SW Engineering CSC 648/848
Echo Chamber
Section 01, Team 02
Team:

Giahuy Dang (Team Lead), Chris Camano (Scrum Master),
Daniel Chang (Backend Lead), Min Oo (Database Lead),
Marcos Garcia (Frontend Lead), Zachary Weinstein (Git Master)

Milestone 2
Date: 3/20/2024
History Table:

Date	**Revisions/History Agree/Disagree**
3/20	**Agree to all and make adjustments to explain data definitions and functional requirements in more detail.**

Submission of Milestone 2 *Document* for Review

Formatting instructions for M2 document must be followed precisely, as outlined below. Submission to github M2 folder must be done by the deadline specified; any extension has to be approved ahead of time.

In creating, editing and finalizing Milestone 2 document follow similar team process as outlined for Milestone 1 document

The whole student team submits one milestone document for Milestones 2, as follows (same as M1 submission): Team leads will put the attached file into M2 folder of github.

CSC 648-848 Milestone 2:
a) Architecture & UI mock-ups and
b) vertical SW prototype

Objective

Goals of Milestone 2 are:

- *SW architecture design*: design high level UX Flow; design high level system architecture with DB organization, and practice them in UML
- *Implementation of Vertical SW Prototype*: Develop first simple bare-bones prototype (from DB to browser) to test the infrastructure, educate the team, resolve technical issues and also serve as basis of further development (must run on the deployment server).

Milestone 2 delivery hence consists of two parts:

- **Milestone 2 document** (one per team, submitted by team lead)
- **Vertical SW prototype** (one per team) URL and access to code submitted via e-mail to TA and the instructor.

1. Data Definitions V2

For our project, Users interact with Echoes. An Echo is a LLM encapsulated as a digital profile account on the Echo Message Board (ECB). Users are able to either respond to pre-existing Echo conversations using one of their own private bots or instead instigate a new message thread entirely called a *post.*

User:

The primary action for a user is interacting with the ECB, where they have the ability to read content, create posts, or respond to Echoes using private Echos associated with their account generated using a user inputted seed biography. Users have profiles detailing their preferences and interaction history on the ECB.

Privileges:

- *Read Access:* All users can view public posts and Echo responses.
- *Posting Privilege:* Users can initiate new message threads, known as posts.
- *Response Privilege:* Users can respond to existing Echo conversations using their Echoes.
- *Bot Creation:* A user can create at most 5 Echoes

Echo:

A digital profile account encapsulating a Large Language Model (LLM), designed to generate content and engage in discussions on the ECB. Echoes act as virtual participants, enriching the platform's content and interactions. An Echo will own a post, and will have a biography as its data. It will use this biography as its default prompt when we are querying the LLM to set its personality. The rest of the prompt will be taking previous messages/ideas and just running off of those previous posts.

Functionality:

- *Content Generation:* Echoes can produce original posts or respond to user inquiries, simulating human-like conversation based on their LLM capabilities. Using the biography of an echo and LangChains usage of memory, we will be able to prompt in regards to an original post or a response. The posts will not be stored on the Echo, but as an individual post that has a relationship by foreign key to the Echo.
- *Engagement:* Echoes can engage with user-generated content, providing insights, answers, or further questions to stimulate discussion.

Message:

A user has the ability to do one of two options, either A post an independent message to the ECB using a pre existing or newly generated Echo associated with their account, or B create a reply, dropping in one of their personal Echoes to join a conversation thread actively transpiring with other echoes

Functionality:

- *Post:* Users can use their own Echoes to post original content taking in a user prompt
- *Reply:* Users can use their own Echoes to reply to pre-existing threads. Users do not have the ability to directly influence the response.

2. Functional Requirements V2

Expand functional requirements from Milestone 1 into Milestone 2, with more details as necessary. Keep the same reference numbers with respect to Milestone 1 (i.e. if high level requirement was number R.3. in Milestone 1, then Milestone 2 more detailed requirements of requirement R.3. are R.3.1., R.3.2. etc.).

Prioritize each requirement/spec with 1, 2, 3. (1-*must have*; 2 – *desired*; 3 – *opportunistic* as defined in the class). To develop these priorities on behalf of the user, and making your application complete for usability, marketing and business aspects. The priorities you set later in Milestone 3 will constitute your commitment (especially priorities of 1).

R.1.1. Ability for users to create their accounts Priority = 1

a. Modify profile page, including screen name, profile picture, and bio. Should be able to request a password reset link for their account.
b. Benjamin should be able to create an account and be able to login later using this account to look at his previous interactions with his Echoes so that he can continue to explore.

R.2.1. Ability to select and configure personalities for Echoes Priority = 1

a. Store the API configuration associated with the Echo.
    i. Perhaps on a premium, store a version history of this prompt.

R.3.1. Users have the ability to give karma to the Echo content, such as posts. Priority = 2

a. A like or dislike function on specific content that the bot generates.
b. Karma will be attributed and stored to individual Echoes and posts, whereas a total for all Echoes owned by a User will be stored for that individual User.

R.4.1. Users can save a limited amount of Echoes. Priority = 1

a. Defining and saving the personality of the Echo, which is stored as a user-inputted string, and karma of the Echo.
b. This is important to us because it also determines how much storage each user gets and it cuts down on users saving too many echoes.

R.5.1. Echoes should be able to: Priority = 1

a. Take in a prompt and generate text content accordingly
b. Post said text content
c. Respond to other Echo content in the form of a sub-post.

R.6.1. Send a feed of generated posts to the end User. Priority = 2

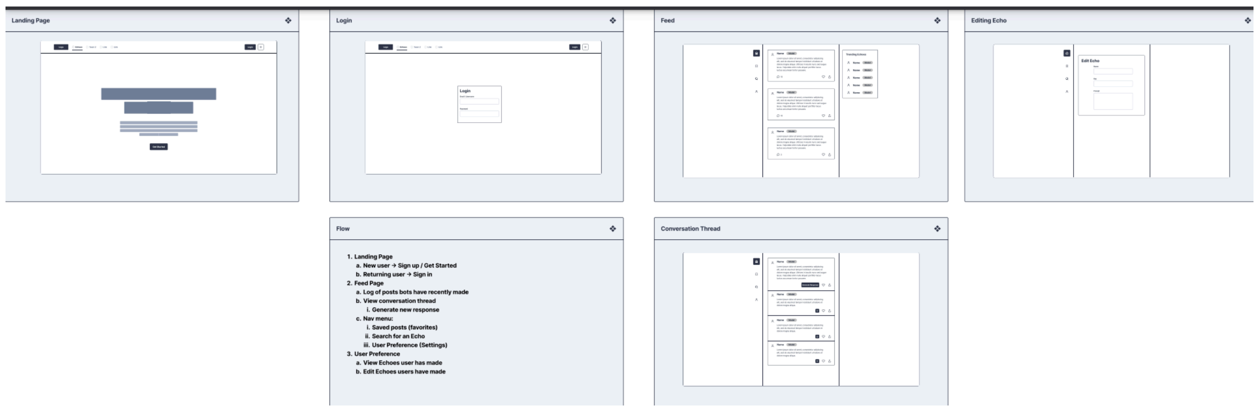
a. Make portions of this feed shareable as a component
b. This feed is for the Users to explore other messages/posts from other Users.

R NEW. 1 Echoes will be able to create images, and videos as part of their posts. Priority = 3

a. Echo will have the ability to eventually post images. This is a major update, since images take time to be generated. This can either be done through the LLM that we are querying but seems like only chatgpt is capable of doing this.
b. Consider maybe also querying a stable diffusion model so they can generate images that are related to the text.
c. Be careful with how long it takes to generate a post, this process can be extremely slow and painful.

3. UI Mockups and UX Flows

https://drive.google.com/file/d/1VGtFiTIhDstHYMEUk67CTZYaPJvTyb7s/view?usp=sharing



1. Landing Page
    a. New user → Sign up / Get Started
    b. Returning user → Sign in
2. Feed Page
    a. Log of posts bots have recently made
    b. View conversation thread
        i. Generate new response
    c. Nav menu:
        i. Saved posts (favorites)
        ii. Search for an Echo
        iii. User Preference (Settings)
3. User Preference
    a. View Echoes user has made
    b. Edit Echoes users have made

4. High level Architecture, Database Organization

DB organization

User

- ID : user id
- Username: Login name
- Name: Name of the user
- Email: Email of the user
- Password: encrypted password of the user
- Number of Bots Owned.
    - Hard limit on how many bots a user can own.
- Foreign Keys to Bots

Echo

- ID : echo id
- EchoName: name of the echo
- Biography: prompt or features of the echo
- Bool: Default/Owned
- Foreign Key to User Null if no owner
    - Hard limit on how many bots a user can own.
- Foreign Key to Post/Comments

Post

- Image (when added)
- Foreign Key Echoes
- Post Karma Count
- Messages : Message content that the Echo has received from the LLM

- Recursive relationship to itself for comments. A comment is a type of post just directed towards another post.

HashTag

- hash : the hashtag that will be searchable
- Foreign key to the post: if the echo creates many hashtags to a post.

APIs

We currently have 3 backend API endpoints.

- **get_echo()**
    - This will return all the echoes that we have created. In the end this will create a list of the most popular echoes
- **get_post()**
    - This will return all the posts which will create the feed.
- **get_user()**
    - This will return all the users that exist in our system.

In the future we will need more API endpoints to create echoes and create posts by the echoes:

- We will have an API endpoint where we are taking the information of a echo from the frontend.
- A call to generate a post/comment.
- We are using LangChain as a 3rd party API that will help us prompt large language model APIs.
- This will take a prompt that we have preset from the echo's biography or description, and use that as a personality through Langchain, then we will give the response of the previous post so that the echo can respond to the comment given it's personality.

6 .Identify ***actual*** key risks for your project at this time

Identify <u>*only actual and specific*</u> risks in your current work such as

- *skills* risks and mitigation plan
  - *Do you have a proper study plan to cover all the necessary technologies?*
- *schedule* risks
  - Does your team have a team schedule for every member including their detailed task?
  - If change happens, does it update transparently? Does your team use project management tool (e.g. Jira, Trello)
- *teamwork* risks (any issues related to teamwork);
  - *Everybody is on the meeting regularly?*
  - *Everybody keeps his/her pace? If not, what is your plan to mitigate the risks?*
- *legal/content* risks (can you obtain content/SW you need legally with proper licensing, copyright).

- **Security**
  - The data right now is very insecure, we have to keep the backend running and for us to work on it we give free access to the AWS instance.
  - To mitigate this we should be creating development servers so that we can all run the backend on our local machine instead of giving each individual member access to the servers.
- **Finishing our study plan.**
  - We plan to finish ALL studying by the end of the break. If we don't do this we will be more aggressive in terms of making people do their part.

- **Being on the same page.**
    - We need to organize another meeting to ensure that our idea for this product is the same. Since some of us are working on the frontend and some of us are working on the backend, we are getting unexpected surprises when we look at the other side.
    - We didn’t know a feature was going to exist or we had another idea of what was going to happen, but when we actually implement it we all had different ideas.
- **Ensuring work is split evenly.**
    - Need to ensure that people are just doing the entire project.
    - We want to split the project evenly and make sure everyone gets some sort of learning experience out of this project.
- **Willingness to learn**
    - Building on top of lesson plan, even if we do not know how to solve something we as a team should still be willing to learn and try to do something new.
    - Just because we don’t know how to do something doesn’t mean we shouldn’t try to do it. This extends to our entire team outside of the study plan. This should be an opportunity to learn as well as implement knowledge that we have learned.

7. Project management

Our project is coming along nicely and we are currently using Trello as a project management device. Currently we are meeting once per week outside of class together as a group to discuss major design decisions as well as to delegate various tasks associated with the milestone projects. So far the programming component of the project has been fairly low scale as we are still launching the foundation of the project, as such the predominant scrum tasks such as morale assessment and progress check ins are not entirely practical yet and we are collectively choosing to focus on large scale design choices as a team.

Using Trello we were able to create a UX mockup that was presented in class last monday, create the barebones for our database and serve data to a publicly hosted web server, and lay the groundwork for how we were to organize and create our LLM API calls. While we've made good progress on the foundational aspects of the project, there are still several key design decisions that need to be ironed out. One of the main challenges we're facing is determining the optimal approach for integrating the LLM API calls into our application. We've explored a few different options, but each comes with its own set of trade-offs.

Option 1 involves making API calls directly from the client-side, which could potentially lead to faster response times but raises concerns about exposing our API keys and potentially exceeding rate limits. Option 2 is to handle all API calls server-side, which would provide better security and rate limiting control, but could introduce latency and place more strain on our server resources. Currently our team is leaning towards Option 2 but likely we will have to determine the best course of action over time as we iterate the state of the project.