likely travel and use our app. Another improvement we would have liked to make is to compress the images, to improve load speed. One drawback of allowing images is that it takes up a large amount of bandwidth to send. As such, users with poor connections will have difficulty using the app.**

**In our future endeavors, we will pay more attention to User Experience instead of Developer Experience because our apps are made for the user.**