DEVELOPER MANUAL

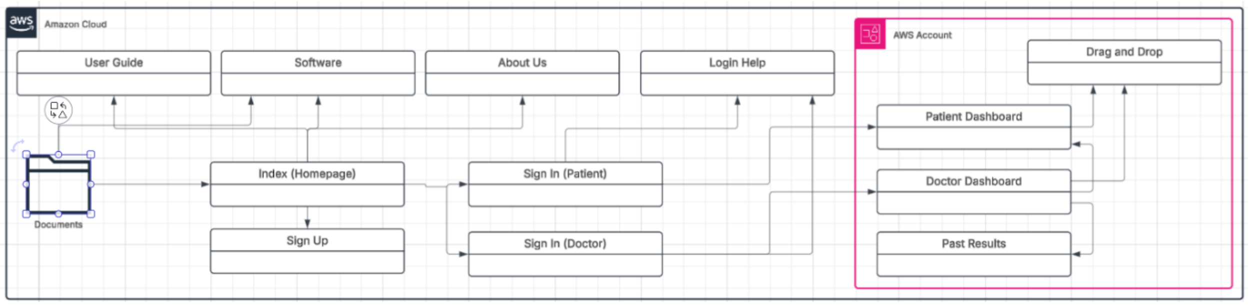


Figure 1

In Figure 1 the structure of the website can be observed. Before delving into each page and functionalities some important notes:

The documents folder is not accessible to the user as it is purely a backend resource to hold all photos and documents used by various webpages within the website.

The pages contained within the pink highlight box are only accessible after the user has logged into the website via the patient or doctor portal.

The pages User Guide, Software, About Us, and Index are accessible on every page. Therefore, their connections were not displayed to reduce noise and avoid confusion.

All functionalities use API calls that trigger the lambda associated with it. All the API calls occur in JavaScript within the HTML files for the webpage and all lambda functions are written in JavaScript with small bits of MySQL when accessing the database.

The only way to access the database is through an API call to trigger a lambda function that will handle all database queries.

The system is programed in the following languages: HTML, CSS, JavaScript, MySQL, and Python. A basic understanding of all these languages and AWS is assumed when describing the functionalities of the system as this is a developers manual intended for developers only.

All API calls are of the POST method type.

User Guide:

The user guide webpage holds all manuals and safety guides. The manuals include Developer Manual, User Device Guide, User Manual, and Safety Guidelines. This page contains standard HTML, and its corresponding CSS style guide is named style_deviceGuide.css.

Software:

This page holds all the requirements for the Senior Design class and should be removed if the website is put into production. This includes all six milestones, presentations, and preliminary reports. No deviation from standard HTML occurs with the CSS file named style_software.css.

About Us:

This is where we highlight the key features of the SmartStride system (including the database and website) are displayed. The HTML code is standard with the style sheet names as style_aboutUs.css

Login Help:

Login help allows a user to reset their password if they can input their username and answer the security question associated with their account. The database is accessed, and the old password is deleted and the new hashed password is saved to the database. The HTML contains some JavaScript. The JavaScript code collects the username entered by the user then the ITW_PasswordRecovery API is used to trigger the lambda function with the body containing the input username. The lambda then searches the database to see if the user exists and will return a body containing the security question and answer if the user exists if not the lambda will return a body containing the response of "User not found". If the user exists, the security question will be displayed to the user with a response box and an enter button. If the user does not fill the answer out the message 'Please provide an answer to the security question.' Will be displayed to the user. If the user answers the script will call the API again to send the answer to the same lambda function that will check the database and compare the results and send a correct or incorrect response back to JavaScript who will convert it to a readable response for the user. Upon a correct response the UI will change again, allowing the user to enter a new password. After entering the new password, the API will be called again to send the password to the lambda where it will be hashed and saved to the database. The UI will then display the response code from the lambda saying whether the password change was successful or not. The style is named style_loginHelp.css.

Index:

The homepage of the website is named index because it is required by AWS for the main page to be named index. *Trying to change the name will result in errors* when trying to deploy the website through AWS Amplify. The homepage contains links to the login portals as well as a link to the sign up page. The HTML and CSS is straightforward, and the style is called style_index.css.

Sign Up:

The sign up page is a form that the user must fill out to access the highlighted pink area of the website detailed in figure 1. The form collects the users first and last name, security question and answer, username and password, and asks the user to select if they are a patient or doctor. Upon selecting patient the user is asked to enter their doctors username and a security blurb appears informing the user that by typing in a doctors username they are legally allowing them to see, manipulate, and analyze their patient data. The code is quite simple when they select doctor nothing triggers but when the user selects patient it triggers the JavaScript code within the HTML that changes the blurb and user input box for doctors username to visible. Upon completing the form, the user clicks enter which will trigger the JavaScript again. First the data from the form is collected then the API signUpAPI is called which takes the user data and sends it to the corresponding lambda function. The lambda then checks the database to make sure the username is not already within the database and if it is not the information will be saved to the database. Afterwards the response code is sent back, and a message of success or failure is displayed. After a successful response is returned, the message is displayed then the user is redirected to the login page for their user type (patient or doctor). The style sheet is named style_signUp.css.

Sign In:

The sign in pages for doctor and patient are identical apart from dialog regarding user type and therefore their sections have been combined. The sign in portals contain links to the opposite sign in portal, login help page and sign up page. The JavaScript codes functionality works as follows: The user enters their username and password which triggers the JavaScript that immediately calls the API getUserPass that will send the username and password to the corresponding lambda function that will first check what kind of user it is and search the corresponding database table for that user's username. Once the username is found the lambda will compare the two hashed passwords and send a response code back. Upon success or failure, a message is displayed to the user. Messages are as follows: Login failed., An error occurred. Please try again., User not found., and Login Successful. The style is named style_patientLogin.css for both login pages.

Drag and Drop:

The drag and drop page are interesting because the API and lambda functions are accessible through the webpage and from the Raspberry Pi app. The drag and drop page is only accessible through the patient dashboard and is a back up method of uploading patient data in case of Raspberry Pi failure. It will only accept CSV files, and it can accept two kinds of data the step count that is used for the pie chart models that are used on the patient page and past results page and the accelerometer over time data that is used to create the graph on the patient page that is only visible to the doctors. The page has a section where a user can drag a CSV file and "drop" it that will trigger the JavaScript of the webpage through the event listener, or the user can click on the box and the JavaScript will be triggered pulling up the user's documents folder on their PC. Once a file is selected for uploading the file will be parsed and the API DragDropHandler will be called and sent through the API will be the CSV file data and the patient's username. Since only patients can have patient data the user type is not needed by the lambda function. Lambda will first check the parsed CSV data to check how many rows it contains. If there are only two rows, which is the maximum number of rows the pie chart CSV files can hold, it will put the data into the pie chart data table within the database. If the parsed data is more than two rows it is the graph data and will be inputted into the patient session data table. Before either form of data is inputted into the database the session number needs to be assigned. The Lambda will do this by checking the username in the database to see if they have any other data from a previous session. If the patient does the lambda will take the session number from the data and add one to the number for the new data. If there is no session number meaning no previous data, the lambda will assign the session number as one. This same thing happens before the pie chart data is uploaded as well with the additional adding of the date of upload to allow the data to be separated by month. This same process occurs when CSV files are uploaded from the Raspberry

Pi. While this is happening the JavaScript within the HTML file will display the file's name along with the uploading status indicator. After the lambda returns a response of successful or failed upload through the API the JavaScript will change the indicator to show success or failure. Upon successful upload the size of the CSV file will also be displayed. The style is named style_dragAndDrop.css.

Past Results:

The past results page is straightforward as it displays the pie charts that are separated by month. This means that even if the patient has multiple sessions of data from one month the data will be combined into one chart. The page allows for filtering by months and years. The JavaScript for this page contains five methods. The first method is fetchPastResults(). This method instantly occurs when being redirected to the page and takes the username of the patient and calls the API GetPieChartData to trigger the lambda function that will collect all data under that username regardless of when the data occurred. At the end of this function the next method is called, createPieChart(). This is where the actual visualization of the pie chart is made. Taking the data collected by lambda the method will take the number of normal and ITW steps and devide them by the total number of steps taken to create the pie chart. This occurs for how many months the user has step data for. The next method is applyFilters(). This method is triggered by the event listener that watches if the user applied filters. The dates selected are given to the next method fetchFilteredResults(). This method does the same thing as fetchPastResults() but it gives the lambda function a time constraint so only the data within the dates are displayed. The next method is resetFilters(). This method will wipe the current dates from the filter menu and will call fetchPastResults() to retrieve all the results again.

Doctor Dashboard:

The doctor's dashboard is a list of their patients. The user has options of what they can do on this page. The list of patients has a view button that will take the doctor to that patient's page and has a drop-down menu that allows the doctor to remove a patient from their patient's list. The doctor also has the option to add a patient by typing in the patient's username and clicking the add button. Both functionalities are handled by an API call that will trigger. The first API is RemovePatient that will take the username of the patient and send it to the lambda function that will go into the doctors and patients table in the database and remove the patient from the list of that doctors' patients in the doctors' table and will remove the patients listed doctor and return a response code. The other API is called AddPatient and will take the username of the patient and check if that patient already has a doctor. If yes, then a response code will be sent back explaining the failure. If the patient does not have a doctor, the patient's table will be updated with the doctor's username and the doctor's patient list will be updated within the doctor's table in the database and a response code will be returned. All responses will be shown to the user through a small pop-up window.

Patient Dashboard:

Lastly, the patient dashboard. This is the largest of all the pages in terms of code length and number of functionalities. When a user enters the page after logging in multiple triggers go off in order. The First API to be called is GetPatientData which functions by taking the username from the login page and the lambda function searches the database for the patients information within the patient table in the database. Upon finding the patient all information from the table is returned except for the password, security question and answer. This information is then used to fill out the first two sections of the patient dashboard: Patient Information and Practitioner Information. The name of the patient will be displayed as well as their last data upload date along with more information. This will also be searching for the doctor's information from the doctor's table. The lambda will take the username of the doctor that is stored in the patient table and will complete a second search in the doctor's table to take the information. Next the goals is loaded onto the patient page. This will occur one of two ways based on the type of user that is accessing the page. If the patient is accessing the page the method loadGoals() will call the API GoalHandler and send the patients username to the lambda that will use it to search the patient_goals table and will retrieve all the goals for that patient and whether they are completed or not and send them back through the API. After the goals are fetched the goals will be saved in a list that is displayed through the creation of the goalsContainer. Each goal will have a check box next to it and if the goal is completed the check box will be checked if not it will be left blank. If the user is a doctor the same API will be called in the same way but the method will function differently instead of just displaying the checklist the addGoalForm will be created. This form allows doctors to add, remove or change the status of goals. If the patient has no goals the corresponding container for user type will still be created. If the doctor chooses to add a goal the method addGoal() will be executed. It will take the text of the goal and the username of the patient that is associated with the goal and send it to the GoalHandler API and then to the lambda that will update the goals table to include the new goal. If The goal is marked completed or unmarked completed the method updateGoal() will execute sending the goal, and patient username, and updated statis of the goal to the same API GoalHandler. The lambda will then find the goal in the table and update the status of the goal. The last one is if a goal is deleted by a doctor and this will trigger the deleteGoal() method. This will create a prompt asking if the user is sure they want to delete the goal. If yes then the GoalHandler will be called and sent the goal and patient username that will be used by the lambda to delete. Then loadGoals() is recalled to refresh the goal list. Next the fetchAndDisplayPieChartData() will trigger. This is the exact same code that is completed in the Past Results page along with the CreatePieChart() that is called by fetchAndDisplayPieChartData(). Since the functionality has already been explained we will move on to the final trigger. Lastly the fetchAndDisplaySessionData() method within the JavaScript of the HTML file will trigger. This function will handle the graph that is displayed only to the doctor and will only be generated if the user type is doctor. The API ITW_GetCSV will be called and given the patients username to pass to the lambda function. The lambda will then go to the patient_session_data table within the database and retrieve all the patient's data for the last session and will create a graph using the data. The accelerometer data is on the X axis and time is on the Y axis with EMG points on the graph. This method also handles the UI while loading it will display the message "Loading patient data…" with a loading wheel while the data is retrieved, and the graph is created. Once the graph is ready the message and loading wheel will be deleted and replaced by the graph. The patient dashboard if accessed by a doctor triggers the UI to display certain things that are hidden from the patient. Those items are the graph and the go back to clinician dashboard button.

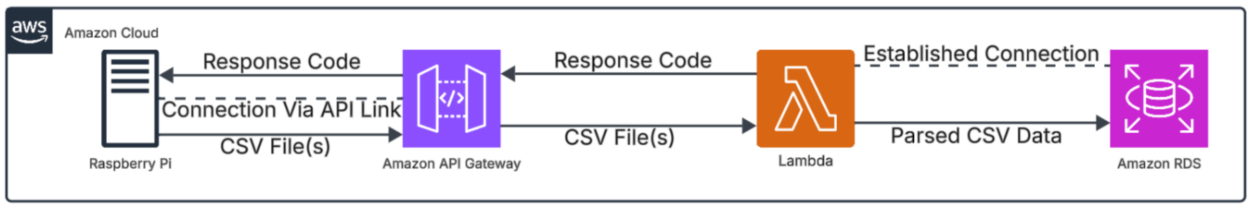


Figure 2

In Figure 2 the connection from the Raspberry Pi to the database is depicted. The Pi contains python code that creates a simple GUI that the user can access and select start monitoring this will monitor a folder on the Pi called "CSV Data". After a patient completes a session with the SmartStride device data is sent via Bluetooth from the ESP 32 to the Raspberry Pi and on the Pi, it is run through the machine learning code to be filtered and segmented and the extracted data is saved as a CSV file in the CSV Data folder. When a new CSV enters the folder, it will trigger the logic of the python code and send that CSV to the API DragDropHandler. This is the same CSV used on the Drag and Drop webpage so it will not be explained again as the functionality is the exact same. The Pi app has a message board where the user will get updates of what the app is doing. Firstly, it will state that it has begun monitoring then it will display the file if a new CSV file is found and send it to the API when the API sends a response back to the Pi app it will display if it was successfully uploaded or not. If the user clicks stop monitoring the message "Monitoring completed" will be displayed on the message board.

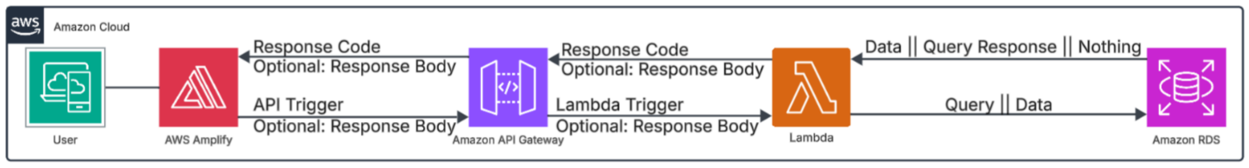


Figure 3

In figure 3 you can see the basic structure of how the website communicates with the database through the lambda and API. First an API call must be done using the APIs endpoint. The endpoint acts like an address to identify what API is being called. Each API contains a trigger. The trigger is what calls the lambda function and makes it execute. All responses are created within the lambda code and the response body type that is allowed is assigned in the API Gateway. All access control headers are also assigned in the API Gateway. This determines what URLs are allowed to call an API. Currently all APIs have the control header of '*' meaning any URL is allowed to call the API. This is what allows the Pi app to call the API. All lambdas need to be given access to the database through IAM roles and VPCs and subnets. These three items control what has access to any service in AWS. Meaning if the VPC of the RDS database is 3 then the lambda must also have the VPC of 3 assigned to it. Of course, the actual VPCs have more complex names, and it is important to keep track of what VPC is assigned where. The lambda must also have the IAM Role of VPC meaning the lambda can have and use the VPC. The subnets must also be added to the lambda in a similar way as the VPCs. Subnets are a more exact version of the VPCs. A single VPC can have multiple subnets attached to it in a parent child relationship. So, if VPC 3 is assigned to a lambda subnet 1 or 2 need to be assigned to the lambda as well. The most important VPC is default (sg-0929797a1b9d70358). This is the VPC that allows a developer to add their IP address to the security group's inbound rules. The security groups can be managed through the EC2 menu in AWS. EC2 is the managing service for all VPCs, subnets and security groups.

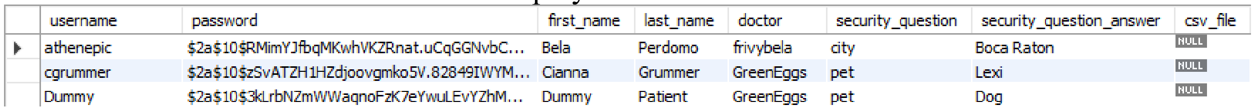
IMPORTANT: Inbound rule sgr-0f22ce765d538aaa0 CANNOT be deleted. Deleting this inbound rule will prevent all access to the database and the website will be completely unfunctional.

To update the HTML, CSS, or JavaScript contained in a HTML file all files must be zipped in a folder then the zip folder will need to be deployed through the Amplify service. This will take a few minutes to deploy. The name of the zip folder does not matter.

The CloudWatch Logs is the console for the lambda functions each lambda has its own log group that can be accessed, and every use of the lambda will be recorded in CloudWatch and can be reviewed for statistical or debugging work.

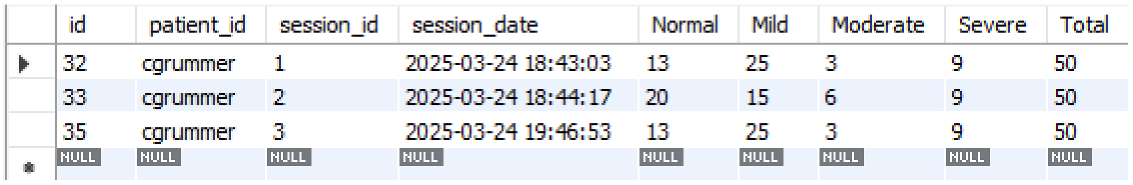
Amazon Web Services has many tutorials on their website to help guide the development of the SmartStride website.

The RDS database structure is displayed below.



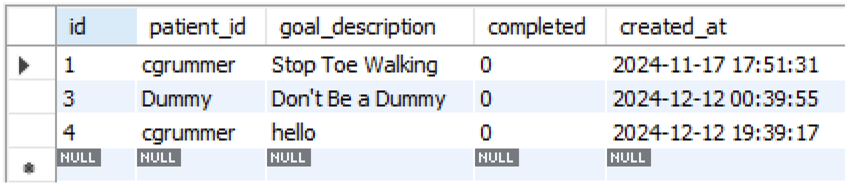
username	password	first_name	last_name	doctor	security_question	security_question_answer	csv_file
athenepic	$2a$10$RMimYJfbqMKwhVKZRnat.uCqGGNvbC...	Bela	Perdomo	frivybela	city	Boca Raton	NULL
cgrummer	$2a$10$zSvATZH1HZdjoovgmko5V.82849IWYM...	Cianna	Grummer	GreenEggs	pet	Lexi	NULL
Dummy	$2a$10$3kLrbNZmWWaqnoFzK7eYwuLEvYZhM...	Dummy	Patient	GreenEggs	pet	Dog	NULL

Patients table.



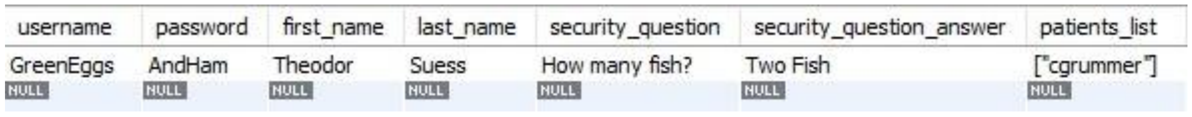
id	patient_id	session_id	session_date	Normal	Mild	Moderate	Severe	Total
32	cgrummer	1	2025-03-24 18:43:03	13	25	3	9	50
33	cgrummer	2	2025-03-24 18:44:17	20	15	6	9	50
35	cgrummer	3	2025-03-24 19:46:53	13	25	3	9	50
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Pie chart data table.



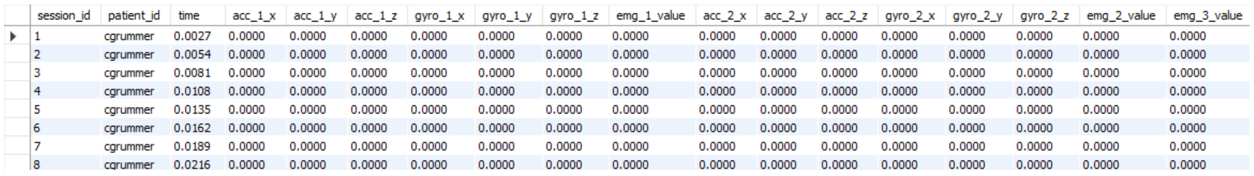
id	patient_id	goal_description	completed	created_at
1	cgrummer	Stop Toe Walking	0	2024-11-17 17:51:31
3	Dummy	Don't Be a Dummy	0	2024-12-12 00:39:55
4	cgrummer	hello	0	2024-12-12 19:39:17
NULL	NULL	NULL	NULL	NULL

Patient goals table.



username	password	first_name	last_name	security_question	security_question_answer	patients_list
GreenEggs	AndHam	Theodor	Suess	How many fish?	Two Fish	["cgrummer"]
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Doctors table.



session_id	patient_id	time	acc_1_x	acc_1_y	acc_1_z	gyro_1_x	gyro_1_y	gyro_1_z	emg_1_value	acc_2_x	acc_2_y	acc_2_z	gyro_2_x	gyro_2_y	gyro_2_z	emg_2_value	emg_3_value
1	cgrummer	0.0027	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	cgrummer	0.0054	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	cgrummer	0.0081	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
4	cgrummer	0.0108	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	cgrummer	0.0135	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
6	cgrummer	0.0162	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
7	cgrummer	0.0189	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
8	cgrummer	0.0216	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Patient session data table.