Link to this slideshow

https://docs.google.com/presentation/d/1MHyVjvdY55M2Z7slP8u5uPMsvq9qQOab9_jUMaZsPng/edit?usp=sharing

The Project Vision

Project Name: Timeline

Link to Repo: https://github.com/liyu-hz168/Timeline
Link to Milestone tag: https://github.com/liyu-hz168/Timeline/tree/milestone-1
Link issues:https://github.com/liyu-hz168/Timeline/issues

Please run:

```
git clone
git checkout 541eeac
npm install
npm start
```

The Builders

Gerindra Adi:

- Main UI designer and UX point person
- Implementing timeline UI + scrolling animation + data filtering for different views
- Implement the thumbnail component to display preview of different memories

Leon Ge:

- Set up mock data
- Implement the memory modules component that will be displayed within memory card component

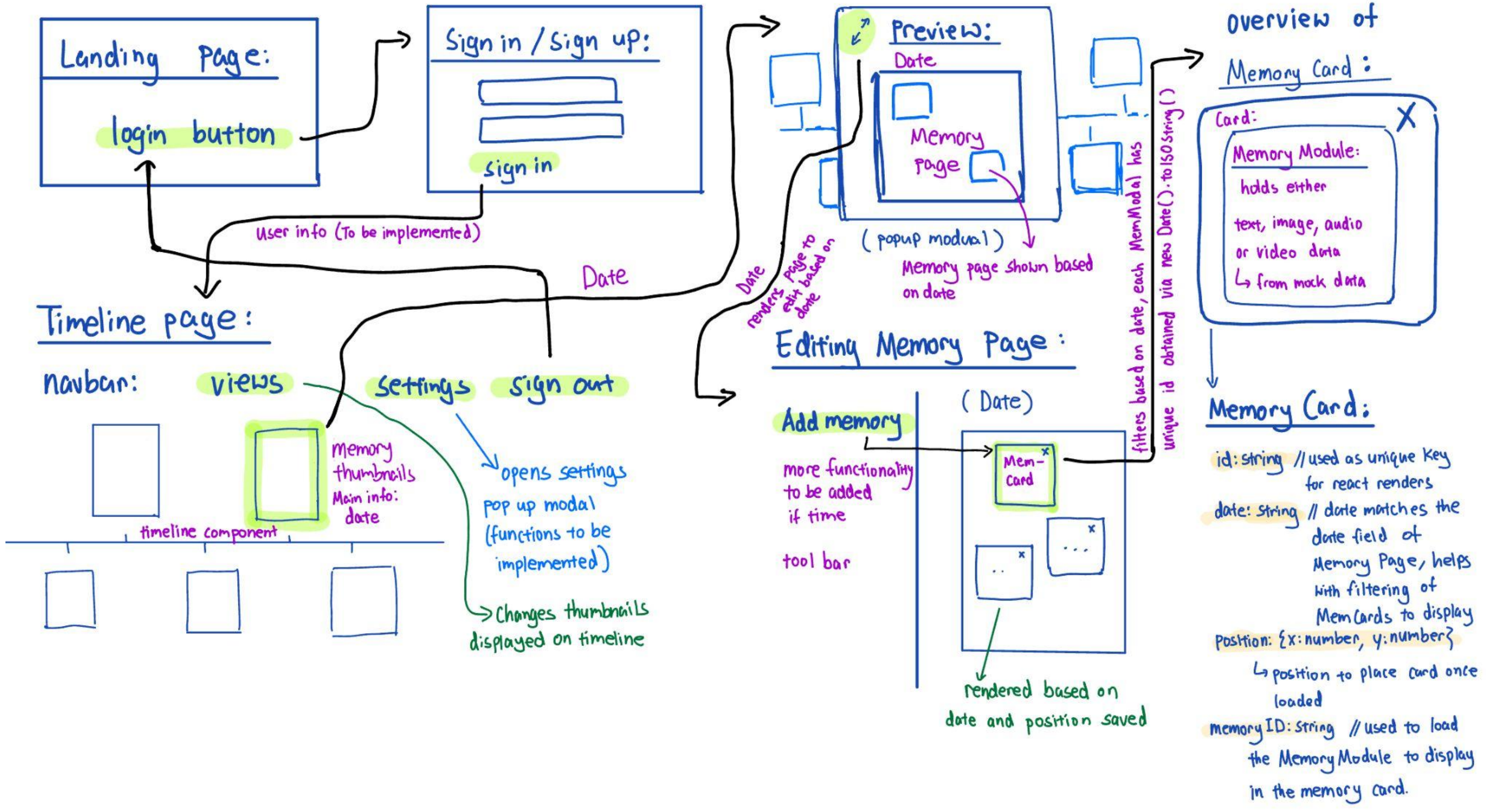
LiYu Zeng:

- Implementing draggable components (memory cards) on the memory page
- UI for navbar
- Basic routing

Selena Lo:

- Implement preview of memory page
- Implement land and sign in page

Software Architecture Overview


Landing Page:
login button
Sign in / Sign up:
sign in
User info (To be implemented)
Date
Timeline page:
navbar:
views
settings
sign out
memory thumbnails
Main info: date
timeline component
opens settings pop up modal (functions to be implemented)
Changes thumbnails displayed on timeline
Preview:
Date
Memory page
(popup modual)
Memory page shown based on date
Date
renders page to edit based on date
Editing Memory Page:
(Date)
Add memory
more functionality to be added if time
tool bar
Mem-Card
rendered based on date and position saved
filters based on date, each MemModal has unique id obtained via new Date().toISOString()
overview of Memory Card:
Card:
Memory Module:
holds either text, image, audio or video data
↳ from mock data
Memory Card:
id: string // used as unique key for react renders
date: string // date matches the date field of Memory Page, helps with filtering of MemCards to display
Position: {x: number, y: number}
↳ position to place card once loaded
memoryID: String // used to load the Memory Module to display in the memory card.

Software Architecture Overview-Continued

How state management is handled:

- Global context is being used to keep track of the state of all the memory cards (The individual components that renders on each memory page)
- Similarly global context is used to track whether or not the user is in editing mode. Editing mode allows the user to move around the memory cards & add/delete.
- Local state is being maintained by each memory card to keep track of their positions on the memory page. Updates as the user move around the cards

Historical Timeline

FEb 21st

Start of UI design/Initial wireframing

March 10th

UI Design completed, start of component development

March 31st

Timeline component developed, navbar component. (Issues: 1, 2, 3, 4)

April 1st

Edit memory page, memory cards component, memory preview, bug fixes (Issues: 5, 7, 18)

April 3rd

Mock data and memory module content (Issues: 8, 11)

April 4th

Landing page, Integrating all the components, final refinements(Issues: 10)

Design and Styling Guidelines

Typography:

- Modern, sleek design using a minimal of 3 typefaces
  Fonts: PP Editorial New (serif) for logo and navigation, Helvetica for body text
- **Considerations**: Avoid light font weights, use left/right alignment over justified text, and follow Apple's guidelines for readability.

Color Scheme:

- Monochromatic, high contrast color scheme

Layout:

- Minimalist with symmetrical and balanced visuals for ease of usage
- Uniform gaps between elements, rounded corners, and centered alignment where possible
- Main design philosophy: As simple as possible, intuitive and easy to navigate/use

Link: https://drive.google.com/file/d/1I7XDvslxdWCrniZfMXnbn7_weppndRVf/view?usp=sharing

Individual Team Member Contributions

Gerindra Adi - Assigned Work Summary

Assigned Issues:

- 2 - Implement the Timeline Page (Closed):
    - The crux of the entire project, I was tasked with creating the Timeline UI, complete with horizontal scrolling, dynamic resizing animations using GSAP, and the ability to switch between different weekly, monthly, and yearly views. The views were implemented via three different ways of filtering user data.
- 3 - Apply TailwindCSS to the Timeline Page (Closed):
    - In order to keep the design consistent with the UI/UX standardization document, I was to use Tailwind CSS to apply stylings to the timeline page. I made sure to carefully use my Figma document to ensure the stylings were accurate to the brief.
- 4 - Create Timeline Thumbnail Component (Closed):
    - On the timeline, there sits many thumbnail components that each correspond to a memory page. These thumbnail components needed to be reusable, as many of them need to be loaded based off the user data. I made the thumbnails so that they took the appropriate user data as props and sat on the timeline in a uniform way.
- 24 - Implement data filtering function for the weekly, monthly, and yearly views (closed):
    - To implement these views, i created three different functions that filtered the mock data based on month, week, and year, using Javascript's native date library to do so.

All Closed Pull Requests:

- https://github.com/liyu-hz168/Timeline/pull/19 - Merged my branch with the complete timeline page with the branch with the completed Navbar page
- https://github.com/liyu-hz168/Timeline/pull/21 - Merged my branch with changes from main to be able to work with the mock data provided by Leon
- https://github.com/liyu-hz168/Timeline/pull/23 - Merged my branch to main, as I got the timeline to work with the navbar and with the mock data

Gerindra Adi - Assigned Work Summary

Notable Commits:

- Completed the Timeline with horizontal scrolling and gaussian growth-curve based dynamic styling:
  - https://github.com/liyu-hz168/Timeline/commit/b6e45967830a3c451cc00524d43d77568662b672
- Ensured the Timeline bar scaled responsively to the width of the screen:
  - https://github.com/liyu-hz168/Timeline/commit/d606b9cdaae5818156ecb843bbad3fb01ec3c7e5
- Created and implemented the reusable thumbnail component:
  - https://github.com/liyu-hz168/Timeline/commit/35c0c79d260fb351fddd5200cd7e8f1ff54ea25c
- Fixed text overflow issue on thumbnails:
  - https://github.com/liyu-hz168/Timeline/commit/e6624f313eca36cd4859ae69ac441f0047cefa7a
- Added UI/UX Standardization File:
  - https://github.com/liyu-hz168/Timeline/commit/9ada8d46da3012b32b83c83e764f03eecb4dbc98
- Created and integrated timeline view switching functions on the timeline itself\
  - https://github.com/liyu-hz168/Timeline/commit/a5940a06e7ba9d8808e759e88c33d4abe99a6653

Gerindra Adi - Screenshots and Demos

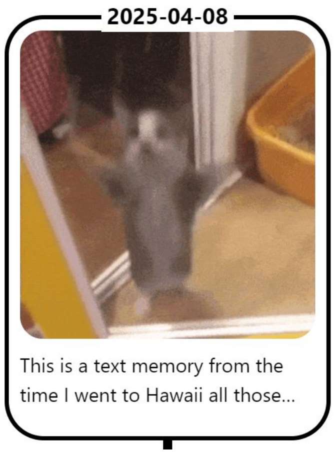
Thumbnail Component:

- Contains a single image and text from a specific memory card stored in the mock data
- Has the date from the day on top
- Has smooth rounded edges
- Text that overflows is cut off from the bottom and clamped with ellipses. The clamplines is dynamically calculated based off of the height of the card

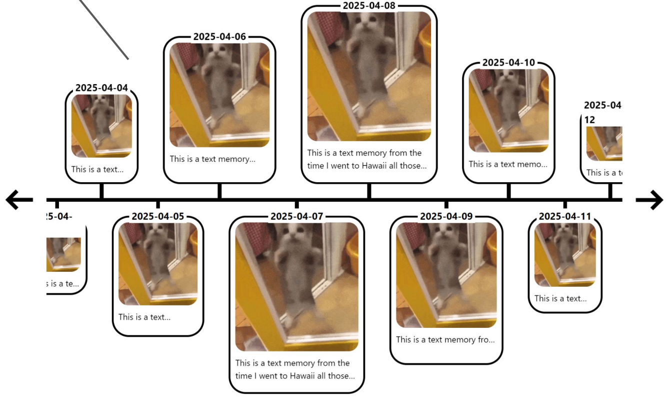
Timeline Component:

- Very complex component that utilizes two scrollable galleries at the top and bottom that scroll horizontally in unison when either is scrolled
- When scrolled, the items in the edges shrink and the items closer to the center grow
- These animations are made smoother by gsaps tweening capabilities, decelerating and accelerating smoothly
- The left and right buttons are shortcuts to scroll left and right at half the width of the timeline. It can be spammed multiple times to go fast. Brrr.

Thumbnail Component


2025-04-08
This is a text memory from the time I went to Hawaii all those...

Timeline Component

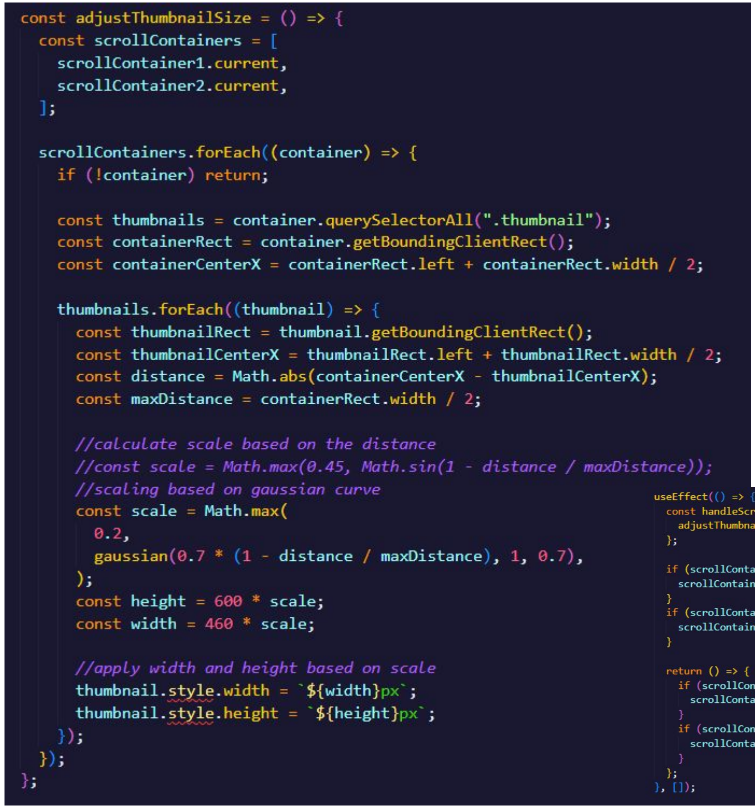


Gerindra Adi - Component Interaction

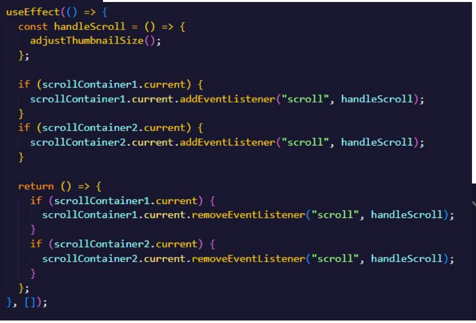
How my timeline interacts with other components is highlighted in the software architecture overview. But here is a simple explanation. There is user data that stores data from each memory. This user data is parsed by filtering functions, then sent to the timeline component as an array of thumbnail objects. These thumbnail objects are then rendered onto the timeline as thumbnail components. When a thumbnail is clicked, the corresponding memory page is previewed and opened. When the week, month, or year view is clicked on from the navbar, the timeline rerenders to show the cards matching that view.

Gerindra Adi - Code and UI Explanation

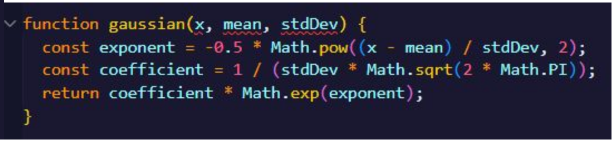
Key Piece of Code:
This piece of code is responsible for resizing the thumbnails the closer they are to the center of the page. ContainerCenterX is the center of the timeline, calculated using the getBoundingClientRect function. Then, for each thumbnail, we calculate the distance away it is from the center, and use a gaussian scaling function determine a scale at which we would transform the thumbnail’s dimensions. I had determined the minimum scale to be 0.2. Then it sets these thumbnails’ dimensions to its new dimensions. I then had it run on a scroll event listener so it would update every time the user scrolled.



```
const adjustThumbnailSize = () => {
  const scrollContainers = [
    scrollContainer1.current,
    scrollContainer2.current,
  ];

  scrollContainers.forEach((container) => {
    if (!container) return;

    const thumbnails = container.querySelectorAll(".thumbnail");
    const containerRect = container.getBoundingClientRect();
    const containerCenterX = containerRect.left + containerRect.width / 2;

    thumbnails.forEach((thumbnail) => {
      const thumbnailRect = thumbnail.getBoundingClientRect();
      const thumbnailCenterX = thumbnailRect.left + thumbnailRect.width / 2;
      const distance = Math.abs(containerCenterX - thumbnailCenterX);
      const maxDistance = containerRect.width / 2;

      //calculate scale based on the distance
      //const scale = Math.max(0.45, Math.sin(1 - distance / maxDistance));
      //scaling based on gaussian curve
      const scale = Math.max(
        0.2,
        gaussian(0.7 * (1 - distance / maxDistance), 1, 0.7),
      );
      const height = 600 * scale;
      const width = 460 * scale;

      //apply width and height based on scale
      thumbnail.style.width = `${width}px`;
      thumbnail.style.height = `${height}px`;
    });
  });
};
```



```
useEffect(() => {
  const handleScroll = () => {
    adjustThumbnailSize();
  };

  if (scrollContainer1.current) {
    scrollContainer1.current.addEventListener("scroll", handleScroll);
  }
  if (scrollContainer2.current) {
    scrollContainer2.current.addEventListener("scroll", handleScroll);
  }

  return () => {
    if (scrollContainer1.current) {
      scrollContainer1.current.removeEventListener("scroll", handleScroll);
    }
    if (scrollContainer2.current) {
      scrollContainer2.current.removeEventListener("scroll", handleScroll);
    }
  };
}, []);
```



```
function gaussian(x, mean, stdDev) {
  const exponent = -0.5 * Math.pow((x - mean) / stdDev, 2);
  const coefficient = 1 / (stdDev * Math.sqrt(2 * Math.PI));
  return coefficient * Math.exp(exponent);
}
```

Gerindra Adi - Challenges and Insights

The biggest challenge in this milestones were communication, time management, and - on a technical level - getting more familiar with React and some of its libraries. We held weekly meetings to ensure smooth operations, but sometimes they would not prove to be as productive as we would have liked. Additionally, some of our assignments were based on the completion of other members' assignments, which made it tough to progress sometimes when there were bottlenecks. This is no fault of anyone in particular, but it must inform how we sequence, space out, and delegate assignments in the future. In the future, we need to create agendas for each sprint before we start them so they are more productive.

Additionally, this was only my second time making a significant project in React, so it was a big challenge getting used to all of its idiosyncrasies. However, the biggest challenge I had was to make the Timeline work in an aesthetically pleasing way, and I learned the importance of starting small, prototyping, and iterating over and over until I can get a model to work. I worked my way up with the timeline. Initially, just trying to get one row of black squares to work, then two rows to work, then resizing them based on distance and so on and so forth.

In later sprints, I would like to add a feature where the user can quickly switch between weeks, months, and years using a little dropdown menu in the navbar, like in google calendars. Additionally, I would like to optimize the scrolling animation as it is quite sluggish right now and could benefit from some refactoring. In later sprints, I think we need to rethink how we are storing user data, as I think it is too convoluted right now and could use some simplifying.

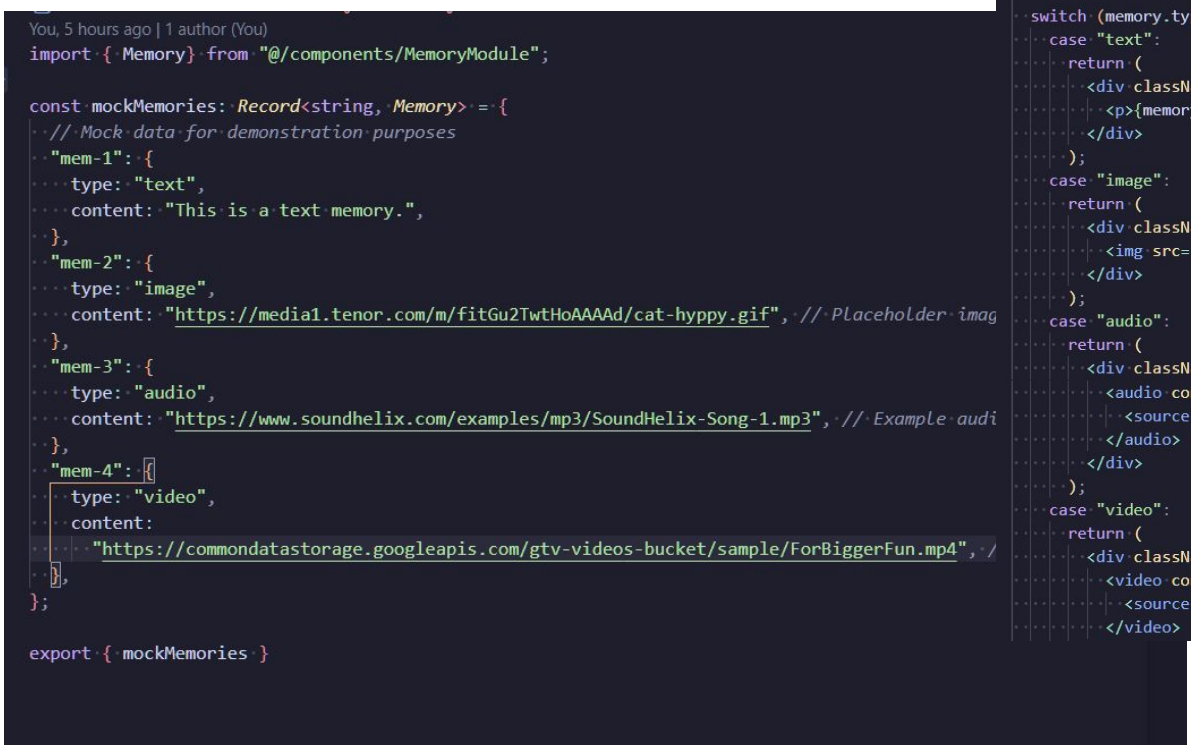
Leon Ge

My personal part in this milestone involves writing the “memory module” component, which is integrated in the “memory card” component.

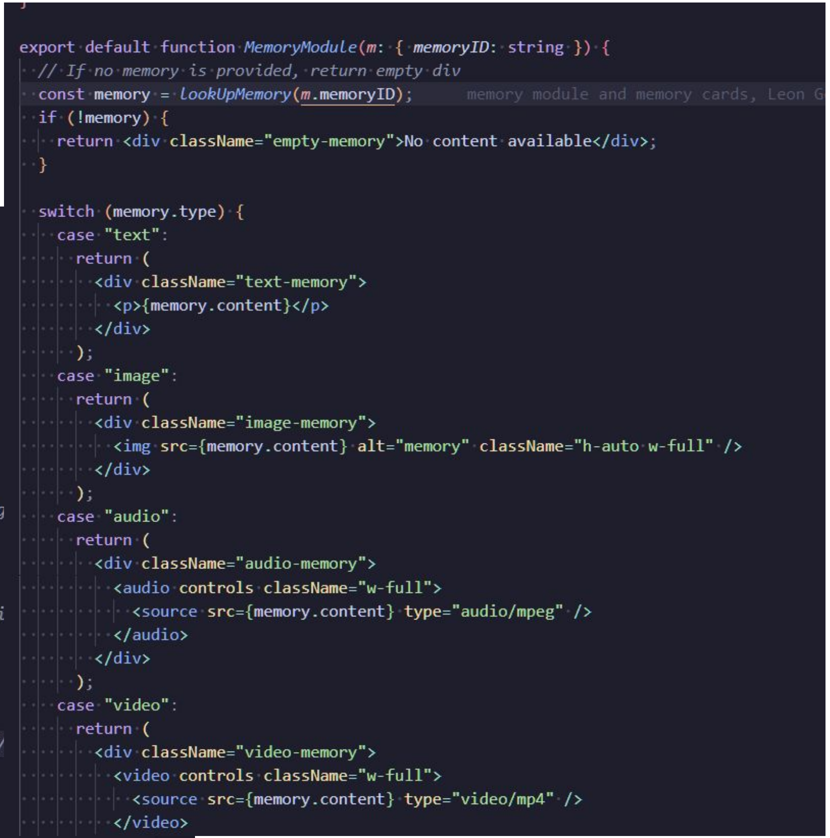
The component takes in a “memory id” as a string prop to send a asynchronous request to a backend (when built) to request a specific memory object, which then uses to render using different html elements based on the type of the memory.

I was assigned to issue 8, 11, however needed to divide my feature due to workload reasons.

Leon



```
You, 5 hours ago | 1 author (You)
import { Memory} from "@/components/MemoryModule";

const mockMemories: Record<string, Memory> = {
  // Mock data for demonstration purposes
  "mem-1": {
    type: "text",
    content: "This is a text memory.",
  },
  "mem-2": {
    type: "image",
    content: "https://media1.tenor.com/m/fitGu2TwtHoAAAAd/cat-hyppy.gif", // Placeholder imag
  },
  "mem-3": {
    type: "audio",
    content: "https://www.soundhelix.com/examples/mp3/SoundHelix-Song-1.mp3", // Example audi
  },
  "mem-4": {
    type: "video",
    content:
      "https://commondatastorage.googleapis.com/gtv-videos-bucket/sample/ForBiggerFun.mp4", /
  },
};

export { mockMemories }
```

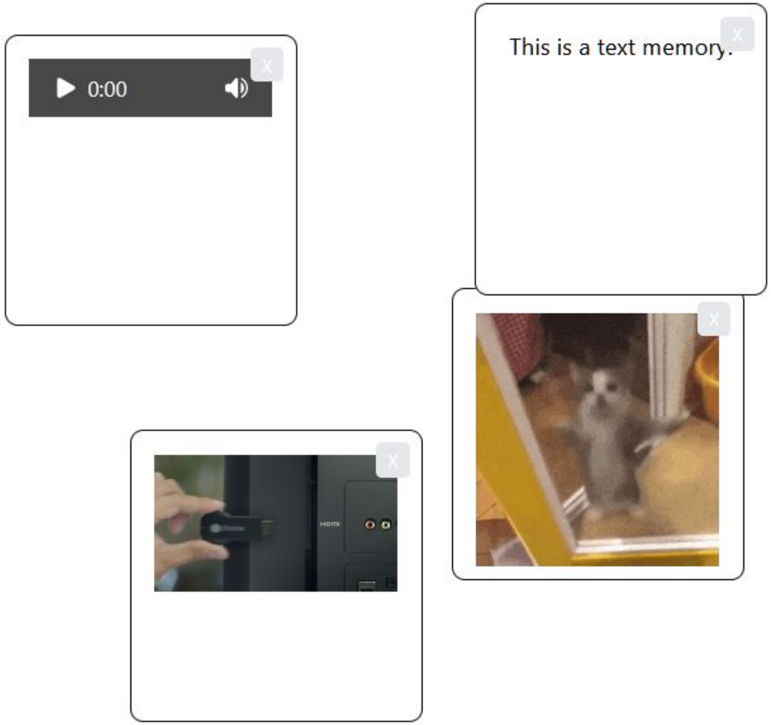


```
export default function MemoryModule(m: { memoryID: string }) {
  // If no memory is provided, return empty div
  const memory = LookUpMemory(m.memoryID);      memory module and memory cards, Leon G
  if (!memory) {
    return <div className="empty-memory">No content available</div>;
  }

  switch (memory.type) {
    case "text":
      return (
        <div className="text-memory">
          <p>{memory.content}</p>
        </div>
      );
    case "image":
      return (
        <div className="image-memory">
          <img src={memory.content} alt="memory" className="h-auto w-full" />
        </div>
      );
    case "audio":
      return (
        <div className="audio-memory">
          <audio controls className="w-full">
            <source src={memory.content} type="audio/mpeg" />
          </audio>
        </div>
      );
    case "video":
      return (
        <div className="video-memory">
          <video controls className="w-full">
            <source src={memory.content} type="video/mp4" />
          </video>
```

Leon

This is the memory cards page, my teammate contributed to the outer card frame, while I contributed to the inner react component that renders the actual content.

Because this component simply renders the element with html tags, it can be reused elsewhere in the code



This is a text memory.

Leon

Challenges:
It was difficult to design the data struct to represent each card and memory, thus I needed to come up with a redesign.

Future improvements:
Use a backend server to handle requests, and also better fill in the memory card component with the inner component

LiYu Zeng

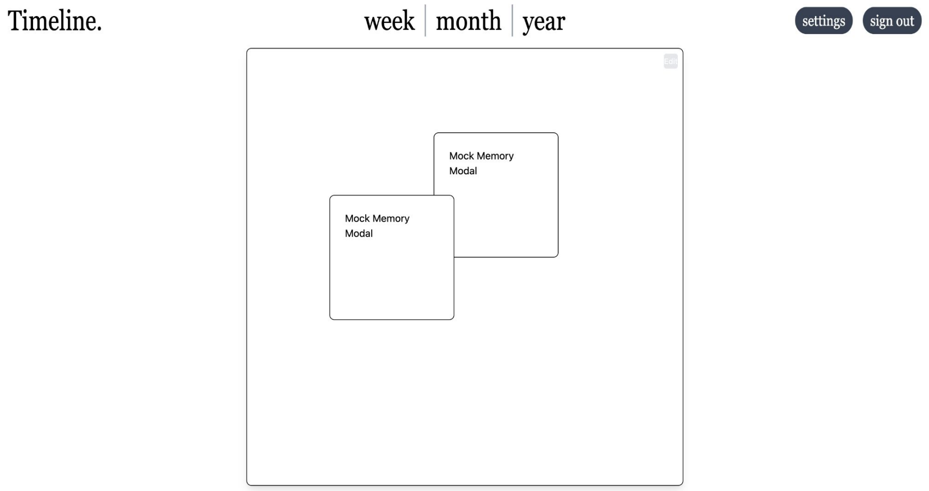
Assigned work summary:

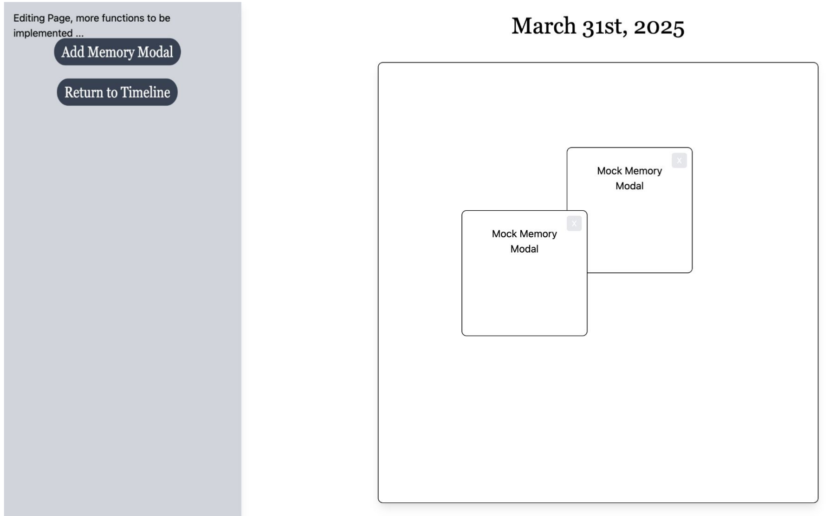
- Issues: 1, 6, 7, main commits in the memory-page branch
- Main tasks: I created the Navbar component and set up the Memory Page in both edit and preview modes. I implemented the logic for draggable memory cards on the Memory Page, along with the add and delete functionalities for the memory cards. Additionally, I handled the routing between different pages.
- Issues not closed: https://github.com/liyu-hz168/Timeline/issues/16

LiYu Zeng - Continued

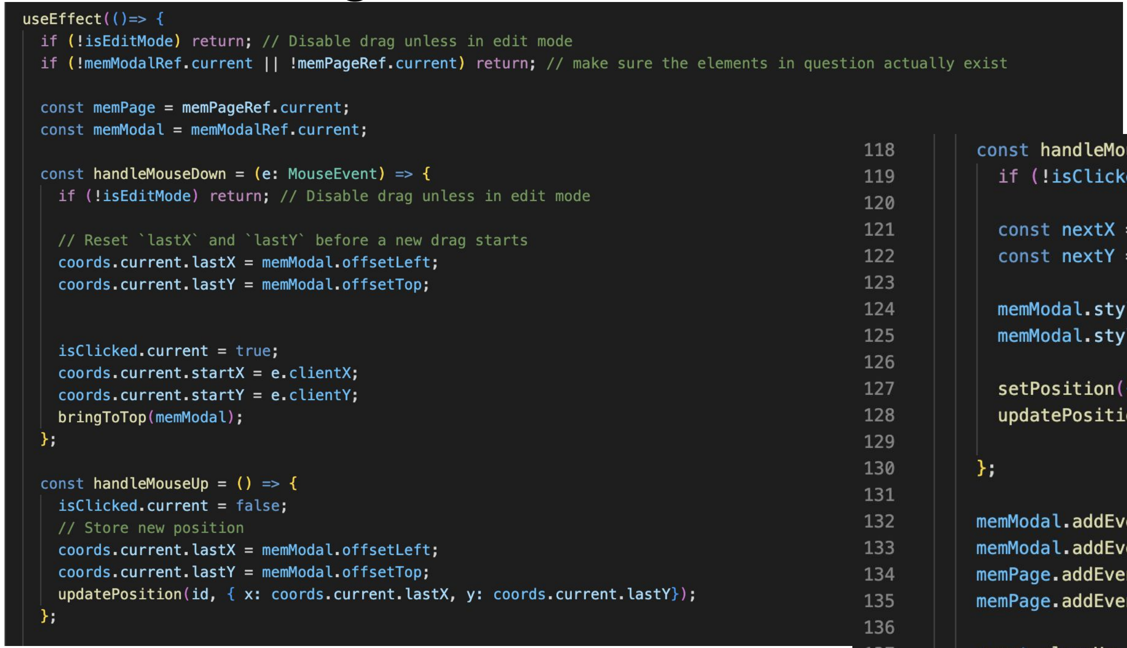
Screenshots: (Screenshots from before integrating with the rest of the application so it might look a little different)

- When memory page not in editing mode, no memory card components can be moved around and no delete and add can happen.
- In edit mode (2nd image), the memory cards are draggable and add/delete allowed

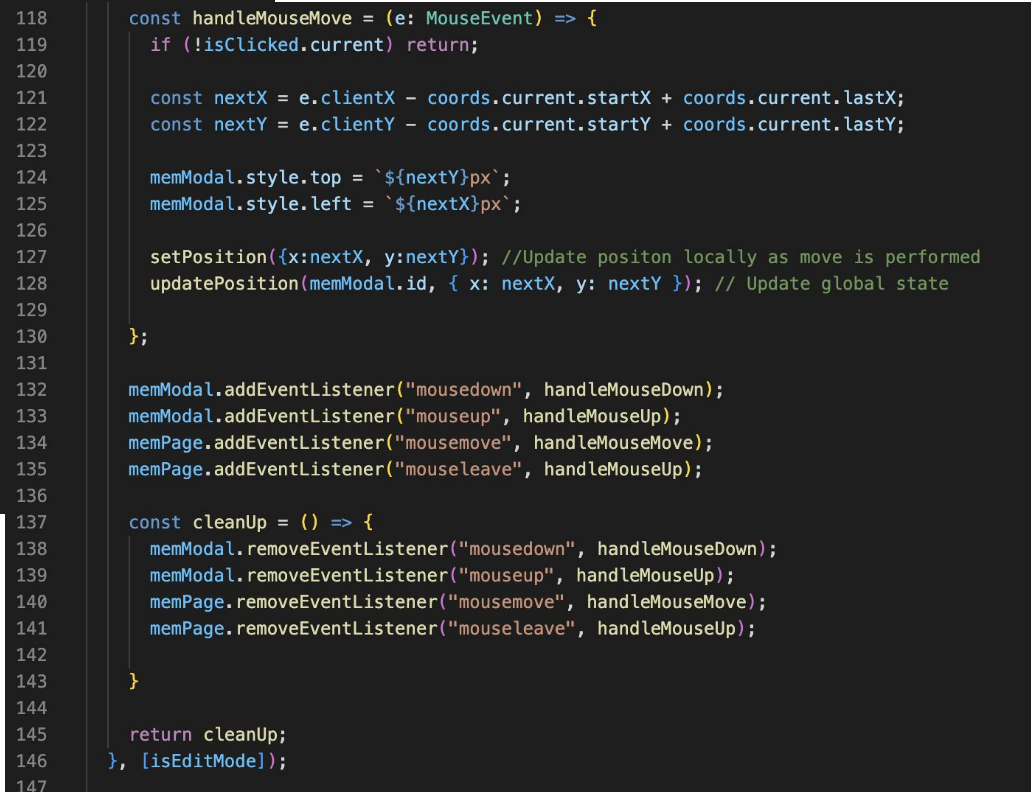



Editing Page, more functions to be implemented ...

LiYu Zeng - Continued



```
useEffect(()=> {
  if (!isEditMode) return; // Disable drag unless in edit mode
  if (!memModalRef.current || !memPageRef.current) return; // make sure the elements in question actually exist

  const memPage = memPageRef.current;
  const memModal = memModalRef.current;

  const handleMouseDown = (e: MouseEvent) => {
    if (!isEditMode) return; // Disable drag unless in edit mode

    // Reset `lastX` and `lastY` before a new drag starts
    coords.current.lastX = memModal.offsetLeft;
    coords.current.lastY = memModal.offsetTop;


    isClicked.current = true;
    coords.current.startX = e.clientX;
    coords.current.startY = e.clientY;
    bringToTop(memModal);
  };

  const handleMouseUp = () => {
    isClicked.current = false;
    // Store new position
    coords.current.lastX = memModal.offsetLeft;
    coords.current.lastY = memModal.offsetTop;
    updatePosition(id, { x: coords.current.lastX, y: coords.current.lastY});
  };
```



```
  const handleMouseMove = (e: MouseEvent) => {
    if (!isClicked.current) return;

    const nextX = e.clientX - coords.current.startX + coords.current.lastX;
    const nextY = e.clientY - coords.current.startY + coords.current.lastY;

    memModal.style.top = `${nextY}px`;
    memModal.style.left = `${nextX}px`;

    setPosition({x:nextX, y:nextY}); //Update positon locally as move is performed
    updatePosition(memModal.id, { x: nextX, y: nextY }); // Update global state

  };

  memModal.addEventListener("mousedown", handleMouseDown);
  memModal.addEventListener("mouseup", handleMouseUp);
  memPage.addEventListener("mousemove", handleMouseMove);
  memPage.addEventListener("mouseleave", handleMouseUp);

  const cleanUp = () => {
    memModal.removeEventListener("mousedown", handleMouseDown);
    memModal.removeEventListener("mouseup", handleMouseUp);
    memPage.removeEventListener("mousemove", handleMouseMove);
    memPage.removeEventListener("mouseleave", handleMouseUp);

  }

  return cleanUp;
}, [isEditMode]);
```

LiYu Zeng - Continued

Code and UI Explanation:

This code implements the core logic behind the draggable functionality. While it doesn't directly affect the UI beyond the basic design of the memory cards (as shown on the previous slide), it is essential to enabling a key feature of the application.

The main challenge was figuring out how to properly handle the mouse events and the math involved in moving the components. Initially, my components would move off-screen, away from the mouse. The solution was to refine my formulas and ensure I was using the correct variables in the calculations to keep the components within the viewable area.

LiYu Zeng - Continued

Component Hierarchy & Interaction:

- My work mainly fits into the memory pages and the edit memory page functionality of our application.
- Timeline thumbnail -> Memory page preview ->click edit -> edit memory page

Challenges and Insights:

- The main challenge I faced was understanding the math behind the draggable components and debugging the functionality to ensure smooth interactions. Additionally, I needed to come up with a basic data structure that could be easily built upon by my teammates as they continued the development process. Personally, I’m not fond of planning, so this project became a valuable learning experience for me. Working in a team highlighted the importance of communication and collaboration, as it allowed us to share insights, resolve issues more efficiently, and align on our goals.

Future Improvements & Next Steps:

- The next steps involve enabling users to add different types of data into the memory cards for display on the memory page. Currently, the add functionality is hardcoded to return an empty memory card, so we will need to implement the necessary features to allow users to input and store relevant data.
- Additionally, the way we are rendering the memory thumbnails on the timeline can be optimized for better performance and responsiveness. We should also explore more efficient ways to store and manage the memory data within the React state, ensuring scalability and reducing potential rendering issues as the application grows.

Selena Lo

Assigned work summary:

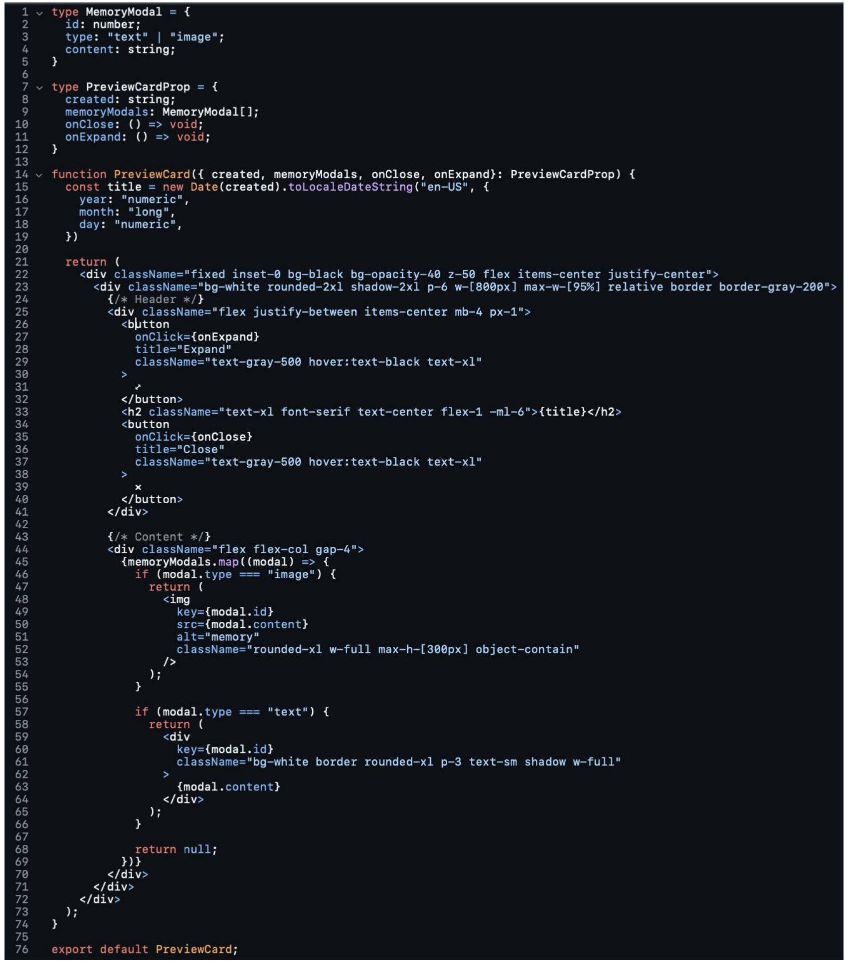
- **Issues:** #5, #10 – main commits in the auth-pages and timeline-preview branches

  Main tasks: I created the Sign-Up and Log-In pages, which serve as the gateway to the memory app, ensuring a secure and user-friendly experience for accessing personal memories. I also implemented the Timeline Preview screens, designed to offer users an intuitive and visually engaging way to browse and expand their saved memories. Each memory is represented as a card that displays a brief snippet of the content—clicking a card expands it into a larger preview while maintaining the user’s position within the timeline for seamless navigation.

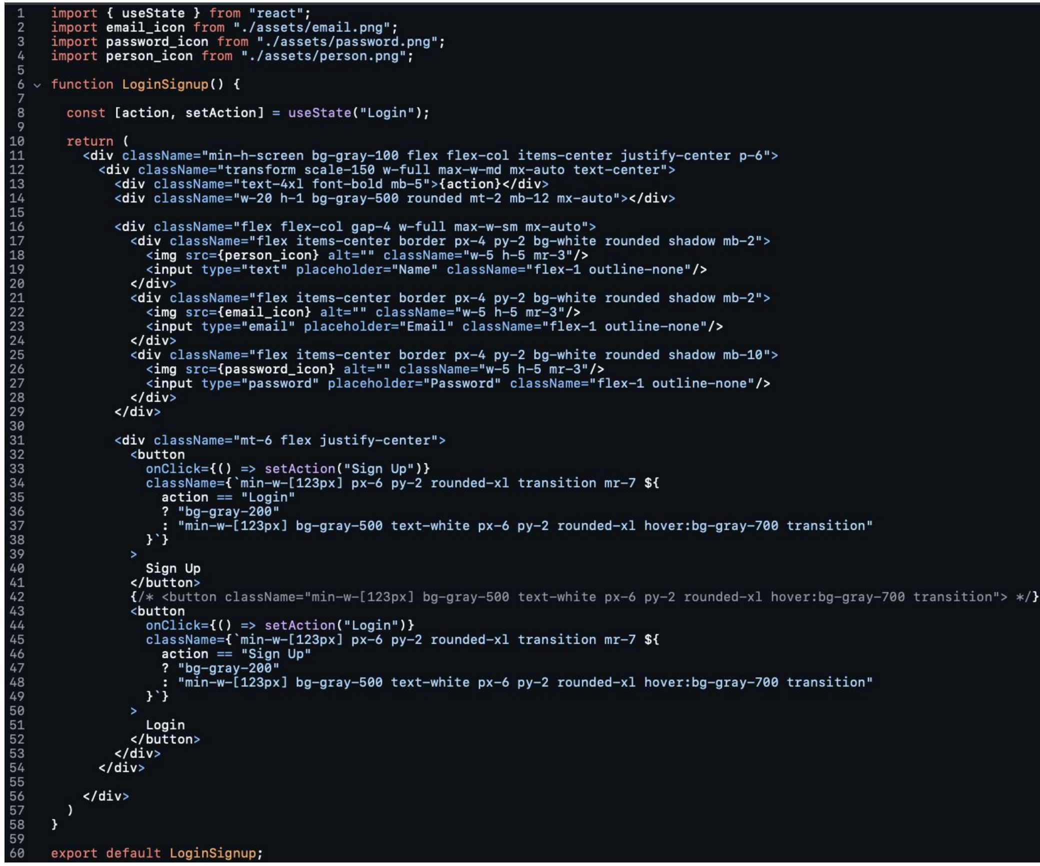
- **Issues not closed:**

  None – all assigned issues have been completed and closed.

Selena Lo - Continued

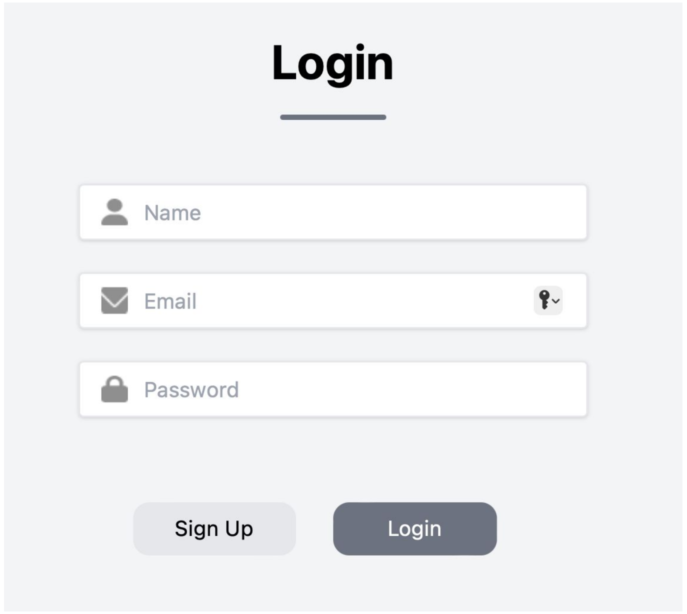


```tsx
type MemoryModal = {
  id: number;
  type: "text" | "image";
  content: string;
}

type PreviewCardProp = {
  created: string;
  memoryModals: MemoryModal[];
  onClose: () => void;
  onExpand: () => void;
}

function PreviewCard({ created, memoryModals, onClose, onExpand}: PreviewCardProp) {
  const title = new Date(created).toLocaleDateString("en-US", {
    year: "numeric",
    month: "long",
    day: "numeric",
  })

  return (
    <div className="fixed inset-0 bg-black bg-opacity-40 z-50 flex items-center justify-center">
      <div className="bg-white rounded-2xl shadow-2xl p-6 w-[800px] max-w-[95%] relative border border-gray-200">
        {/* Header */}
        <div className="flex justify-between items-center mb-4 px-1">
          <button
            onClick={onExpand}
            title="Expand"
            className="text-gray-500 hover:text-black text-xl"
          >
            ⤢
          </button>
          <h2 className="text-xl font-serif text-center flex-1 -ml-6">{title}</h2>
          <button
            onClick={onClose}
            title="Close"
            className="text-gray-500 hover:text-black text-xl"
          >
            ×
          </button>
        </div>

        {/* Content */}
        <div className="flex flex-col gap-4">
          {memoryModals.map((modal) => {
            if (modal.type === "image") {
              return (
                <img
                  key={modal.id}
                  src={modal.content}
                  alt="memory"
                  className="rounded-xl w-full max-h-[300px] object-contain"
                />
              );
            }

            if (modal.type === "text") {
              return (
                <div
                  key={modal.id}
                  className="bg-white border rounded-xl p-3 text-sm shadow w-full"
                >
                  {modal.content}
                </div>
              );
            }

            return null;
          })}
        </div>
      </div>
    </div>
  );
}

export default PreviewCard;
```



```jsx
import { useState } from "react";
import email_icon from "./assets/email.png";
import password_icon from "./assets/password.png";
import person_icon from "./assets/person.png";

function LoginSignup() {

  const [action, setAction] = useState("Login");

  return (
    <div className="min-h-screen bg-gray-100 flex flex-col items-center justify-center p-6">
      <div className="transform scale-150 w-full max-w-md mx-auto text-center">
        <div className="text-4xl font-bold mb-5">{action}</div>
        <div className="w-20 h-1 bg-gray-500 rounded mt-2 mb-12 mx-auto"></div>

        <div className="flex flex-col gap-4 w-full max-w-sm mx-auto">
          <div className="flex items-center border px-4 py-2 bg-white rounded shadow mb-2">
            <img src={person_icon} alt="" className="w-5 h-5 mr-3"/>
            <input type="text" placeholder="Name" className="flex-1 outline-none"/>
          </div>
          <div className="flex items-center border px-4 py-2 bg-white rounded shadow mb-2">
            <img src={email_icon} alt="" className="w-5 h-5 mr-3"/>
            <input type="email" placeholder="Email" className="flex-1 outline-none"/>
          </div>
          <div className="flex items-center border px-4 py-2 bg-white rounded shadow mb-10">
            <img src={password_icon} alt="" className="w-5 h-5 mr-3"/>
            <input type="password" placeholder="Password" className="flex-1 outline-none"/>
          </div>
        </div>

        <div className="mt-6 flex justify-center">
          <button
            onClick={() => setAction("Sign Up")}
            className={`min-w-[123px] px-6 py-2 rounded-xl transition mr-7 ${
              action == "Login"
              ? "bg-gray-200"
              : "min-w-[123px] bg-gray-500 text-white px-6 py-2 rounded-xl hover:bg-gray-700 transition"
            }`}
          >
            Sign Up
          </button>
          {/* <button className="min-w-[123px] bg-gray-500 text-white px-6 py-2 rounded-xl hover:bg-gray-700 transition"> */}
          <button
            onClick={() => setAction("Login")}
            className={`min-w-[123px] px-6 py-2 rounded-xl transition mr-7 ${
              action == "Sign Up"
              ? "bg-gray-200"
              : "min-w-[123px] bg-gray-500 text-white px-6 py-2 rounded-xl hover:bg-gray-700 transition"
            }`}
          >
            Login
          </button>
        </div>
      </div>

    </div>
  )
}

export default LoginSignup;
```

Selena Lo



Selena Lo - Continued

Code and UI Explanation:

The Sign-Up and Log-In pages were built using React and form handling logic to ensure smooth user input and routing. The UI focuses on clarity and ease of use, setting the tone for a welcoming first interaction with the app. The main challenge here was managing form state and navigation while keeping the layout responsive and clean.

For the Timeline Preview screens, the goal was to provide a visually appealing way for users to browse their saved memories. Each memory is represented as a compact card, which can be clicked to expand into a larger preview, all within the same timeline view. Implementing this required careful component structuring and conditional rendering. A key challenge was making sure expanded views didn't disrupt the overall timeline layout, which was solved by fine-tuning the CSS and ensuring state updates didn’t trigger unnecessary re-renders.

Selena Lo - Continued

Component Hierarchy & Interaction:

- My work primarily fits into the user authentication flow and the timeline viewing experience.
- App landing page → Sign-Up / Log-In pages → App
- App → Timeline Page → Timeline Preview cards → Expand memory card view

Challenges and Insights:

- One of the main challenges I encountered was creating a smooth user experience for the Sign-Up and Log-In forms while ensuring form validation. I had to manage form state cleanly and account for edge cases like incomplete inputs and basic client-side validation.
- On the preview cards side, the challenge was designing a layout that was both visually engaging and intuitive. Making sure that the memory cards show without breaking the timeline view required careful use of state and conditional rendering. This project taught me a lot about balancing UI/UX design with clean component structure. Collaborating with the team also emphasized the importance of consistent styling and reusable components.

Future Improvements & Next Steps:

- The next steps involve enabling users to add different types of data into the memory cards for display on the memory page. Currently, we only have mock data for modules of texts and images. Enhancing this functionality to support user-generated input will make the timeline more dynamic and meaningful.
- The way we are rendering the memory thumbnails on the timeline can be optimized for better performance and responsiveness. We should also explore more efficient ways to store and manage the memory data within the React state, ensuring scalability and reducing potential rendering issues as the application grows.