Jesse Tiemens Software Engineering Capstone Documentation

Task 3

# B. Functionality Attributes

* Inheritance example:
  + The React framework, that this application uses heavily, is built on the concept of inheritance, as all React components that are created automatically inherit from the base ReactComponent class.
* Polymorphism example:
  + The ServerChannel component exemplifies polymorphism, as it is a single defined component that renders differently depending on properties passed during construction. Specifically, it changes based on the type property of the channel object that is passed to its properties. It looks up which icon to render next to its label based on which type of channel is passed, it will render a Hash icon for a text channel, a Mic icon for an audio channel, and a Video icon for a video channel.
* Encapsulation example:
  + All modals are encapsulated in a modal provider that is applied to the root layout document.
* Search functionality with multiple row results and displays:

A screenshot of a computer

Description automatically generated

* A database component with the functionality to securely add, modify, and delete the data:
  + Messages are stored in the database, and can be interacted with using the following methods:
    - **Add**: Typing a message into the chat box on the bottom of the screen and sending it adds it to the database.
    - **Modify**: Messages that you have previously sent can be modified by clicking the edit button in the top-right corner of the message display, which will update the database as well as send out a signal on a web socket to update the message on all other users’ screens.
    - **Delete**: All users can use the delete button in the top-right corner of the message that they have sent, and moderators/administrators can delete anyone’s messages. This does not technically delete the message from the database, as it will only remove the content of the message, and still show up to other members like so:
    - A screen shot of a computer

      Description automatically generated
    - For examples of actual deletion from the database, that happens when channels or servers are deleted.
* Ability to generate reports with multiple columns, multiple rows, date-time stamps, and title:
  + Visiting /generate-reports will provide these tables to report servers and profiles that exist on the application.

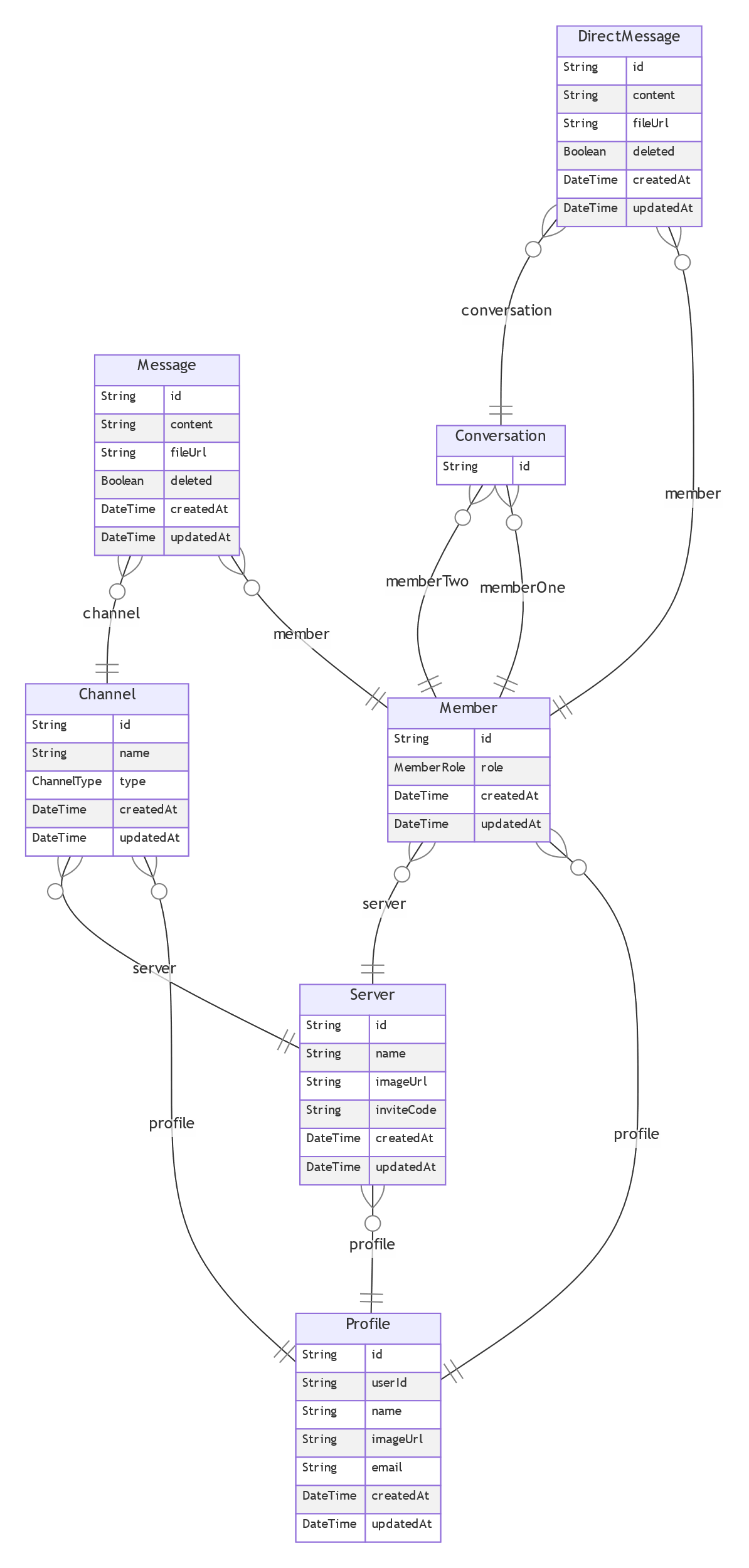
A screenshot of a computer

Description automatically generated

* Validation functionality:
  + A user is allowed to upload attachments to messages in the form of only image formats or pdfs, validation code is provided to prevent any other file format from being uploaded.
* Industry-appropriate security features:
  + The entire website is protected by an authorization middleware process. Accessing any page on the website will redirect to a sign-in/sign-up page so that unauthorized access is impossible. Further authorization on a server-by-server basis is provided through administrator and moderator roles that can be assigned by server owners and administrators.
* Design elements that make the application scalable:
  + The fact that we have chosen cloud solutions for both hosting the server and the database is inherently scalable. Design elements that the application itself utilizes to promote scalability include:
    - Any area on the user interface that has a variable amount of content is placed within a ScrollArea component.
    - Messages on servers and direct message conversations will obviously rack up many items as people chat. There is an inherent pagination to the way the client loads messages. It only loads the most recent page’s worth of messages, based on the viewport height of the device. As the user scrolls up, it will load more messages progressively. This allows for a smooth user experience that scales with the number of messages in each location.
    - As servers grow, there will end up being many users and channels. There is a search component for quickly locating elements, so that the scale of the server does not impact a user’s ability to locate a particular element quickly and easily.
* A user-friendly functional GUI:
  + Our GUI is user-friendly, as it is trying to replicate a GUI that is already user-friendly. However, it can also be argued that our GUI is user-friendly as no functionality is more than a couple of clicks away. Our GUI is also functional, as every interaction that is available to the user have all been tested thoroughly to ensure functionality.

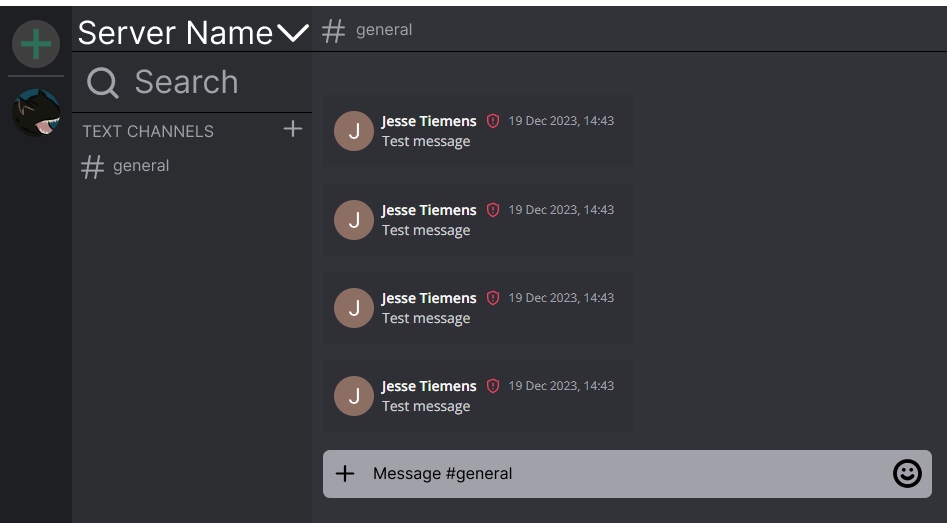
# C. Other Documentation

## Class Diagram



## Design Diagram

Low-fidelity wireframe of main view:



## Web App Host Location

<https://ele-discord-clone.up.railway.app/>

## GitLab Repository Location

<https://gitlab.com/wgu-gitlab-environment/student-repos/jtieme2/d424-software-engineering-capstone>

## User Setup/Running Maintenance Guide

While the application is still being developed, these are the steps to set up and run the application.

1. Install Visual Studio Code.
2. Install Node.JS.
3. Clone GitLab Repository.
4. Open the source code folder in Visual Studio Code.
5. Open a Terminal in the bottom section of Visual Studio Code.
6. Run “npm run dev” to launch the server.
7. Visit <http://localhost:3000/> to visit the website.
8. Running “npx cypress open” will launch Cypress, the tool used for testing.
9. Running “npx prisma studio” will open Prisma Studio, a GUI for interacting with the application’s database.

After the application is deployed to Railway, our planned cloud service provider, the maintenance is incredibly simple. Upon the source code repository receiving a commit on the main branch, the server hosting the application is automatically redeployed with the new changes.

## User Perspective Application Use Guide

1. Visit the website.
2. Log in or register.
3. Create your own server or click on an invitation link you’ve received from a friend.
4. If you created your own server, if you click the server’s name and click Invite People to open a window that will generate an invitation link you can copy to your clipboard and send to your friends.
5. You can see which servers you are in on the left-most navigation panel of the screen, clicking on servers will allow you to navigate to that server.
6. The next panel will show you the available channels in that server, selecting a text channel will change the main section of your screen to show the messages that users have sent in that channel, with an input for you to type and send your own messages at the bottom.
7. Other channel options include audio and video channels, clicking on one of those will change the main section of your screen to be a media room where you can talk to and see the users that are currently in those rooms.
8. When you are done with the application, you can click your user avatar in the bottom-left of the screen and select sign out.

# D. Testing Documentation

In previous documentation for this project, we had landed on unit testing being our preferred methodology of testing. However, this approach is no longer valid, given the structure of this application. This application heavily relies upon server-side rendered components and tightly coupled database interactions, which are inherently incompatible with unit testing.

The methodology of testing we have used is more in line with End-to-End Testing, or Integration Testing. We have developed end-to-end testing specifications for the main interactive parts of the application for use with the Cypress testing suite.

All tests will result in no lasting change made to the application, as all the tests follow a similar pattern of first creating an instance of the component to be tested, then modifying it in various ways before deleting it, to test that creation, modification, and deletion all work.

## Servers Testing

Testing of the server component will be divided into 3 different tests, server creation, server renaming, and server deletion.

### Server Creation

**Test Steps**:

1. Click the create server button.
2. Upload a server image.
3. Confirm the upload.
4. **Assert** that the upload has finished.
5. Generate a random number between 1-100, and type into the Server Name textbox “Server #[number]”.
6. Confirm the server creation.
7. Wait 5 seconds for the application to catch up.
8. Click on the new server in the navigation pane.
9. **Assert** that the server navigated to has the name generated in step 5.

**Test Script**:

it("server creation", () => {

    // Click create server button

    cy.get("[data-test-id='create-server-button']").click();

    // Type in randomized server name

    cy.get("[data-test-id='create-server-name']").type(serverName);

    // Upload server image

    cy.fixture("server-image.png", null).as("serverImageFixture");

    cy.get("input[type=file]").selectFile("@serverImageFixture", { force: true });

    // Click upload confirmation

    cy.get("[data-test-id='create-server-image'] button").click();

    // Wait for upload to finish before

    cy.get("[data-test-id='uploaded-server-image']").should("exist");

    // Clicking confirm

    cy.get("[data-test-id='create-server-confirm']").click();

    // Assert proper creation

    // Wait a couple seconds for the server to process

    cy.wait(5000);

    // Click first server in list

    cy.get("[data-cy='server-icon']:first-child button").click();

    // Assert that it has the correct name

    cy.get("[data-test-id='server-name']").should("have.text", serverName);

  });

### Server Renaming

**Test Steps**:

1. Click the server dropdown.
2. Click the Server Settings option.
3. Clear the Server Name textbox.
4. Type “Server #[previous random number – 1]” into the textbox.
5. Confirm the changes to the server settings.
6. **Assert** that the name has been changed.

**Test Script**:

it("server renaming", () => {

    // Click server dropdown

    cy.get("[data-test-id='server-name']").click();

    // Click server settings option

    cy.get("[data-test-id='server-settings']").click();

    // Rename in modal

    serverName = "Channel #" + (randomNumber - 1);

    cy.get("[data-test-id='edit-server-name']").clear().type(serverName);

    // Confirm modal

    cy.get("[data-test-id='edit-server-confirm']").click();

    // Assert changes

    cy.get("[data-test-id='server-name']").should("have.text", serverName);

  });

### Server Deletion

**Test Steps**:

1. Click server dropdown.
2. Click Delete Server option.
3. Confirm deletion.
4. **Assert** that the browser is redirected to the default Test Server.

**Test Script**:

it("server deletion", () => {

    // Click server dropdown

    cy.get("[data-test-id='server-name']").click();

    // Click delete server option

    cy.get("[data-test-id='delete-server']").click();

    // Confirm modal

    cy.get("[data-test-id='delete-server-confirm']").click();

    // Confirm redirection to Test Server

    cy.get("[data-test-id='server-name']").should("have.text", "Test Server");

  });

**Servers Testing Results**:

A screenshot of a computer

Description automatically generated

## Channels Testing

Testing of the channel component will be divided into 5 different tests, channel creation, channel renaming, channel type changing, including audio and video, and finally, channel deletion.

### Channel Creation

**Test Steps**:

1. Click the plus button next to the Text Channels section.
2. Generate a random number between 1-100, and type into the Channel Name textbox “Channel #[number]”.
3. Confirm the channel creation.
4. **Assert** that the most recent channel to be added has the name generated in step 2.
5. Navigate to the newly created channel.
6. **Assert** that the welcome message is displayed.

**Test Script**:

it("channel creation", () => {

    // Click Plus button on Text Channels

    cy.get("[data-test-id='text-channels-label'] [data-cy='create-channel']").click();

    // Type channel name into modal

    cy.get("[data-test-id='create-channel-name']").type(channelName);

    // Confirm modal

    cy.get("[data-test-id='create-channel-confirm']").click();

    // Confirm the channel exists on the sidebar

    cy.get("[data-test-id='text-channels-content'] [data-cy='channel-label']:last-child p").should("have.text", channelName);

    // Click on new channel

    cy.get("[data-test-id='text-channels-content'] [data-cy='channel-label']:last-child").click();

    // Confirm channel welcome is displayed

    cy.get("[data-test-id='channel-welcome']").should("have.text", "Welcome to #" + channelName);

  });

### Channel Renaming

**Test Steps**:

1. Click the edit button next to the channel created in the previous test.
2. Clear the Channel Name textbox.
3. Type “Channel #[previous random number – 1]” into the textbox.
4. Confirm the changes to the channel settings.
5. **Assert** that the channel’s label has been changed following step 3.

**Test Script**:

it("channel renaming", () => {

    // Click edit button

    cy.get("[data-test-id='text-channels-content'] [data-cy='channel-label']:last-child [data-cy='edit-button']").click({ force: true });

    // Rename

    channelName = "Channel #" + (randomNumber - 1);

    cy.get("[data-test-id='edit-channel-name']").clear().type(channelName);

    // Confirm

    cy.get("[data-test-id='edit-channel-confirm']").click();

    // Assert changes were made

    cy.get("[data-test-id='text-channels-content'] [data-cy='channel-label']:last-child p").should("have.text", channelName);

  });

### Change Channel Type to Audio

**Test Steps**:

1. Click the edit button next to the channel referenced earlier in this section.
2. Change the Channel Type to Audio.
3. Confirm the changes to the channel settings.
4. **Assert** that the channel now appears in the Audio Channels section.

**Test Script**:

it("change channel type to audio", () => {

    // Click edit button

    cy.get("[data-test-id='text-channels-content'] [data-cy='channel-label']:last-child [data-cy='edit-button']").click({ force: true });

    // Change dropdown to audio

    cy.get("[data-test-id='edit-channel-form'] select").select("AUDIO", { force: true });

    // Confirm

    cy.get("[data-test-id='edit-channel-confirm']").click();

    // Assert changes were made

    cy.get("[data-test-id='audio-channels-content'] [data-cy='channel-label']:last-child p").should("have.text", channelName);

  });

### Change Channel Type to Video

**Test Steps**:

1. Click the edit button next to the channel referenced earlier in this section.
2. Change the Channel Type to Video.
3. Confirm the changes to the channel settings.
4. **Assert** that the channel now appears in the Video Channels section.

**Test Script**:

it("change channel type to video", () => {

    // Click edit button

    cy.get("[data-test-id='audio-channels-content'] [data-cy='channel-label']:last-child [data-cy='edit-button']").click({ force: true });

    // Change dropdown to audio

    cy.get("[data-test-id='edit-channel-form'] select").select("VIDEO", { force: true });

    // Confirm

    cy.get("[data-test-id='edit-channel-confirm']").click();

    // Assert changes were made

    cy.get("[data-test-id='video-channels-content'] [data-cy='channel-label']:last-child p").should("have.text", channelName);

  });

### Channel Deletion

**Test Steps**:

1. Click the trash icon next to the channel referenced earlier in this section.
2. Confirm the channel deletion.
3. **Assert** that the application correctly redirects to the #general channel.

**Test Script**:

it("channel deletion", () => {

    // Click delete button

    cy.get("[data-test-id='video-channels-content'] [data-cy='channel-label']:last-child [data-cy='delete-button']").click({ force: true });

    // Confirm channel delete modal

    cy.get("[data-test-id='delete-channel-confirm").click();

    // Confirm redirection to general

    cy.get("[data-test-id='current-channel-name']").should("have.text", "general");

  });

**Channels Testing Results**:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# E. Panopto Video Recording