Project Report - Stage 4

Team BEAT

1. Project Changes Since Proposal

The goal of this project is to develop a web application that enables users to explore historical and academic events dynamically through an interactive timeline. Users can search for any topic of interest, such as “mathematics” or “computer science,” and generate a customized timeline showcasing major events in that field. From there, users can further refine their query by searching and filtering accordingly.

Additionally, the platform allows users to create and personalize their own timelines based on academic topics or personal life events. This tool serves as a research or educational aid, providing insights into how specific fields have evolved over time and how different topics intersect. The system aggregates various data sources to create pretty-looking timelines.

We did not change the idea of the application or its intended functionality. However, there are minor changes in the execution and final result, which will be detailed in the later section.

2. Achievement

Usefulness

The application is useful for researchers, students, and history enthusiasts by offering an intuitive way to explore the evolution of topics over time. Unlike static databases, our application is dynamic, allowing users to generate topic-based timelines on demand and refine their searches.

Creativity

What makes Timely stand out is the ability for users to create personal events and view their life milestones alongside historical events on a single timeline. This allows users to track shared experiences over time and explore friends' life stories. In future implementations, users would also have the option to share their timelines externally.

Personalization

In addition to historical exploration, the application serves as a tool for tracking personal milestones, making it valuable not only for academic research but also for personal storytelling.

Key features includes:

1. Create Personal Events: Logged-in users can add personal events with descriptions.

2. Read Events: Users can explore categorized events (e.g., historical, academic, personal) and toggle them on/off. They can filter events based on timeframes and expand event details.
3. Edit Personal Events: Logged-in users can update personal events.
4. Delete Personal Events: Logged-in users can delete personal events.
5. Search & Filter: Users can search for events using keywords and refine timelines by event categories or timeframes.
6. Shared Timelines: Users can create and share timelines with friends, combining life events into a single view.
7. Snapshot & Export: Users can capture and share timeline screenshots.

We successfully allow users to explore academic and historical timelines, create personal timelines, and search/filter by topic and year as we originally envisioned. However, in the end, we failed to complete social sharing ability to full-functioning level and trending insights, which could have enhanced user engagement. These changes will be touched on further in section 5.

3. Schema and Data Changes

Did you change your database schema?

Added last_active, status, and role to Users table

- last_active tracks when the user was last active
- status tracks whether the user is marked for deletion or not
- role tracks the administrative status of the user

Did you change your data source?

We initially wanted to use GDELT since it can be called via SQL queries, but instead went with Wikidata for a similar dataset but greater accessibility. Wikidata is easier to structure the data output into the desired form (event_name, event_date, event_description, etc), though it does use a relational data model and requires slightly more complex queries to work with. A query to get instances of historical events in this schema is

```
SELECT ?event ?eventLabel ?date ?description ?source ?sourceLabel ?author
?propLabel ?categoryLabel WHERE {

 {?event wdt:P31 wd:Q13418847.}  # instance of 'historical event'

 UNION
```

```
{?event wdt:P31 wd:Q2324975 .}  # instance of 'national founding'

?event schema:description ?description.

FILTER(LANG(?description) = "en")




OPTIONAL {

  VALUES ?prop { wd:P646 wd:P5106 wd:P10565 wd:P2671 wd:P1343 }

  ?prop wikibase:directClaim ?p .

  ?event ?p ?source .

  OPTIONAL {

    ?source rdfs:label ?sourceLabel .

    FILTER(LANG(?sourceLabel) = "en")

  }

}
```

Explain what you changed and why.

We have several features that we didn't anticipate having to do with user activity, such as last_active, status, and role. Most of these features were administrative features that allow for better interactivity within the database, such as being able to search for a username or event by substring.

4. ER Diagram and Table Differences

User Table Attributes:

In our initial design the User table only stored a user_id, username, and password. In the final design the user table stored user_id, username, password, role, last_active, and status. These changes were made to add more organization to the application. Without the attributes in the Users table such as role and status, the application would be unable to execute commands such as mark as inactive and show administrator tools. If we had more time, we would add more foreign keys to the tables (such as user_id in Categories) to allow for greater personalization.

Category Hierarchy Table:

Initially our design included a table called categoryHierarchy which stores a parent_category_id and category_descentent. We planned to use this to highlight how different fields interact with and depend on each other. However, while implementing the application, we realized it only made the user experience more confusing. If we had more time, a better design could include a visual representation of the category hierarchy with a collapsible tree and allow users to select a category and all its descendant categories at once.

5. Functionality Changes

+ User-Created Categories
Our original scope only allowed users to create their own events. However, it soon became clear that limiting users to existing categories restricted their ability to personalize their timelines. Therefore, we expanded the functionality to allow users to create their own categories.

+ Admin Dashboard
As the system grew more complex and more users and data were added, logging into GCP to monitor the system became cumbersome. To address this, we added a basic Admin Dashboard where administrators can quickly search for users and events, and view basic statistics for each category, as well as promote users to administrators.

+ Category Leaderboard
Although not part of the original plan, through deeper research into our target audience's motivations, we realized that users would likely be interested in statistics about how many events exist within each category. Thus, we added a Category Leaderboard to highlight the most documented fields.

- Trending Events Page
In Stage 3, we proposed adding a page to display trending events. However, due to limited user interaction data—specifically, the lack of click tracking—we determined that a trending page would not provide meaningful insights. As a result, we decided to remove this feature from the scope.

- Category Hierarchy
In Stage 2, we introduced the idea of adding hierarchies to the various categories in our database to better showcase the interrelation of different fields. Upon closer analysis of the data, we realized that scientific fields are often not easily grouped into clear hierarchies. Rather than enhancing the user experience, this design would likely introduce confusion, so we decided to remove it.

- Direct Instagram App Integration
Initially, we planned to allow users to directly open the Instagram Story page and post their timeline as a story. However, because our application is primarily web-based, users are not always on mobile devices. We decided to postpone this functionality for now. Nevertheless, we created a mock-up of what the Instagram sharing feature could look like if implemented in the future.

6. Advanced Database Features

Search by Keyword:

Search by keyword was implemented in multiple functions in the application, particularly for retrieving events using keyword from event title; and for retrieving users using keyword from username.

Below is the example of keyword search for admin to quickly find an event:

```
   let query = 'SELECT event_id, event_name, user_id FROM Events';

   …

   if (search) {

     query += ' WHERE event_id = ? OR event_name LIKE ?';

     params.push(search, `%${search}%`);

   }
```

This query allows the admin to search for events by id or by using keywords in the title.

Below is another example of keyword search for users to find an event based on their requirement:

```
let sql = `

   SELECT e.*, GROUP_CONCAT(c.category_name) as categories

   FROM Events e

   LEFT JOIN EventCategory ec ON e.event_id = ec.event_id

   LEFT JOIN Categories c ON ec.category_id = c.category_id
   WHERE 1=1

 `;

 …

     sql += ` AND ec.category_id IN (${ids.map(() => '?').join(',')})`;

     params.push(...ids);

…

   sql += ' AND YEAR(e.event_date) >= ?';

   params.push(startYear);

 …

   sql += ' AND YEAR(e.event_date) <= ?';

   params.push(endYear);

…

 if (q) {

   sql += ' AND (e.event_name LIKE ? OR e.event_description LIKE ?)';

   params.push(`%${q}%`, `%${q}%`);
```

This query allows the user to search for events based on categories, year range, and keywords.

Stored Procedure:

This stored procedure sets the users last_active attribute to be the current time. It currently activates upon login or event creation.

```
DELIMITER //

CREATE PROCEDURE setUserActive(IN currentUserId INT)

BEGIN

 UPDATE Users
 SET last_active = NOW()

 WHERE user_id = currentUserId;

END //

DELIMITER ;
```

This stored procedure is useful because last_active must be updated at multiple locations across the program. Though it is only used in two places at the moment, it could easily be used in more, which means that it would be most optimal to store as a stored procedure.

Transaction:

Get Leaderboard: The query for this function involves counting the number of events created and the number of categories a user prefers. Which are used to calculate how active the user is in the app.

```
SELECT

       u.user_id,

       u.username,

       COUNT(DISTINCT e.event_id) AS events_created,

       COUNT(DISTINCT ucp.category_id) AS categories_followed,

       (COUNT(DISTINCT e.event_id) + COUNT(DISTINCT ucp.category_id)) AS
activity_score

     FROM Users u

     LEFT JOIN Events e ON u.user_id = e.user_id

     LEFT JOIN UserCategoryPreferences ucp ON u.user_id = ucp.user_id

     GROUP BY u.user_id, u.username

     HAVING activity_score > 0

     ORDER BY activity_score DESC
```

```
    LIMIT 15;
```

The query helps join and group multiple tables with user information to obtain information.

Trigger:

Prevent Admin delete: When someone tries to delete an user with role ‘dev’, the database does not allow such deletion.

```
DELIMITER //


CREATE TRIGGER prevent_admin_delete

BEFORE DELETE ON Users

FOR EACH ROW

BEGIN

 IF OLD.role = 'dev' THEN

   SIGNAL SQLSTATE '45000'

     SET MESSAGE_TEXT = 'Cannot delete admin users!';

 END IF;

END;

//

DELIMITER ;
```

This prevents any accidental deletion to the admin account, by adding an extra guard to prevent deleting an Admin account, forces delegation to user role before being able to delete the account.

Delete events when creator deleted their account: When an user is deleted from the table, all events created by them will also be deleted.

```
DELIMITER //

CREATE TRIGGER delete_user_events

AFTER DELETE ON Users

FOR EACH ROW

BEGIN

 DELETE FROM Events

 WHERE created_by = OLD.user_id;

END;

//

DELIMITER ;
```

The trigger deletes information upon user account deletion, allowing the user to leave no trace on the site.

Constraints:

The main constraints we used were DEFAULT NULL and NOT NULL; NOT NULL was used for things such as source_name and event_name in their respective tables. Almost every attribute is labelled either DEFAULT NULL or NOT NULL. event_id and source_id were automatically incremented via AUTO_INCREMENT to get a unique id for each new entry. Users.username has a UNIQUE constraint on it to prevent more than one user from having the same username, which could lead to login difficulties.

These were added to keep the data from being manipulated by spontaneous additions. If an event were to be accidentally added, it would be easily caught if the user did not fill out the event_name field. source_year and author in Sources both have the constraint DEFAULT NULL, which makes it easier to search for default values in the database.

7. Technical Challenges (One per Member)

For each team member:

Name	**Challenge**	Advice
Nai-Syuan Chang	Debugging the database connection issue involved testing multiple pipelines and carefully verifying each component to identify the root cause.	1. Check if your computer could connect to your GCP database at all (checking your credentials, IP allowlist). 2. Check if your backend server (Node.js, index.js) was able to connect to the GCP database. 3. Check if your backend API routes were actually receiving the database data correctly (e.g., on your localhost port like 5050, there's a chance that the port you use was already occupied by another app). 4. Check if your frontend was getting data from your backend (so checking API fetch URL).
Beau Magro	Getting the source_id from Sources table to use in Events.source_id when submitting CreateEvent	1. Have to parse output in json format and send the source_id as a source_id field in the output 2. Additionally, handling edge cases (such as NULL values) is important and the call to Sources should be skipped if source_name=NULL
Taisia Kalinina	Middleware debugging and authorization to ensure users can only delete and edit their events.	1. Get a good understanding of authentication flow before setting out delete and edit routes. Specifically understand what information is stored and how the server retrieves it. 2. Test the backend routes separately before connecting to the frontend.

		3. Console.log statements in the backend especially including req.user or req.params.id are extremely helpful with understanding what information is not being sent.
Archie	Designing a user interface that was both visually appealing and intuitive, while making sure it adapted properly to all the different functionality we kept adding during development.	Focus first on the core user journey: what's the most important action a user should be able to take easily? Design aesthetics around that. Also test UI changes as early as possible, even with just a team member, so you don't overdo something that is not working well from a user's perspective.

8. UI updates

Category Option Layout:

In our initial design we planned to have the category options be displayed in a column on the left side of the page. In the process of designing the application we decided to move the categories to the center of the page above the keyword search bar, date filters, and timeline. This layout change improved the visual flow of the page along with condensing all of the timeline filtering into one area of the page. It also allowed the timeline to expand across the full width of the page, making it easier to see all event dots at once and reducing the need to use the horizontal scroll bar.

Keyword Search and Year Filter Accessibility:

In our initial design we had the search bars for the keywords and dates hidden behind a search button. During development, we decided to keep the search bar and year filters always visible on the main page. This makes it easier and faster for users to filter for the specific timeline they want without extra clicks. Additionally, it improves discoverability of the features since users will immediately know they have these options to refine their search.

Create/Edit Event User Interaction Flow:

We initially planned for the create even feature to be a popup on the main page. However after taking into consideration all the fields required to describe an event we decided to move create event and edit event to a dedicated page. The page contains a complete form for each individual event, even including the option to make a new category for the event. As a result users can make events without distractions or the risk of losing unsaved edits if they accidentally click out of the popup before creating an event or saving their updates.

9. Future Work Suggestions

If someone were to continue building on our application, we suggest the following backend and database improvements:

Enhance User Tracking:
Expand the Users table to include additional fields, like the number of login sessions, last login timestamp, and user activities. This would provide a more detailed view of user engagement.

Implement Event History Logging:
Introduce a version control or audit log system for events. Instead of permanently overwriting or deleting event records, maintain a historical log that records all updates and deletions. This would help with transparency of the app.

Expand Admin Dashboard Functionality:
Add more admin tools to the dashboard. In addition to searching users and events, admins should be able to: view detailed user activity, send notification emails directly to users, soft-delete user accounts (marking them as inactive or restricted instead of permanently deleting immediately), ensuring safer account management.

10. Final Division of Labor

Name	Role	Key Contribution
Nai-Syuan Chang	Project Manager, System Engineer	Project Coordination, Connect Backend & Frontend, Login Session Logic
Beau Magro	Backend Engineer, Data Engineer	Implemented Critical Database Logic, Backend Logic, Collect/Import Data, Data Processing
Taisia	Backend Engineer, Data Engineer	Collect,Import & Process Data, Backend Logic, Frontend Features & Backend Connection

Archie	Frontend Engineer, UI/UX Designer, Backend engineer	Implement Frontend Logic, Create UI/UX Design, Recorded Demo Video, leaderboard logic for top users