Capstone Verification and Validation Plan

Version 0
Group 16
COMPSCI 4ZP6
Dr. Mehdi Moradi
Feb 7, 2025

Members		
Alex Eckardt	eckardta@mcmaster.ca	400390784
Owen Gretzinger	gretzino@mcmaster.ca	400407289
Jason Huang	huanj168@mcmaster.ca	400374849
Sahib Khokhar	khokhs5@mcmaster.ca	400396918
Sarah Simionescu	simiones@mcmaster.ca	400363648

Version History			
Version	Authors	Description	Date
0	Eckardt, Gretzinger, Huang, Khokhar, Simionescu	Initial Document	Jan 29, 2025

Eckardt, Gretzinger, Huang, Khokhar, Simionescu

1. Project Description

The MeetingBot project offers a service to companies and individuals seeking to create a bot for video meetings, but lacking the resources to build and maintain integrations across all meeting platforms. By providing an open-source, self-hostable alternative to expensive paid platforms, we hope to make building meeting bots more accessible to developers. This document details the verification and validation plan that will be performed against our service.

2. Component Test Plan

2.1 Unit Tests

2.1.1 Back-End API Unit Tests

The backend API will be tested in Jest.

- This will allow us to perform unit tests and send mock API requests.

(getBots) /api/bots GET

- Ensure a mock request returns the correct output.
- Returns a list of bots when there are bots available to return.
- Ensure that an empty list is returned when there are no bots available.

(getBot) /api/bots/{id} GET

- Ensure a mock request returns the correct output.
- Returns a single bot's information correctly.
- Ensure that an error is returned when an incorrect id is submitted.

(createBot) /api/bots POST

- Ensure a mock request returns the correct output.
- Ensure a bot is correctly created after POST.
- Ensure each configuration in the request can be run.
- Ensure the database table corresponding to the bots list is successfully updated.
- Return's the newly created bot's information.

(updateBot) /api/bots{id} PATCH

- Ensure a mock request returns the correct output.
- Ensure the request updates the result in the database.
- Return the new updated values correctly.
- Ensure that if the bot id is incorrect, the proper error is returned.

(deleteBot) /api/bots/{id} DELETE

- Ensure a mock request returns the correct output.
- Ensure the correct bot gets deleted after the request.
- Ensure deleting a non-existent bot gives the correct error.
- Return the correct message that the bot has been deleted.

(getRecording) /api/bots/{id}/recording GET

- Ensure a mock request returns the correct output.
- Ensure the database table corresponding to recordings can be accessed.
- Return the recording link.
- Ensure an incorrect bot id shows the proper error.

[PRIVATE] (deployBot) /bots/{id}/deploy POST

- Ensure a mock request returns the correct output.
- Ensure the bot deploys with the correctly provisioned resources.
- Return the deployed bot's information.
- Ensure an incorrect id returns the proper error.

(getEventsForBot) /api/events/bot/{botId} GET

- Ensure a mock request returns the correct output.
- Return all the events associated with the bot correctly.
- Ensure an incorrect bot id returns the correct error.

2.1.2 Bot Script Unit Tests

As the bots are written using Playwright and Puppeteer, we can use the jest-puppeteer NPM module to write structured unit tests for the following bot actions. The running and deployment of a bot is up to the backend.

On startup, we want to ensure that the bot can connect to S3. In the event of a bad connection, we will retry several times before exiting. In the event of bad given parameters, the bot will just exit.

2.1.2.1 Navigation Unit Tests

Each Meeting platform has its own UI which can be changed at any time – thus unit tests for navigation are volatile - there is no guarantee that they will work in the future.
The tests we want to work on are:

- Ensure that the correct *bot* joins the correct *meeting* link when given a config file through the use of mock responses.
    - Since Teams, Meet, Zoom are different platforms, they have to have different bots. We want to ensure that the correct instance of a bot is spun up.
- Ensure the url creator creates the correct meeting link given the bot's config file through assertions of test and expected values.
- In the event something happens where the bot's instructions are no longer 'in step' with the state of the page, we will treat it like an error and abort.
    - Typically, this would only happen before the bot joins the meeting, (as this is where the navigation scripting is most important) so we can provision a new bot to replace it.
- Popups would be defined by the meeting page, so if something like that occurs we need to update our bot's scripting.

2.1.2.2 Meeting Join Unit Tests

- Ensure each bot can go through the meeting's join pages through the reviewal of logs (FAIL, PASS)
- Ensure each bot can enter their name and profile picture before the meeting starts.

- Need to handle cases where a bot is unable to join a meeting
    - For instance, Google Meets might flag the bot as 'suspicious', and not allow it to join the meeting at all.
    - Similar options with Teams and Zoom.

2.1.2.2 Meeting Leave Unit Tests

- Ensure that a bot leaves the meeting if one of the following conditions is met.
    - Silence: Bot should leave after if it does not detect any audio from participants for a set amount of time.
    - Empty Meeting: Bot should leave after it detects that it is the only meeting participant,
        - In the event of a scheduled join and the bot was only ever the only participant
        - If there were, at some point, other participants in the meeting but are now gone
    - Waiting Room Timeout: If the bot is not admitted to the meeting for a set amount of time, the bot should exit.
- In the event a bot cannot leave a meeting, we can simply close the web-page, which in turn will end the recording.

2.1.2.1 Recording Unit Tests

Ensure that a bot records the screen in the expected screen dimension
- Mock recording, and reading back video meta-data

Ensure that the recording is correctly uploaded to Amazon S3
- If met with an error, attempt to re-upload several times before exiting
- Create Mock tests with dummy files to ensure connection is valid

2.1.2.1 API Event Unit Tests

Ensure the bot can recognize an event occuring, and send that information to the backend.
- Track Participant Join
- Track Participant Leave
- Media Share Start
- Media Share End

These are events that occur during a meeting - so we would ideally spin up a meeting and perform these events manually - verifying that the bots can recognize events as they happen

2.1.2.1 Heartbeat Unit Tests

Ensure that the bot reads in the correct heartbeat interval, and that it sends pings to the backend at the correct frequency.
Ensure that the heartbeat loop will, at some point, end - allowing the bot to close. We can set a time limit (10 hours, as failsafe) which will kill the entire bot process anyways.

2.1.3 Front-End Component Unit Tests

To develop and execute the front-end unit tests, we will use Jest and React Testing Library.
- Jest allows us to create unit tests, estimate test coverage, mock API endpoints and more
- React Testing Library allows use to render our components and check for specific properties

Authentication /auth/sign-in /auth/sign-up

- If the user provides valid credentials to sign-in or sign-up, it will make the corresponding API request
- If the API request returns successfully, they will navigate to the dashboard
- If the user provides invalid credentials, a corresponding error message will appear on the screen
- If the request throws an error, a corresponding error message will appear on the screen
- If the request to log-in or sign-up is in progress, the user will see a loading indicator

Main Dashboard /app

- If the user has not yet created an API key, they will see the “Starter Dashbaord”, otherwise they will see the “Dashboard”
- The links on each card navigate the user to the appropriate URL
- If the request to fetch data for each respective card is loading, that card will show a loading state
- If the request to fetch data for each respective card faces an error, that card will show an error state

API Keys /app/keys

- If the user’s API keys are currently fetching, they will observe a loading state
- If the fetch throws an error, the user will see a corresponding error message
- If the fetch is successful, the user will observe the keys listed in a table on the screen
- If the user clicks the “Generate API Key” button, it will make the corresponding request
- If the request is in progress, the user will observe a loading state
- If the request is successful, the user will observe a success message, and the list of API keys will refetch
- If the request throws an error, the user will observe an error message

Bots /app/bots

- If the user’s bots are currently fetching, they will observe a loading state
- If the fetch throws an error, the user will see a corresponding error message
- If the fetch is successful, the user will observe the bots listed in a table on the screen
- If the user clicks the “Details” button, it will open a model and make the corresponding request
- If the request is in progress, the user will observe a loading state
- If the request is successful, the user will observe the details of the bot displayed in the modal on the screen
- If the request throws an error, the user will observe an error message

Usage /usage

- If the user’s usage details are currently fetching, they will observe a loading state
- If the fetch throws an error, the user will see a corresponding error message
- If the fetch is successful, the user will observe the usage, aggregated by the selected time period, displayed on a graph
- If the user changes the selected time period, the request will refetch with the news selected time period

2.2 Performance Tests & Metrics

2.2.1 Back-End API Performance Tests & Metrics

2.2.1.1 Time to Join Meeting

- **Metric:** The bot should be able to join a meeting within 10 seconds of the request.
- **Measurement Methodology:** Measure the time between the join request and the confirmation of the bot's successful presence in the meeting using automated monitoring scripts.

2.2.1.2 Time to Download Recording

- **Metric:** Transcripts and recordings should be ready for download within 5 minutes of the meeting ending.
- **Measurement Methodology:** Measure the interval from the meeting's end to the availability of downloadable transcripts and recordings using event-based logging and API monitoring tools.

2.2.1.3 Bot Availability and Uptime

- **Metric:** Bot availability should meet or exceed 99.9% uptime over a given month.
- **Measurement Methodology:** Continuous monitoring using synthetic requests and ping tests.

2.2.1.4 API Latency

- **Metric:** API response times for meeting scheduling, transcription requests, and user interactions should be under 200 milliseconds for 95% of requests.
- **Measurement Methodology:** Track using distributed performance monitoring tools.

2.2.1.5 Scalability Performance

- **Metric:** System should handle a minimum of 500 concurrent meeting bots without degradation in performance.
- **Measurement Methodology:** Perform load testing to simulate concurrent users.

2.2.1.6 Security Performance Metrics

- **Metric:** No unauthorized data access attempts or breaches during monitored operations.
- **Measurement Methodology:** Use penetration testing, vulnerability scanning, and logging through tools like AWS GuardDuty.

2.2.2 Bot Component Performance Tests & Metrics

2.2.2.1 Recording Metric

- **Metric:** A bot should be able to record a meeting it at least 720p, ensuring all valid elements are visible
- **Measurement Methodology:** Track during development testing and ensuring quality

2.2.2.2 Regression

- **Metric:** Ensuring that as time goes on, the bot is still able to go through each meeting platform as expected.

2.2.3 Front-End Component Performance Tests & Metrics

2.2.3.1 Latency Metric

- **Metric:** The frontend should be responsive, taking less than 5 seconds to start up and load in data for clients, and less than 2 seconds for switching pages (low latency).
- **Measurement Methodology:** Using Performance Tests such as Chrome's Web Tools (Lighthouse, Performance Throttling)

2.2.3.2 Mobile Usable

- **Metric:** The frontend should be able to be used by mobile devices, as clients may want to check on the fly.
- **Measurement Methodology:** Performance Tests such as Chrome's Web Tools (Mobile View)

2.2.3.3 Regression

- **Metric:** Ensuring that as new features are added to the backend, frontend functions as expected