Milestone 2

Title:

SmartStride: Toe-Walking Rehab

Names & Emails:

Cianna Grummer cgrummer2019@fit.edu

Alec Anzalone aanzalone2021@fit.edu

Kiera Ceely kceely2021@fit.edu

Bela Perdomo iperdomo2021@fit.edu

Caleb Phillips cphillips2021@fit.edu

Faculty Advisor:

Dr. Gu gul@fit.edu

Progress of Milestone 2:

Task	Progress	To Do
Create patient login page	100%	N/A
Create practitioner login page	100%	N/A
Establish security of logins	100%	N/A
Create forgot password/ help links and page	100%	N/A
New users sign up page	100%	N/A
New users sign up functions	100%	N/A
Create login functions	100%	N/A

Discussion:

- Create patient and practitioner login page:
    - Creating the login pages for practitioners and patients was not very difficult. Within the html I created a form that collects the username and password entered by the user with a submit button. Embedded in the html is JavaScript code that connects to the AWS API through a POST method that takes the username and password and sends it as a body to the API. Connecting to the API gave me some trouble in finding the correct syntax to establish the html to API connection. After

the connection is established the API sends the information to the Lambda login function then sends a response back to the API that sends it back to the JavaScript. Based on the response received from the API the page will be redirected to the user's account. Under Login Functions task I will go into more detail about the Lambda and API.

- Establish security of logins:
    - Establishing the Security of the logins gave me some difficulty in finding a viable method on how to make the passwords secure in every step of the API to Lambda to Database and back process. I found a package called bcrypt that allows me to hash the passwords from when they are first entered in the sign up or login. When signing up the password is stored in the database hashed so when the lambda pulls the password to check if the user entered the correct password, it comes back hashed. The lambda during a login will be sent the hashed version of the password that the user entered to compare. Simple syntax was used to compare hashed passwords.
- Create forgot password/help links and page:
    - The forgot password will take the user to a new page that has them input their username and based off of if the user is found or not the user will be able to answer their security question. If the user answers the security question correctly, they will be given their password to login. This information will be taken in as a form and much like the login and sign up the input will travel through the API to the lambda function that will verify if the user exists and if their security question is answered correctly and send the unhashed password to be displayed to the user. If the user is not found questions will be shown such as: "Did you login to the correct patient or practitioner login?" "Don't have an account? Sign up here:" The second question will take them to the sign-up page where they can create their account while the first question prompts the user to determine if they are using the wrong login form or not.
- New users sign up page:
    - The sign-up page works similarly to the login pages in that it has a form in the html that takes in the user input. The input is longer asking for first name, last name, security questions, etc. This data is sent through the same API as the login functions but using a different API resource. The resource still uses a POST method so we can send a body (the user input) to the sign-up lambda.
- New users sign up functions:
    - The functions used to complete the sign-up feature is the new API resource with a POST method that connects to a sign-up lambda function. The lambda function is in JavaScript which gave me a bit of trouble since I have never used JavaScript before this project. The sign-up lambda will determine what kind of user is signing up based on the user input and based on what the user has selected the

lambda will connect to the database and input the information into the corresponding table, patient or practitioner, after hashing the password of the user. The lambda function will then send a message back to the API stating if the sign up was successful or not. The API will send the message to JavaScript within the sign_up.html. Based on the response received the page will be redirected to the login page of the patient or practitioner based on the user type that signed up.

- Create login functions:
    - The login function was the hardest part of milestone 2 because it is the first lambda function and API call that I made. The function works very similar to the sign-up function but instead of inputting information into the database the lambda function will connect to the database and search in the table for a username matching the one used to try and login. If no user is found it will send a response to the html saying no user was found. If a user was found the password will be pulled from the database and compared to the password used to try to login. If the passwords do not match the lambda will send a response of invalid credentials. If the username and password are both a match the lambda will send a response back saying it was a successful login. The responses gave me some trouble because they are coded under numbers so if there is an error it will send response 401 but if it is successful, it will send a response code in the 200s. I had trouble with this on the html side because it took any 200 responses and claimed it was successful even if it was a response 202 (Invalid credentials). I fixed this issue by checking the exact response code that was given instead of checking if that response code was in the “ok” or 200s range. I also had issues with establishing the connections between the APIs, Lambdas, and html codes because AWS operates on VPC security groups that say what functions are allowed to do what. VPCs are like security cards allowing my Lambda function to access the database or my HTML code to access the API call. This was not obvious when setting the functions up. My other issue was establishing the triggers and layers for my lambda functions. The layers are basically import packages that you need if you want to use MySQL or bcrypt, so I had to create layers for these packages. Triggers are how the lambda function connects to the API resource. Since one API can hold many different resources and each resource can hold many different types of methods the trigger is needed to specify which resource and which method within that resource the lambda is going to be using.

Contribution:

I have created all the functions, API calls, API resources and methods, Lambda functions and database connections, as well as creating all the VPC security groups that are needed to complete the functions myself.

Plans for Milestone 3:

Task	Progress	To Do
Connect the logins to individual patient and practitioner pages	10%	Add to the existing API and Lambda to create pages
Add patient button to practitioner page	0%	Create the button and connect the correct page
Add patient page	0%	Create page with a form to add a patient
Add patient functions	45%	Create new API resource and method to connect to new lambda function
Use "Dummy" data to be displayed to patient and practitioner	15%	Get "Dummy" data from BME group, determine what should be displayed, Use the data to create functions to display the data

Discussion (Future Milestones):

- Connect the logins to individual patient and practitioner pages:
    - I need to collect the data and display individual information when logging into a user's page. Currently a static page is displayed after logging into a profile. I will need to make the practitioner's page display a list of their patients along with links to their patients' profiles that will hold the data from the individual patient selected. The patients page when logging in will need to update dynamically for whoever has logged in. To accomplish all this my plan is to collect all the data form the database table for the patient or practitioner then split their information string into variables that can be displayed in different areas of their profile page to make it "personalized". For the data sections I will need the "Dummy" data before I can accurately display that information.
- Add patient button to practitioner page:
    - Within the practitioner's page they should be able to add or remove patients from their patient list. If a doctor gets a new patient, it should be updated when the patient signs up but if an existing patient wants to switch practitioners their current practitioner will need to remove them from their patients list then the new practitioner will need to add the patient to their patient list. This should be a simple form that asks for the practitioner's username and the patient's username that they want to add or remove to send the information to the lambda function that will carry out the task.

- Add patient page:
    - A new page will be needed to hold the form to enter the patient that the practitioner wants to add or remove. This page will also hold the JavaScript that connects to the API resource for the lambda function to be called.
- Add patient functions:
    - A new API resource and method will need to be created as well as a new lambda function that will handle the adding or removal of the patient from the practitioner's patient list within the database. This will update the foreign keys within the practitioner's table and subsequently update the practitioners profile page where a list of their patients is shown.
- Use "Dummy" data to be displayed to patient and practitioner:
    - I will want to create a display for patient data that can be seen by the practitioner. This will be shown to the patient as well but in a more digestible way such as a graph showing when their ITW steps are more frequent or if they have increased or decreased their ITW steps in the past week or month by some percentage. I will want to create the displays based off the dummy data but any refinement on the displays will need a fixed form of data that my group is still unsure of. I will be trying to keep the input of data flexible while still displaying some form of data. This is subject to change based on my group's determination of the format of the data that will be received by the website and/or the database.
        - It is also important to note that based on my group's determinations the format of the data displayed will change as well as how much progress will be made in this area. Since this is very flexible, I will try to create the functionality of displaying data but without knowing the format of the input I may need to redo this task in milestone 4 depending on what my group decides.

Meeting Dates:

Wednesdays 1pm-2pm

Client Feedback:

Group Feedback:

- The UI was improved greatly
- Instead of clinicians use practitioners
- SmartStride was determined to be one word not two
- The format of the patient data collected by the device may change from CSV files to graphs or something else to be determined later
    - Keep the data input versatile/ flexible

- Graph.py (The Python file used to create the graphs of data) may be written by Alec

Dr. Chan's Feedback:

- Discuss how to present the data to the doctors as they might not know how to read the data graphs
- Determine what a doctor would find helpful from our collected data when diagnosing a patient with ITW
    - Talk to the BME group about this
- Possibly include a timeline of when frequent ITW steps are taken
    - Knowing when the ITW steps are most frequent might help the patients in their rehabilitation process

Advisor Meetings:

Discussion happened over email. The BME midterm presentation will happen in the week of 10/28/24 where I will be able to give a proper update.

Advisor Feedback:

•

Faculty Advisor Signature: _____ Date: _____

Evaluation by Faculty Advisor:

Task for Faculty Advisor: detach and return this page to Dr. Chan (HC 209) or email the scores to pkc@cs.fit.edu

Score (0-10) for each member: circle a score (or circle two adjacent scores for .25 or write down a real number between 0 and 10)

Cianna Grummer	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10

Faculty Advisor Signature: _____ Date: _____