**Scheduling program**

1. **Description**

The program will be used to schedule meeting between provider and customer for example dog walker and dog owner or a doctor and a patient.

The doctor will have a calendar with days and slots.

The patient will have slots to choose from and will select the slot.

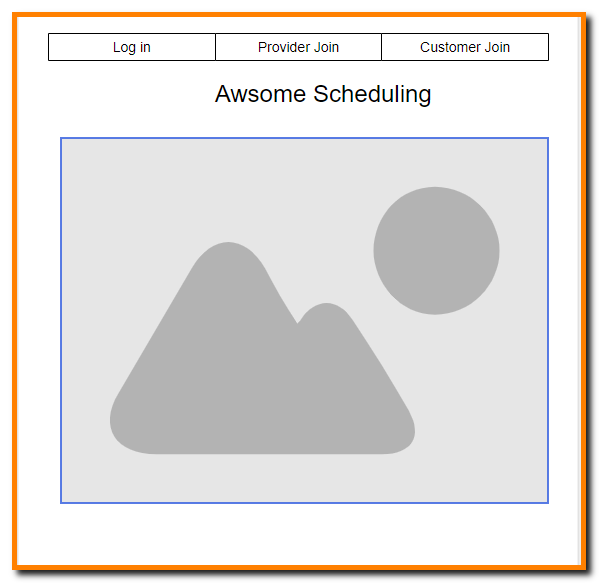
1. **Scenarios**
2. Provider Join
3. Starts the program.
4. Selects the Menu option of Provide Join
5. Enters the data of the provider.
6. Opens Provider dashboard.
7. Customer Join
8. Starts the program.
9. Selects the Menu option of Customer Join
10. Enters the data of the customer.
11. Opens Customer dashboard.
12. Setting a session

Done by a customer.

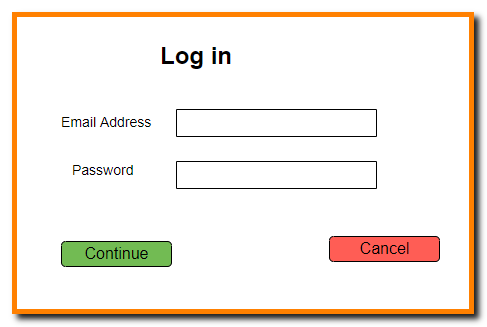
1. Log in using email and password.
2. The program opens on the customer dashboard.
3. Selects a new session
4. Enters the data.
5. Cancelling a session

Done by customer.

1. Log in using email and password.
2. Program opens customer dashboard.
3. Selects the session to be deleted.
4. Deletes the session.
5. Managing provider sessions
6. Log in using email and password.
7. Program opens Provider Dashboard.
8. If needed deletes a session.
9. Sending reminders
10. When a session is scheduled by a customer, the program sends an email to the provider
11. When a session is canceled by a customer, the program sends an email to the provider.
12. When a session is cancelled by a provider, the program sends an email to the customer.
13. When a session is about to start, the program sends an email to both customer and provider.
14. **Records**
15. Provider
16. Customer
17. Service
18. Session
19. **Screens**
20. Main Screen
21. Provider Joining Screen
22. Customer Joining screen
23. Login
24. Provider Dashboard.
25. Customer Dashboard.
26. Session
27. Selecting Service
28. Selecting Provider
29. Selecting Session
30. **Main screen**



1. Navigation
2. Log in: opens log in screen
3. Provider Join: opens Provider Join Screen
4. Customer Join: opens Customer Join Screen
5. Exit:
6. **Log in Screen:**



1. Continue button:

The program looks for the user if not found gives an error message,

Else if the user is customer, it opens Customer screen if the user is Provider, it opens provider screen.

1. Cancel button:

Returns to Main Screen.

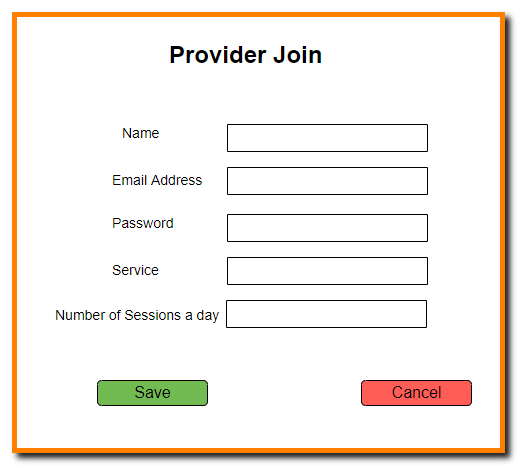
1. **API Calls**
2. Name: logIn
3. Description: Common to customer and provider determines if the user is in the database.
4. Type: POST
5. Request : Email Address.
6. Action: finds out if the user is in the Provider or the Customer table or does not exist.

If there is no user return an error message that should be handled in the front.

1. Response : id: , Type: Customer or Provider
2. Err: No user found

/api/login/

1. Name: getCustomerSessions
2. Description: Provides the list of the customer sessions.
3. Type: Get
4. Request: id
5. Action: finds all the sessions of the Customer.
6. Response : id: , Array of Provider Name, Service Name, Date, Slot.
7. Err:
8. Name: getProviderSessions
9. Description: Provides the list of the provider sessions.
10. Type: Get
11. Request: id
12. Action: finds all the sessions of the Provider.
13. Response : id: , Array of Customer Name, Date, Slot.
14. Err:
15. **Provider Join**



1. Save button:

The program verifies that the record is valid if not valid gives an error message,

Else opens the Provider screen.

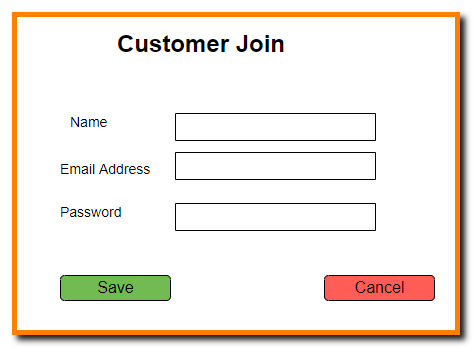
If the Service is not in the Service Table, the program adds it.

When saving, the Service ID is saved to the provider table.

1. Cancel button:

Returns to Main Screen.

1. **API Call**
2. Name: providerSave
3. Type: Post
4. Request : Name, Email Address, Service, Slots.
5. Action: Creates a new Provider record. If the service does not exist creates a new service record. Replaces the service name with service id
6. Response : list of
7. Err:
8. **Customer Join**



1. Save button:

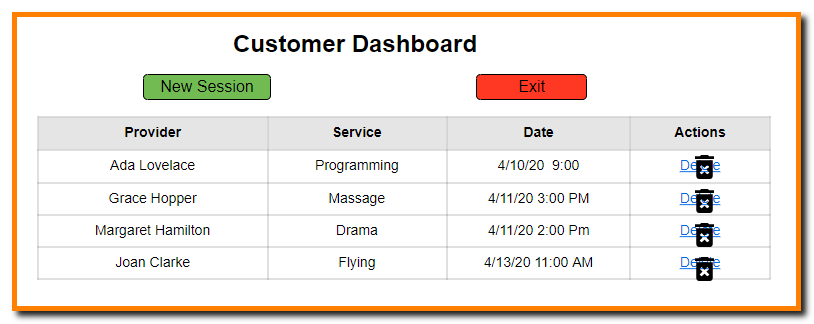
The program verifies that the record is valid if not valid gives an error message,

Else saves and opens the Customer screen

1. Cancel button:

Returns to Main Screen.

1. **API Call**
2. Route: /savecustomer
3. Type: Post
4. keys : name, email
5. Action: Creates a new customer record.
6. Response : [] empty list of customer sessions:
7. Error: "error": "email: awsome@gmail.com is already included, choose a different email.
8. **Customer Dashboard**



1. State Field:

Customer ID

Name

1. List of sessions each Session includes
2. List of grid items
3. Provider.
4. Service
5. Date
6. slot
7. Buttons
8. New Session: Runs the getServices API and opens the New Session Screen.
9. Exit Return to the main menu.
10. Delete: Deletes the session.
11. API Calls
12. **API Calls**
13. Name: deleteSession
14. Description: Provides the list of the customer sessions.
15. Type: Delete
16. Request: Session id
17. Action: deletes the session.
18. Response : res.status(200).end() 200 indicates success.
19. Err:
20. Name: getServices
21. Description: called when the user clicks on the New Session button. Provides the list of all services
22. Type: GET
23. Request:
24. Action: gets the list of all the ids and the names of all services.
25. Response : Table of id and names of services.
26. Err:
27. **New Session**:
28. Select Service



1. Field

Service: dropdown of all the Services,

1. **Buttons**
2. Cancel: Returns to Customer Dashboard
3. Next: Runs the getProvider API and opens the Select Provider Screen passing the service id.
4. **API Call**
5. Name: getProviders
6. Type: Get
7. Request : service Name
8. Action: gets the all the provider of the service.
9. Response : table of Providers id and names.
10. Err:
11. Select Provider



1. Parameters

Service id and Name

1. Field

Provider: dropdown of all the Provider that provide the service

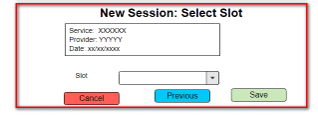
1. Buttons
2. Cancel: Returns to Customer Dashboard
3. Previous: Returns to Select Date Screen.
4. Next: Runs the getDates API and opens the Select Date Screen, passing the service id and provider id
5. API Call
6. Name: getDates
7. Type: Get
8. Request : Provider Name
9. Action: gets the list of all the dates in the next week that the Provider has slots.
10. Response : Array of Dates:
11. Err:
12. Select Date



1. Parameters
2. Service id and Name
3. Provider id and Name
4. Field

Date: dropdown of all the Providers available dates in the next week.

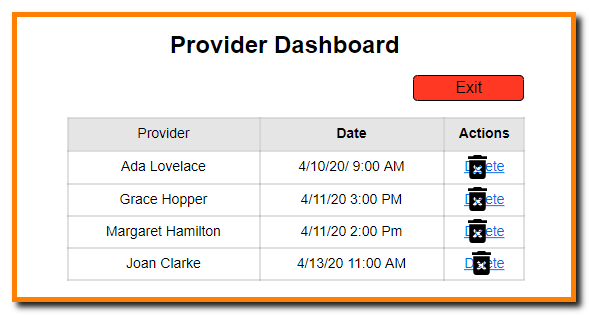
1. Buttons
2. Cancel: Returns to Customer Dashboard
3. Previous: Returns to Select Provider Screen.
4. Next: Runs the getSlots API and opens the Select Slot Screen, passing Service id and Provider id
5. API Call
6. Name: getSlots
7. Type: Get
8. Request : Provider Name, Date
9. Action: gets the list open slots in the date .
10. Response : Array of slots
11. Err:
12. Select Slot



1. Parameters
2. Service id.
3. Provider id.
4. Date
5. Field

Slot A drop down of available slots on the date of the provider

1. Buttons
2. Cancel: Returns to Customer Dashboard
3. Previous: Returns to Select Date Screen.
4. Save: Runs the Save Session API and opens the Select Slot Screen
5. API Call
6. Name: saveSession
7. Type: Post
8. Request : Provider id, Customer id, Date, Slot
9. Action: Saves a new session .
10. Response : Array of slots
11. Err:
12. **Provider Dashboard**



1. List of sessions each Session includes
2. Customer
3. Date
4. Slot
5. Button for delete
6. Buttons
7. Exit Return to the main menu.
8. Delete: Deletes the session.
9. API: deleteSession
10. **Database records**
11. Provider
12. id
13. name
14. email
15. service\_id
16. daily\_slots.
17. Customer
18. id
19. name
20. email
21. Session
22. id
23. customer\_id
24. provider\_id
25. service\_id
26. date
27. slot
28. Services
29. id
30. name

query all provider

for each one, go into services

loop through services array to see if any matches service id

return those provider

vs

query services

return all matching providers

1. Comments
2. Id of user is the email address
3. Future expansion is using Zoom API to schedule a Zoom conference for each session so when creating a session, the program will create a zoom conference and send the invite to the provider and the customer.

ReactDate: <https://www.npmjs.com/package/react-datepicker>

React stepper: <https://www.npmjs.com/package/react-stepper-horizontal>

React Outline

1. Containers – class components, ComponentDidMount loads user data from db
   1. Navbar – “const location = useLocation()”, active classes
   2. Home – Landing page, app information, user instructions, links to about us and get started
   3. About – Talks about the project, link to repo, link to github profiles
   4. Dashboard – After a user is signup and booked, this is where they see their exisiting appointments
   5. Provider Registration – Stepper form, 2 steps
      1. Provider screen
      2. Provider Dashboard.
   6. Customer Registration
      1. Customer screen
      2. Customer Dashboard
   7. New Session
      1. Select Service
      2. Select Provider
      3. Select Session.
   8. Login
      1. Login screen
      2. Dashboard.
      3. Pick Account Type – select customer or provider
      4. Signup/Login – email password
      5. User Details – render form for customer or provider
      6. Pick Services – query all services
      7. Pick Provider – query all provider for selected services
      8. Schedule Time – query all time slots for selected provider
      9. Confirmation – showing booked appointment
      10. Redirect to Dashboard
   9. Scheduling
      1. Pick Provider + Pick Services
      2. Schedule Time
      3. Confirmation
      4. Redirect to Dashboard
2. Components
   1. Header – app title
   2. Footer – github usernames
   3. PageTitle
   4. Form – versioned out for each registration step
   5. Appointment – render each scheduled session (on confirmation page or on dashboard)
   6. Calendar – show all scheduled Appointments (render component for each session)
   7. Profile – after user enters all information, a card to display user’s information
   8. Scheduler – date picker to actually select calendar date value for appointment

React Folder Structure

* Client
  + Public
  + Src
    - Components
      * Header
        + Index.js
        + Styles.css
    - Containers
      * Home
        + Index.js
        + Styles.css
      * Dashboard
        + Index.js
        + Styles.css
      * About
        + Index.js
        + Styles.css
      * Navbar
        + Index.js
        + Styles.css
      * Registration
        + Index.js
        + Styles.css

REPO SETUP

1. Clone manny’s boilerplate
2. Delete .git file (rm -rf .git)
3. Create new repo
4. Set up collaborators on github
5. Set up master branch rules on github
6. Link repo and local folder ( get init…)
7. Add .gitignore (node\_modules, package-lock.json, DS\_Store, .env, eslint file as well)
8. Install Travis and link to github – install in server
9. Install ESLint – install as dev dependency on server ( -dev)
10. Apply Airbnb eslint style – see link in slack – each person does it
11. Install and setup dotenv (environment variables and secret api keys)
12. Create and update .env file in root folder (each person)
13. Setup JawsDB on Heroku (live mysql db)
14. Create a Heroku app
15. Push to Heroku
16. Add collaborator on Heroku
17. Pick react library for date picker and install inside client folder

Switch( case)

Case 1:

Return (

<RegType /> // Step1

)

Form Component

Const CustomerForm = [

{

Label: “”,

Validate: [],

Name: “”

}

]

Const ProviderForm = [

{

Label: “”,

Validate: [],

Name: “”

}

]

Function Form() {

Return (

If (regType === “customer”) {

Return <Form fields={customerForm} />

}

If (regType === “provider”) {

Return <Form fields={providerForm} />

}

)

}

Changes

1. Add password

DROP DATABASE IF EXISTS schedule\_db;

CREATE DATABASE schedule\_db;

USE schedule\_db;

CREATE TABLE provider (

  id INT AUTO\_INCREMENT NOT NULL,

  name VARCHAR(60) NOT NULL,

  email VARCHAR(60) NOT NULL,

  service\_id INT,

  daily\_slots INT,

  PRIMARY KEY (id)

);

CREATE TABLE customer (

  id INT AUTO\_INCREMENT NOT NULL,

  name VARCHAR(60) NOT NULL,

  email VARCHAR(60) NOT NULL,

  PRIMARY KEY (id)

);

CREATE TABLE session (

  id INT AUTO\_INCREMENT NOT NULL,

  customer\_id INT NOT NULL,

  provider\_id INT NOT NULL,

  service\_id INT NOT NULL,

  date DATE NOT NULL,

  slot INT NOT NULL,

  PRIMARY KEY (id)

);

CREATE TABLE service (

  id INT AUTO\_INCREMENT NOT NULL,

  name VARCHAR(50) NOT NULL,

  PRIMARY KEY (id)

);

Questions:

1. Empty session list should it be empty table or error.
2. No user found should it be 0 or error