

---

**NAME :- Pratham Bharat Madhani.**

**BTech Computer Engineer Student at MIT Academy of Engineering, Pune.**

**Problem Statement:**

**Assume that you are the CTO for the outsourcing firm which has been chosen to build an admission form for the Yoga Classes which happen every month.**

**Requirements for the admission form are:**

- Only people within the age limit of 18-65 can enroll for the monthly classes and they will be paying the fees on a month on month basis. I.e. an individual will have to pay the fees every month and he can pay it any time of the month.**
  - They can enroll any day but they will have to pay for the entire month. The monthly fee is 500/- Rs INR.**
  - There are a total of 4 batches a day namely 6-7AM, 7-8AM, 8-9AM and 5-6PM. The participants can choose any batch in a month and can move to any other batch next month. I.e. participants can shift from one batch to another in different months but in same month they need to be in same batch**
- 

**Assumptions and Understanding of Problem Statement :-**

1. I have taken two forms instead of taking one form, So the first form (Enroll Form) is for registering the yoga classes participants and for paying monthly fees and the second form is for updating the batch but one time in a month.
2. According to the problem statement, my understanding and assumptions are :-
  - a. If a person enrolls in the middle of the month (let us take the 15th of the month) then we will not give him a choice to change the batch as already he has made the choice at the time of filling the form.
  - b. At every first day of the month, the participants will get the chance to change their batch but only 1 chance in the whole month, So I have maintained a flag attribute or variable for the same in the database (In table participants, batch\_changed maintains whether the participants has changed the batch this month or not).
  - c. At every first day of the month, every participant has to pay full month yoga classes fees, that is whether he has joined on any day in the previous month, he has to pay monthly fees (i.e 500) at every new month.
  - d. I'm also assuming that the participant will pay at any day of the month and this month's fee will not be carried forward to the next month.

## **Tech - Stack of Yoga classes project :**

1. Frontend :- Reactjs.
2. Backend :- Nodejs, Expressjs.
3. Database :- MySQL (workbench).

## **Database Design :-**

**Name of the Database :- yoga\_classes.**

**Tables in the Database yoga\_classes :-**

1. **Participants.**
2. **Payments.**
3. **Batch.**

**Let us discuss the purpose and use of each and every attribute of the table in detail.**

### **1. Participants Table :-**

**Table creation MySQL code :-**

```
CREATE TABLE `yoga_classes`.`participants` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `firstName` varchar(30) NOT NULL,  
  `lastName` varchar(30) NOT NULL,  
  `email` varchar(50) NOT NULL,  
  `age` int NOT NULL,  
  `address` varchar(150) NOT NULL,  
  `city` varchar(50) NOT NULL,  
  `state` varchar(50) NOT NULL,  
  `country` varchar(50) NOT NULL,  
  `gender` varchar(10) NOT NULL,  
  `batch_id` int NOT NULL,
```

```

`payment_id` int NOT NULL,

`batch_changed` int DEFAULT 1 NOT NULL,

`batch_changed_date` DATE NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (batch_id) REFERENCES batch(batch_id),

FOREIGN KEY (payment_id) REFERENCES payments(payment_id)

);

```

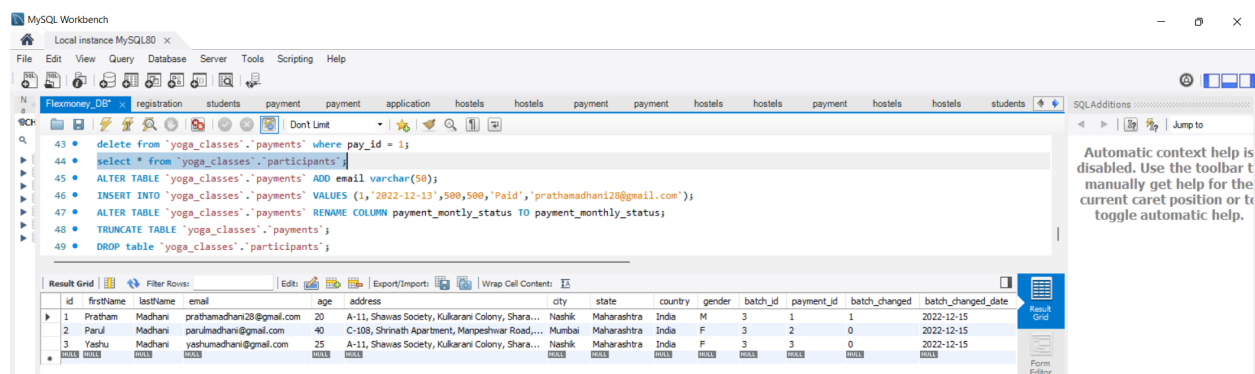
Participants table will store all the information about the participants. It will store all the necessary information about the participants like name, email, age, address, city, state, country, gender, batch\_id, etc.

The two attributes **batch\_id** and **payment\_id** are the foreign keys. The **batch\_id** references the table batch primary key and **payment\_id** references the table payments primary key.

**batch\_changed** which is a flag field or attribute I have used to keep a record of whether the participants have changed the batch in this particular month or not. If **batch\_changed** value is 0, means the participant has the chance to change the batch of this month and if **batch\_changed** value is 1, means the participant has used the chance to change the batch for this month.

The id in the participants table is the primary key of the participant table.

## MySQL workbench image of Participants table :-



The screenshot shows the MySQL Workbench interface. The top pane displays SQL queries for creating and managing the 'participants' table. The bottom pane shows the 'Result Grid' with the following data:

id	first_name	last_name	email	age	address	city	state	country	gender	batch_id	payment_id	batch_changed	batch_changed_date
1	Pratham	Madhani	prathamadhani28@gmail.com	20	A-11, Shivas Society, Kulkarni Colony, Shara...	Nashik	Maharashtra	India	M	3	1	1	2022-12-15
2	Parul	Madhani	parulmadhani@gmail.com	40	C-108, Shrinath Apartment, Manpeshwar Road,...	Mumbai	Maharashtra	India	F	3	2	0	2022-12-15
3	Yashu	Madhani	yashumadhani@gmail.com	25	A-11, Shivas Society, Kulkarni Colony, Shara...	Nashik	Maharashtra	India	F	3	3	0	2022-12-15

## 2. Payments Table :-

### Table creation MySQL code :-

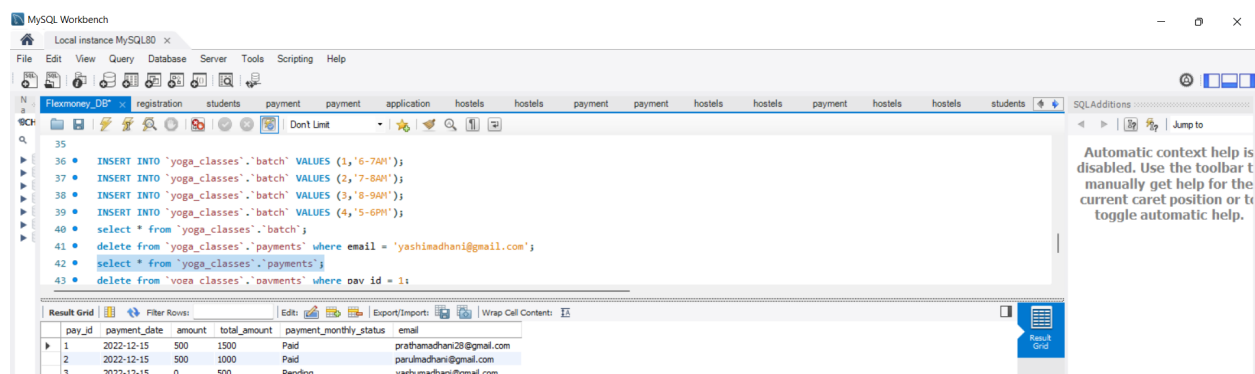
```
CREATE TABLE `yoga_classes`.`payments` (  
  
  `pay_id` int NOT NULL AUTO_INCREMENT,  
  
  `payment_date` DATE NOT NULL,  
  
  `amount` int NOT NULL,  
  
  `total_amount` int NOT NULL,  
  
  `payment_monthly_status` varchar(20) DEFAULT 'Pending',  
  
  PRIMARY KEY(`pay_id`)  
  
);
```

Payments table will keep a track of the payments paid by the participants till date. It will store the details related to the payments like payment date, amount, total\_amount, payment monthly status, etc.

**pay\_id** is the primary key of the payments table. **payment\_date** is the date when the participant will pay the yoga classes monthly fees. amount is the field which keeps the track of money or fees paid by the participant in this month i.e 500. total\_amount will keep the track of total amount fees or payments done to the participants till this month.

**payment\_monthly\_status** is also a flag variable that will denote whether the participants have paid the fees for the month for the month or not. Default value of this field is 'Pending'. If payment\_monthly\_status value is 'Paid' means the participants have paid the fees for this particular month and if its value is 'Pending' means the participants have not the fees for this particular month.

### MySQL workbench image of Payments table :-



The screenshot displays the MySQL Workbench interface. The SQL editor contains the following queries:

```
35  
36 • INSERT INTO `yoga_classes`.`batch` VALUES (1,'6-7AM');  
37 • INSERT INTO `yoga_classes`.`batch` VALUES (2,'7-8AM');  
38 • INSERT INTO `yoga_classes`.`batch` VALUES (3,'8-9AM');  
39 • INSERT INTO `yoga_classes`.`batch` VALUES (4,'5-6PM');  
40 • select * from `yoga_classes`.`batch`;  
41 • delete from `yoga_classes`.`payments` where email = 'yashimadhani@gmail.com';  
42 • select * from `yoga_classes`.`payments`;  
43 • delete from `yoga_classes`.`payments` where pay_id = 1;
```

The Result Grid shows the following data:

pay_id	payment_date	amount	total_amount	payment_monthly_status	email
1	2022-12-15	500	1500	Paid	prathamadani28@gmail.com
2	2022-12-15	500	1000	Paid	parulmadhani@gmail.com
3	2022-12-15	0	500	Pending	yashumadhani@gmail.com

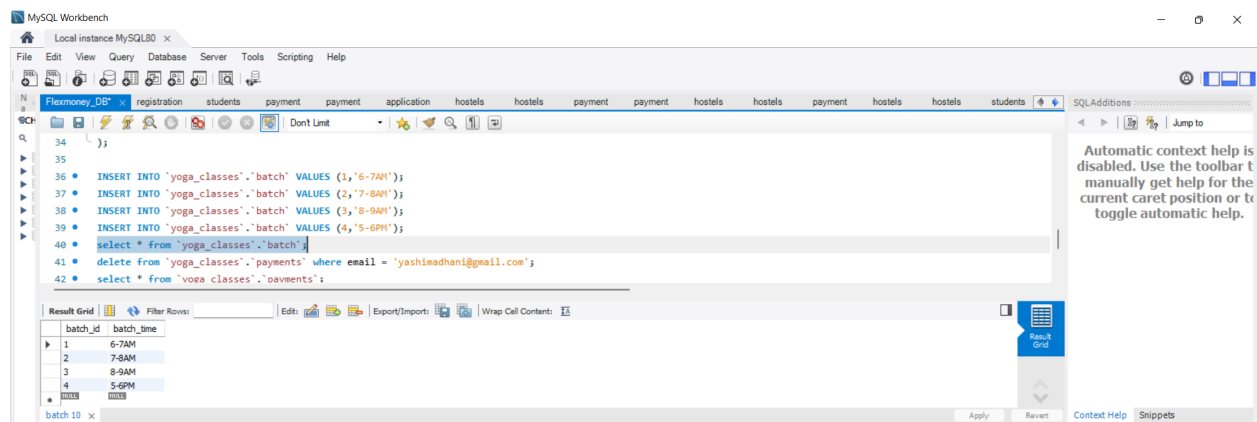
### 3. Batch Table :-

#### Table creation MySQL code :-

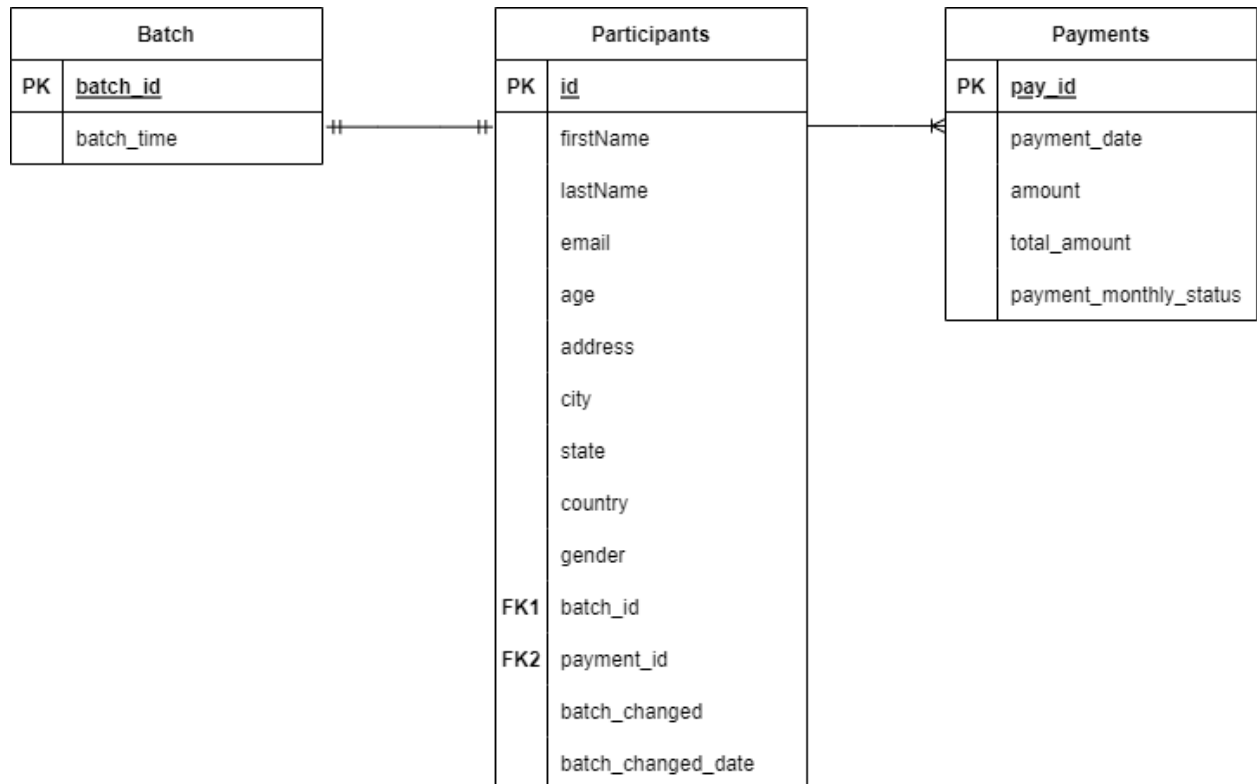
```
CREATE TABLE `yoga_classes`.`batch` (  
  `batch_id` int NOT NULL AUTO_INCREMENT,  
  `batch_time` varchar(20) NOT NULL,  
  PRIMARY KEY(`batch_id`)  
);
```

Batch table will keep the record of all the batch timing. batch\_id will be the primary key of the batch table. **batch\_time** will be the batch timings that are mentioned in the problem statement (i.e it consists of 4 batches with batch timings as 6-7AM, 7-8AM, 8-9AM, 5-6PM).

#### MySQL workbench image of Batch Table :-



## ER diagram of the database design :-



## Front-End :-

I have built the front-end of this project in Reactjs as mentioned in the tech-stack of the project as well. I have made two forms, Enroll Form is the form which is used for registration of yoga classes participants and it can be also used to pay monthly fees of the yoga classes. The update form is used to update batch time of a particular yoga participant.

I have used Formik, Form packages in reactjs to make the enroll and update form. I have used the yup package for the form validation. I have also used react-router package to redirect from one component to another via navbar.

## Enroll Form :-

Yoga Classes Home Update Form

### Enroll Form

**First Name**

**Last Name**


**Email**

**Age**

**Address**

**City**

**State**



Enter email

**Age**

**Address**

**City**


**State**

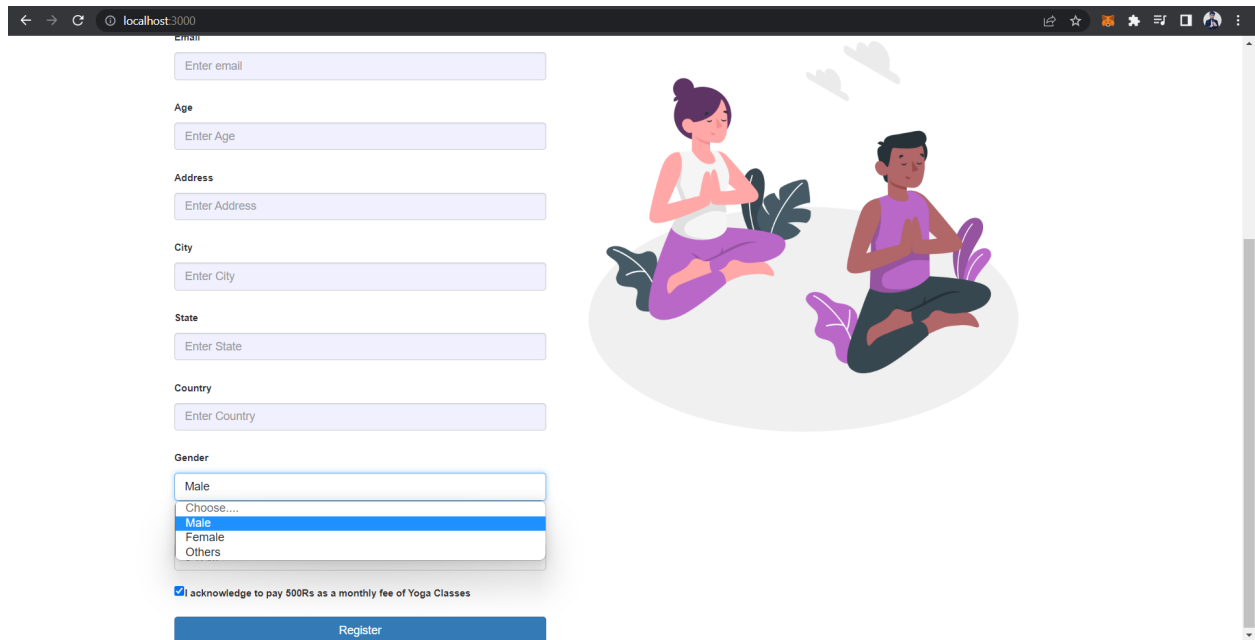
**Country**

**Gender**

**Batch Time**

☒ I acknowledge to pay 500Rs as a monthly fee of Yoga Classes





localhost:3000

Email  
Enter email

Age  
Enter Age

Address  
Enter Address

City  
Enter City

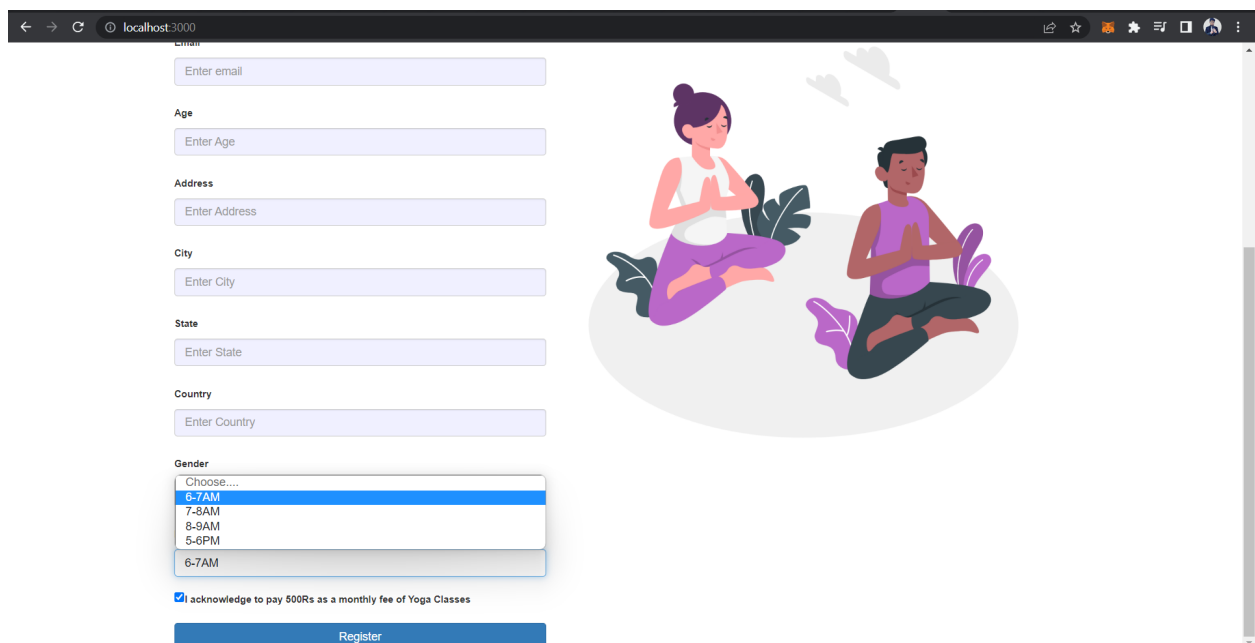
State  
Enter State

Country  
Enter Country

Gender  
Male  
Choose...  
Male  
Female  
Others

☒ I acknowledge to pay 500Rs as a monthly fee of Yoga Classes

Register



localhost:3000

Email  
Enter email

Age  
Enter Age

Address  
Enter Address

City  
Enter City

State  
Enter State

Country  
Enter Country

Gender  
Choose...  
6-7AM  
7-8AM  
8-9AM  
5-6PM  
6-7AM

☒ I acknowledge to pay 500Rs as a monthly fee of Yoga Classes

Register

In the following Enroll form, I have taken the following fields such as First Name, Last Name, Email, Age, Address, City, State, Country, Gender, Batch Time and the acknowledgement from the participants whether he wants to pay Rs 500 as monthly fees or not.

As I have mentioned that I have used yup for the form validation. So I have kept the checkbox as true by default, If the user unchecks the checkbox and tries to submit the form



will not get submitted. If the user input the age less than 18 and age greater than 65 then also the form will not get submitted and it displays the error that age should be greater or equal to 18 and age should be less than or equal to 65 accordingly.

Email is required

Age

12

You must be at least 18 years

Address

Enter Address

City

Enter City

State

Enter State

Country

Enter Country

Gender

Male


Batch Time

6-7AM

Required

☒ I acknowledge to pay 500Rs as a monthly fee of Yoga Classes

Register



Email is required

Age

90

You must be at most 65 years

Address

Enter Address

City

Enter City

State

Enter State

Country

Enter Country

Gender

Male


Batch Time

6-7AM

Required

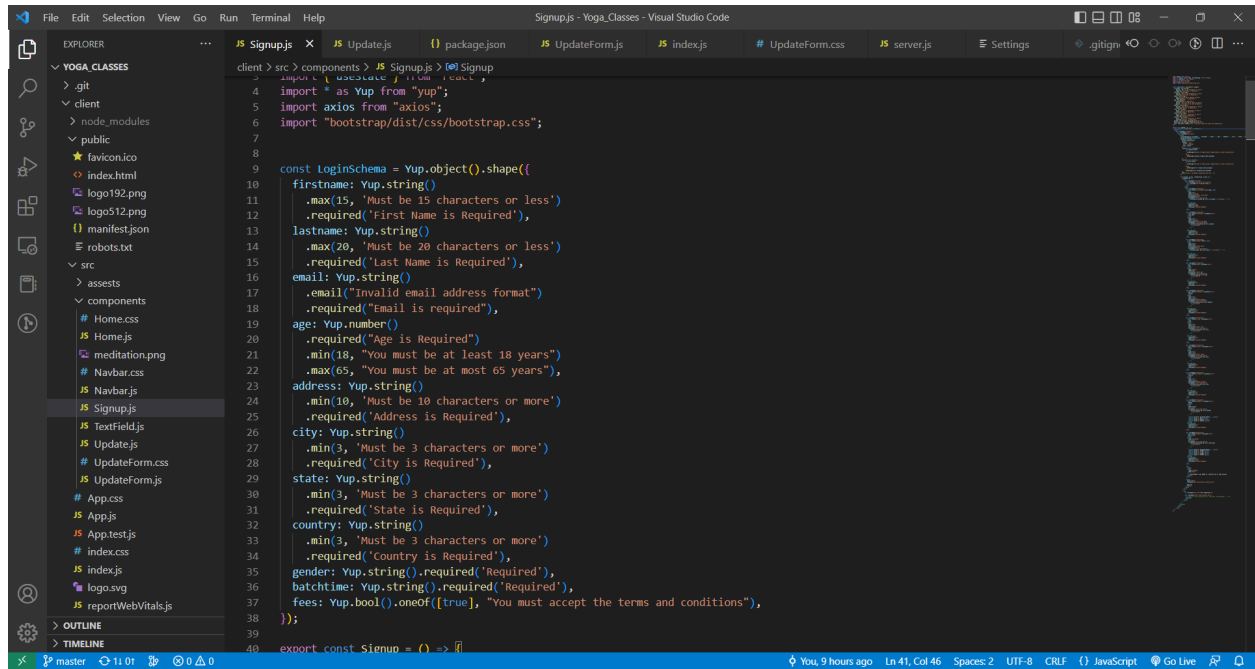
☒ I acknowledge to pay 500Rs as a monthly fee of Yoga Classes

Register



So the form is located in client/components/Signup.js and the Yup object is written with the const LoginSchema. So all the field validation is written in const LoginSchema and formik form just below the LoginSchema, written in export const Signup.

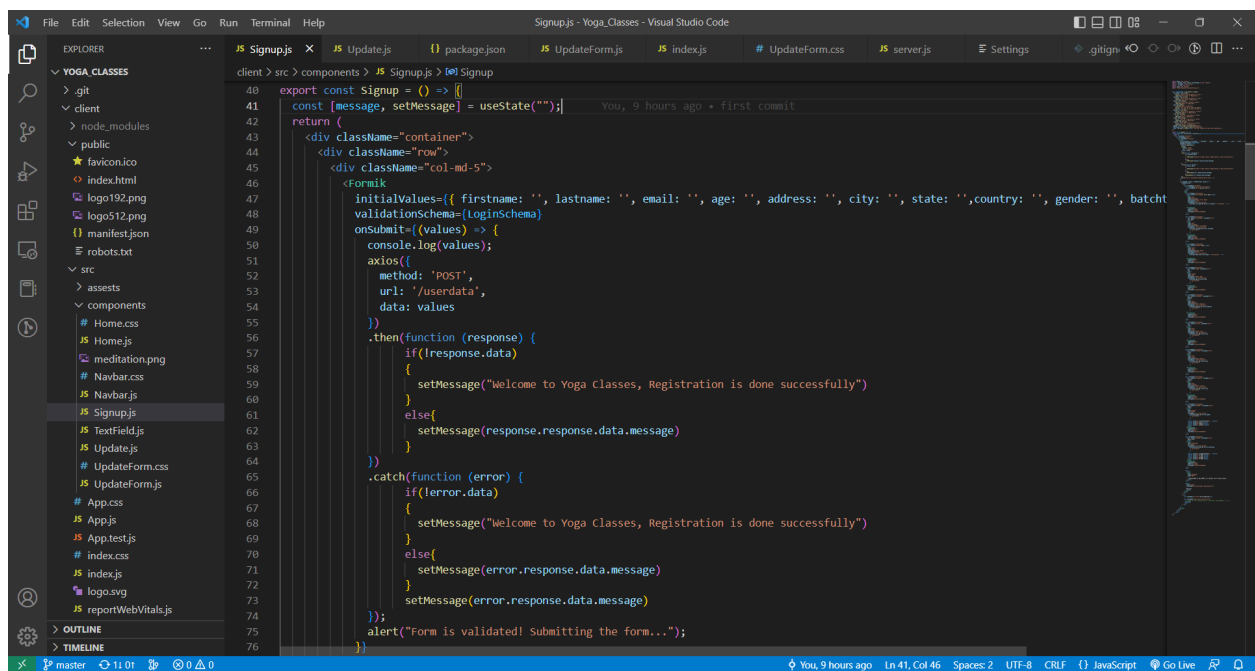
## Yup validation done for the Enroll form :-



The screenshot shows a Visual Studio Code editor with a project named 'Signups - Yoga\_Classes'. The Explorer panel on the left shows the file structure, including a 'client' directory with 'src' and 'components'. The main editor displays the 'Signup.js' file, which defines a Yup validation schema for a login form. The schema includes fields for 'firstname', 'lastname', 'email', 'age', 'address', 'city', 'state', 'country', 'gender', 'batchtime', and 'fees'. The 'fees' field is a boolean with a 'oneOf' rule set to [true]. The schema is used to validate the form data before submission.

```
client > src > components > JS Signup.js > [L] Signup
import { useState } from 'react';
4 import * as Yup from "yup";
5 import axios from "axios";
6 import "bootstrap/dist/css/bootstrap.css";
7
8
9 const LoginSchema = Yup.object().shape({
10   firstname: Yup.string()
11     .max(15, 'Must be 15 characters or less')
12     .required('First Name is Required'),
13   lastname: Yup.string()
14     .max(20, 'Must be 20 characters or less')
15     .required('Last Name is Required'),
16   email: Yup.string()
17     .email('Invalid email address format')
18     .required('Email is required'),
19   age: Yup.number()
20     .required('Age is Required')
21     .min(18, "You must be at least 18 years")
22     .max(65, "You must be at most 65 years"),
23   address: Yup.string()
24     .min(10, 'Must be 10 characters or more')
25     .required('Address is Required'),
26   city: Yup.string()
27     .min(3, 'Must be 3 characters or more')
28     .required('City is Required'),
29   state: Yup.string()
30     .min(3, 'Must be 3 characters or more')
31     .required('State is Required'),
32   country: Yup.string()
33     .min(3, 'Must be 3 characters or more')
34     .required('Country is Required'),
35   gender: Yup.string().required('Required'),
36   batchtime: Yup.string().required('Required'),
37   fees: Yup.bool().oneOf([true], "You must accept the terms and conditions"),
38 });
39
40 export const Signup = () => {}
```

## Formik form :-



The screenshot shows the same Visual Studio Code editor with the 'Signup.js' file. The code now implements the Formik form, including the initial values, validation schema, and the onSubmit handler. The form is rendered using the 'Formik' component from the 'formik' library. The onSubmit handler uses 'axios' to send a POST request to '/userdata' with the form data. The response is handled to show a success message or an error message. The form is validated before submission, and an alert is shown when the form is validated and submitted.

```
client > src > components > JS Signup.js > [L] Signup
40 export const Signup = () => {
41   const [message, setMessage] = useState("");
42   return (
43     <div className="container">
44       <div className="row">
45         <div className="col-md-5">
46           <Formik
47             initialValues={{ firstname: '', lastname: '', email: '', age: '', address: '', city: '', state: '', country: '', gender: '', batchtime: '' }}
48             validationSchema={LoginSchema}
49             onSubmit={values => {
50               console.log(values);
51               axios({
52                 method: 'POST',
53                 url: '/userdata',
54                 data: values
55               })
56               .then(function (response) {
57                 if(!response.data) {
58                   setMessage("Welcome to Yoga Classes, Registration is done successfully")
59                 } else {
60                   setMessage(response.response.data.message)
61                 }
62               })
63             }
64             .catch(function (error) {
65               if(!error.data) {
66                 setMessage("Welcome to Yoga Classes, Registration is done successfully")
67               } else {
68                 setMessage(error.response.data.message)
69               }
70             })
71             .then(function () {
72               alert("Form is validated! Submitting the form...");
73             })
74           />
75         </div>
76       </div>
77     </div>
78   );
79 }
```

## Validation Done for all fields :-

[Yoga Classes](#) [Home](#) [Update Form](#)

### Enroll Form

**First Name**

First Name is Required

**Last Name**

Last Name is Required

**Email**

Email is required


**Age**

You must be at least 18 years

**Address**

Address is Required

**City**



← → ↻ localhost:3000

You must be at least 18 years

**Address**

Address is Required

**City**

City is Required

**State**

State is Required

**Country**

Country is Required

**Gender**

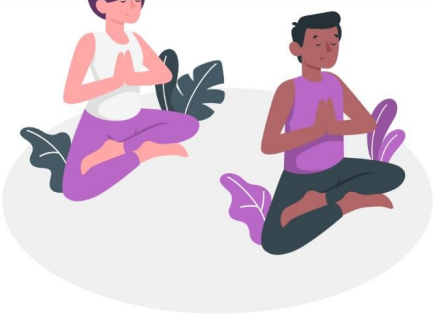
Required

**Batch Time**

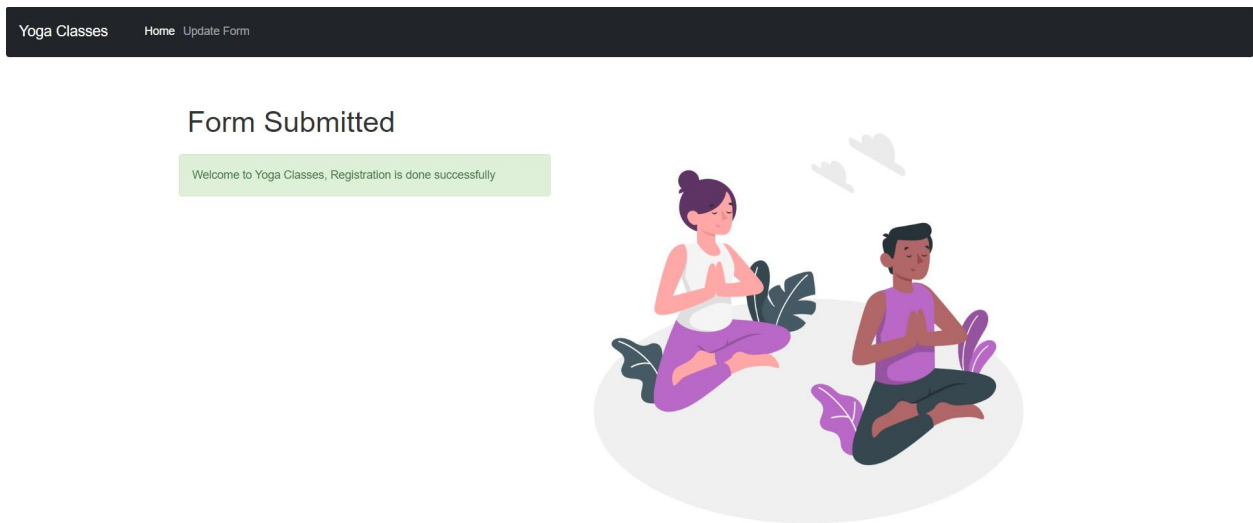
Required

☒ I acknowledge to pay 500Rs as a monthly fee of Yoga Classes

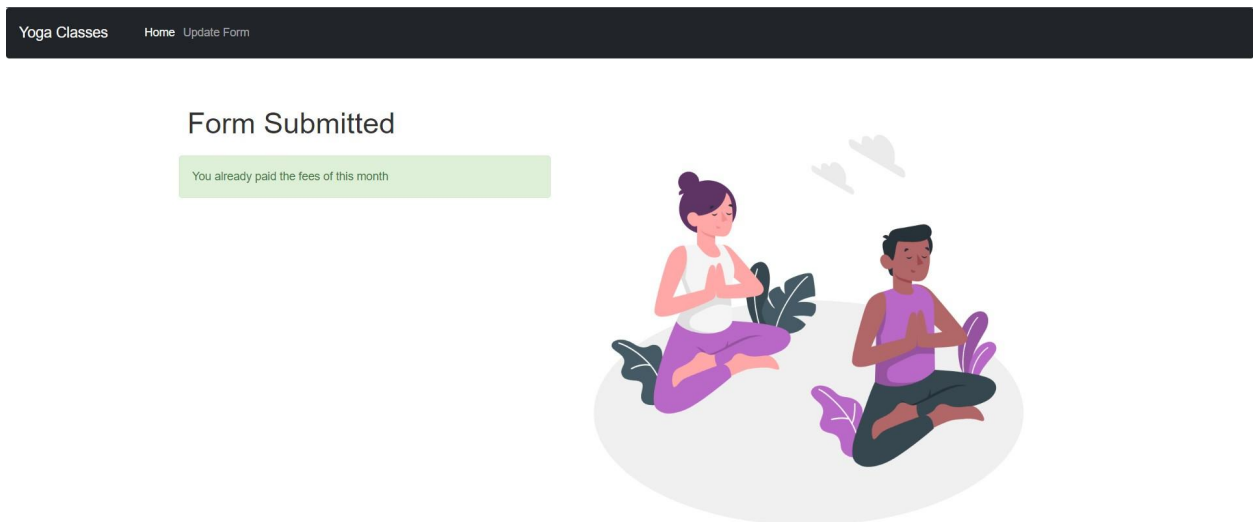
Register



Once the participants have registered successfully for the yoga classes, then gets this webpage :-



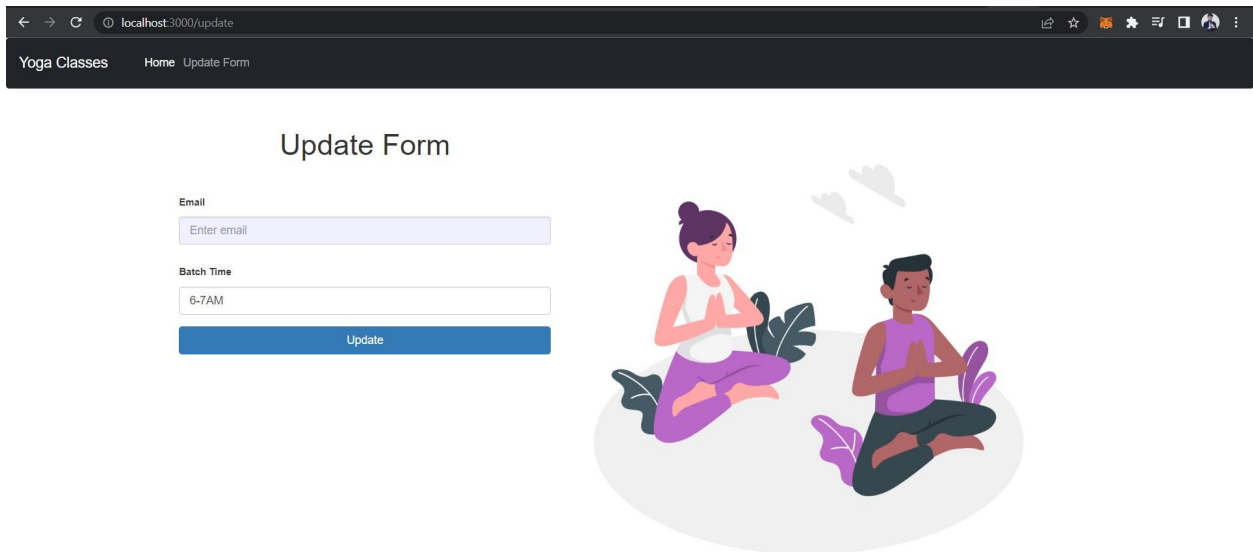
If he tries to pay the fees for the same month multiple times than also we will receive that he has paid the fees for this month



## Batch Updation Form (Batch Update Form) :-

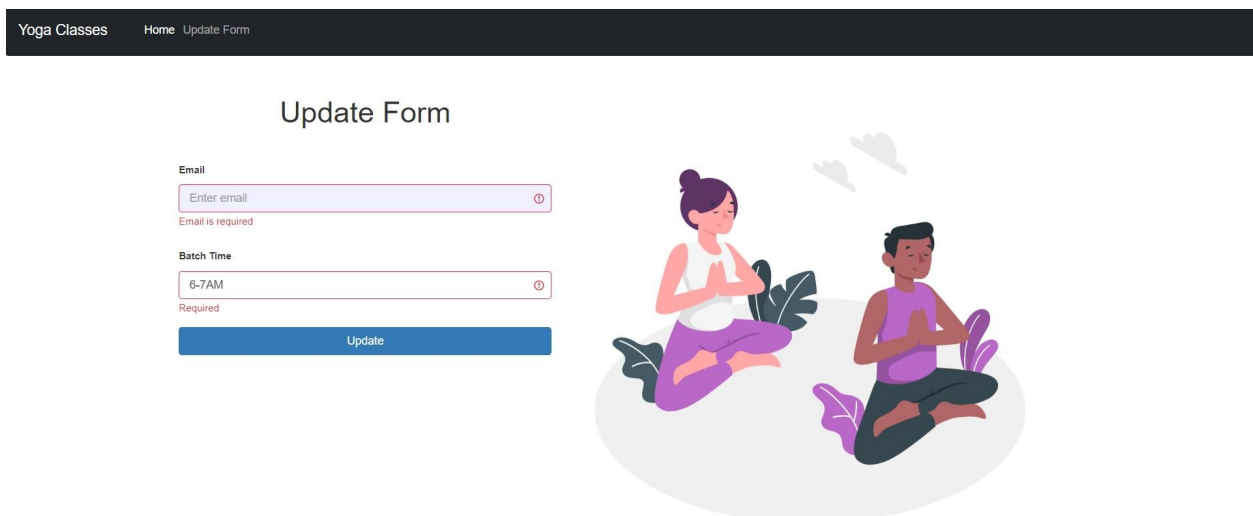
So this is the form that is for the purpose of giving the option to the participants to change the batch but the participant can change for only one time at any time during the month.

## Update Form :-

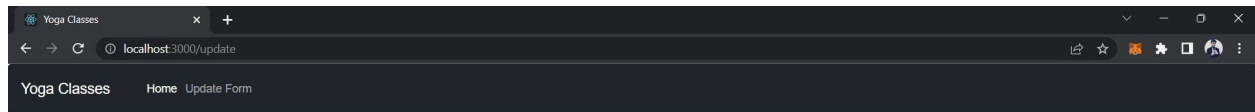


The screenshot shows a web browser window with the address bar displaying 'localhost:3000/update'. The browser's navigation bar includes links for 'Yoga Classes', 'Home', and 'Update Form'. The main content area is titled 'Update Form' and features a form with two input fields: 'Email' with a placeholder 'Enter email' and 'Batch Time' with a placeholder '6-7AM'. Below these fields is a blue 'Update' button. To the right of the form is an illustration of a man and a woman in yoga poses, surrounded by stylized plants and clouds.

## Update Form Validation :-



The screenshot shows the same 'Update Form' as before, but with validation errors. The 'Email' input field now has a red border and a red error message 'Email is required' below it. The 'Batch Time' input field also has a red border and a red error message 'Required' below it. The 'Update' button remains blue. The illustration of the yoga practitioners is still present on the right side of the form.



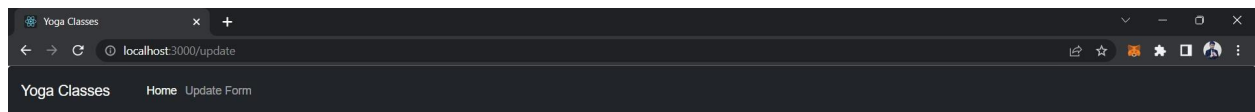
## Update Form

Email

Batch Time

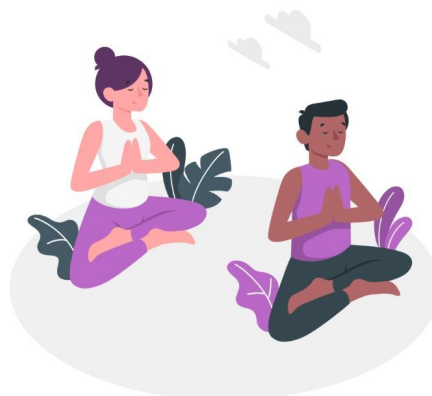


If the participant has the chance to change the batch for this month and he changes the batch then we will receive a message on web page that the batch time is updated successfully :-

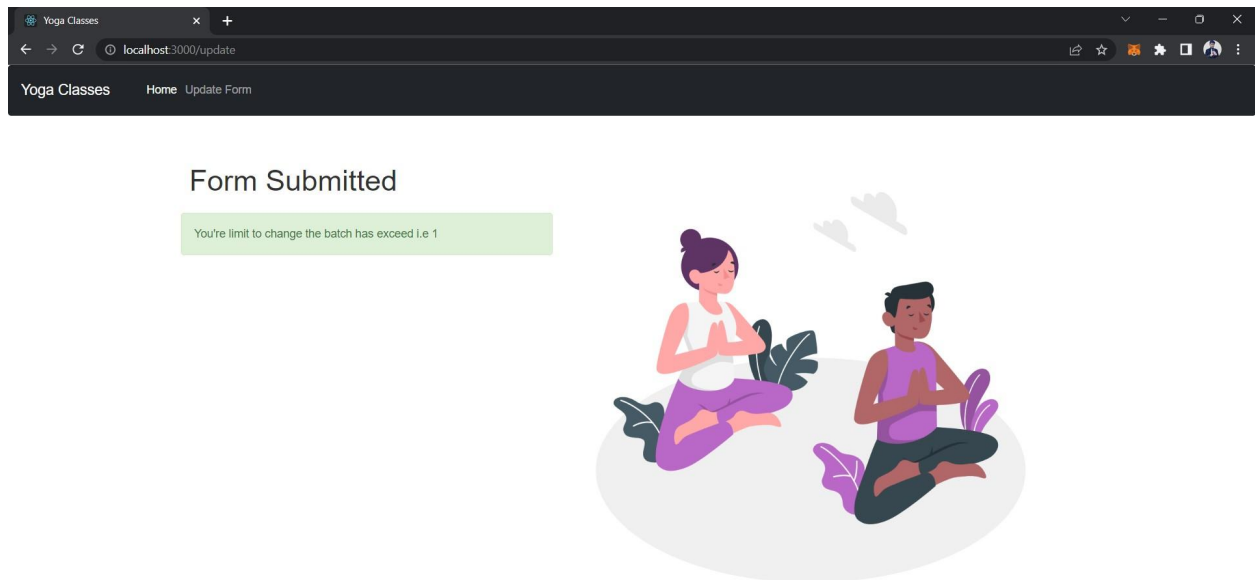


## Form Submitted

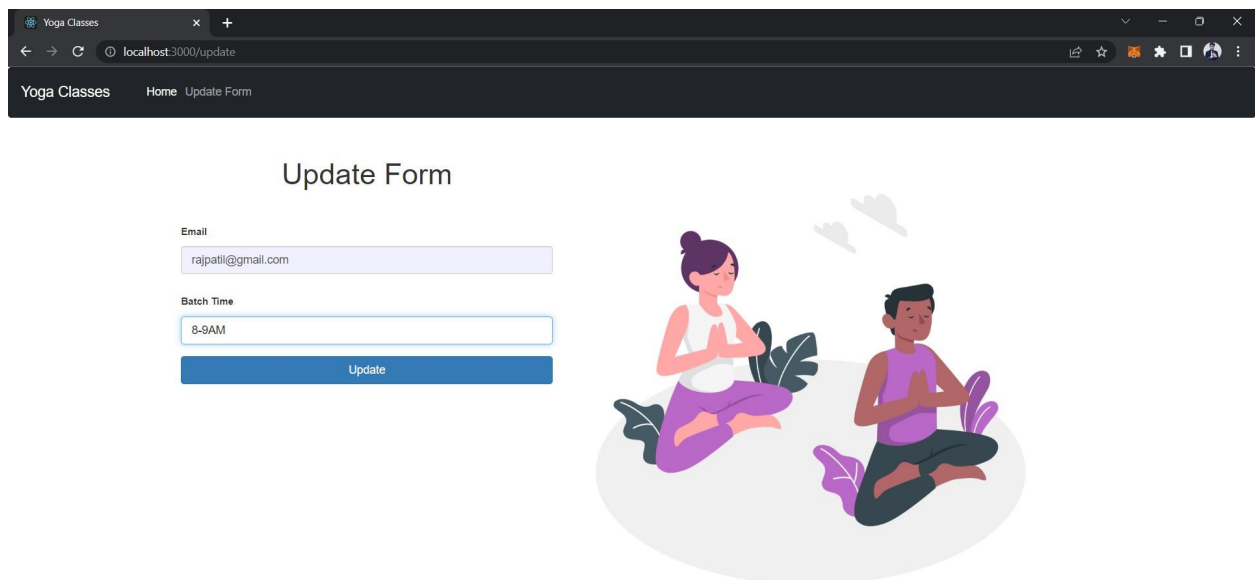
Batch Time Updated successfully!!



If the participant tried to change the batch even he has used the chance to change the batch for this month then he will receive a message on web page that the limit to change the batch has exceed from his end :-



If the unregistered participant tries to change the batch, then it will display the message that please register for the yoga classes first.



## Form Submitted

Please register for Yoga Classes First!!

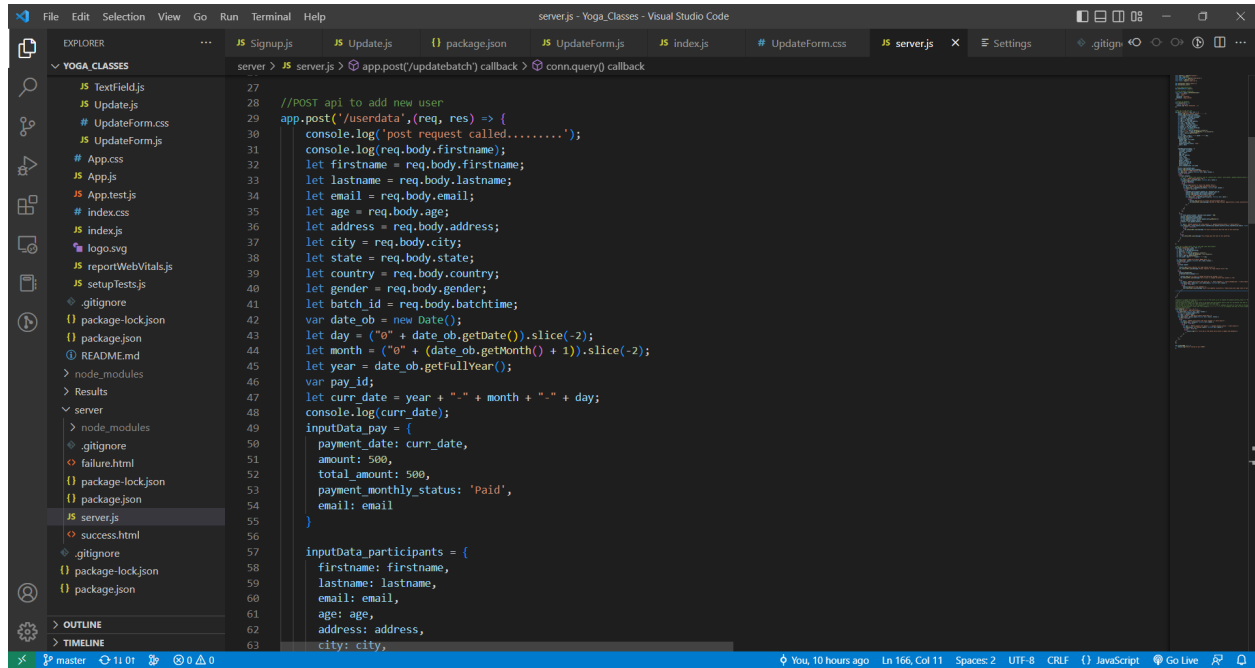




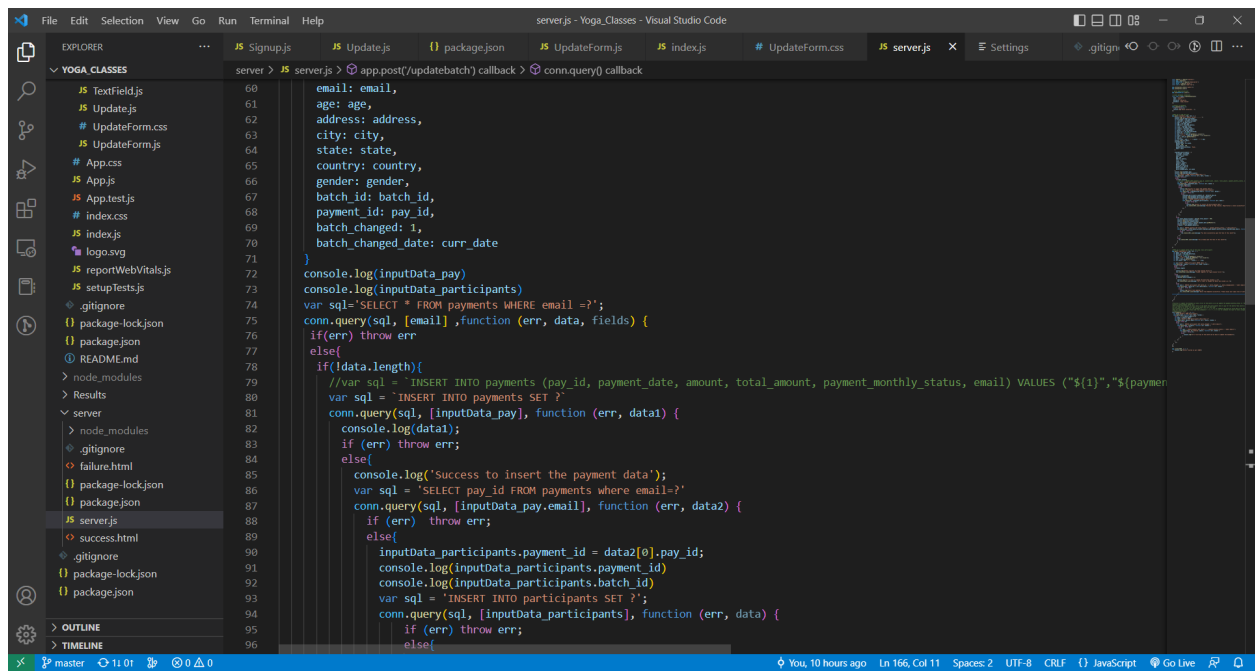
## Backend :-

As I have mentioned in the tech stack I have used Nodejs and Expressjs in the backend. The main code of the backend is present in the server/server.js of the project.

### POST API written to new user :-



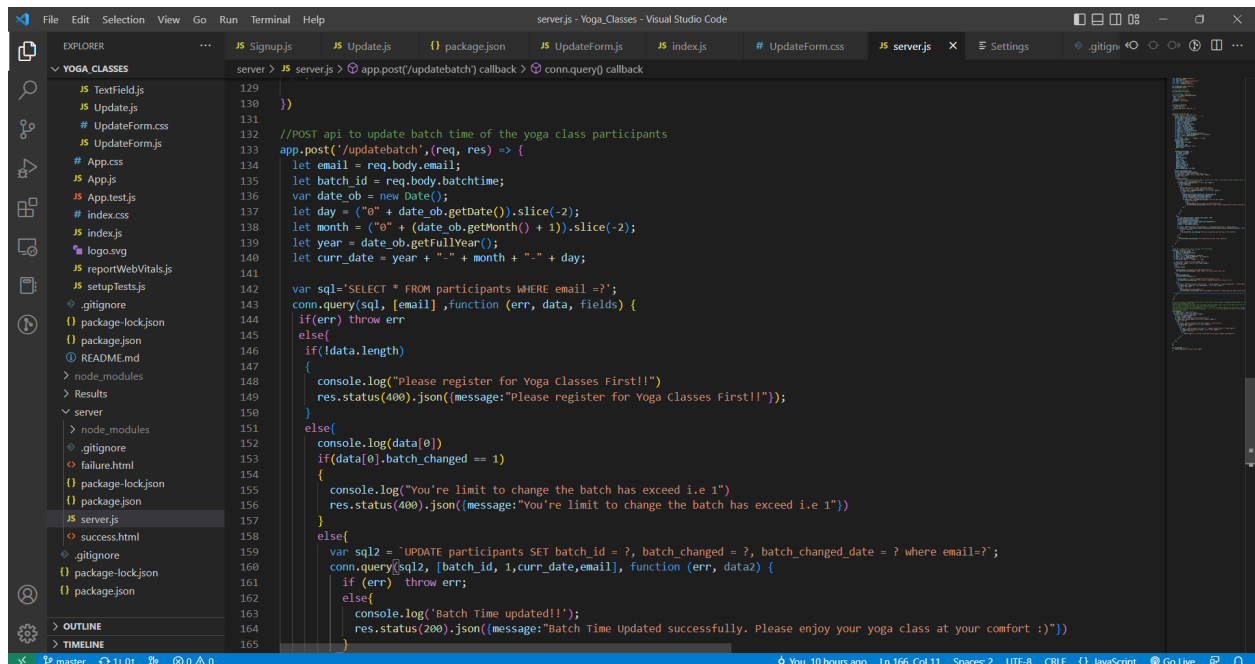
```
server > JS server.js > app.post(updatebatch) callback > conn.query() callback
27
28 //POST api to add new user
29 app.post('/userdata',(req, res) => {
30   console.log('post request called.....');
31   console.log(req.body.firstname);
32   let firstname = req.body.firstname;
33   let lastname = req.body.lastname;
34   let email = req.body.email;
35   let age = req.body.age;
36   let address = req.body.address;
37   let city = req.body.city;
38   let state = req.body.state;
39   let country = req.body.country;
40   let gender = req.body.gender;
41   let batch_id = req.body.batchtime;
42   var date_ob = new Date();
43   let day = ('0' + date_ob.getDate()).slice(-2);
44   let month = ('0' + (date_ob.getMonth() + 1)).slice(-2);
45   let year = date_ob.getFullYear();
46   var pay_id;
47   let curr_date = year + "-" + month + "-" + day;
48   console.log(curr_date);
49   inputData_pay = {
50     payment_date: curr_date,
51     amount: 500,
52     total_amount: 500,
53     payment_monthly_status: 'Paid',
54     email: email
55   }
56
57   inputData_participants = {
58     firstname: firstname,
59     lastname: lastname,
60     email: email,
61     age: age,
62     address: address,
63     city: city,
```



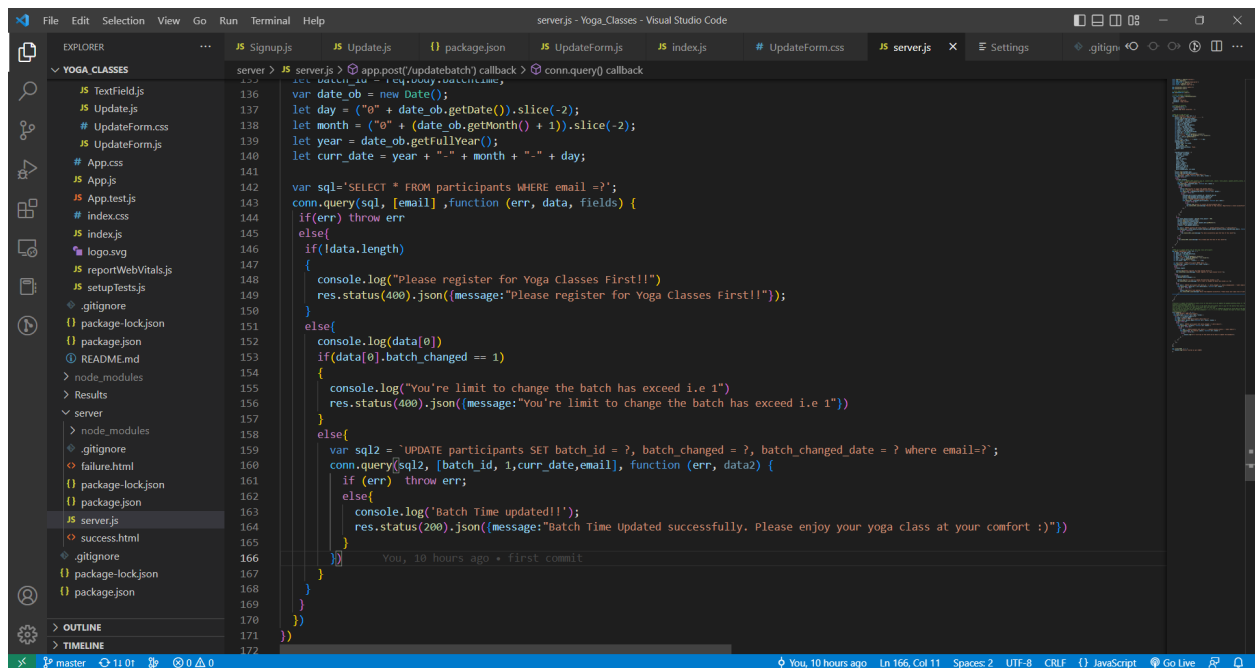
```
server > JS server.js > app.post(updatebatch) callback > conn.query() callback
60   email: email,
61   age: age,
62   address: address,
63   city: city,
64   state: state,
65   country: country,
66   gender: gender,
67   batch_id: batch_id,
68   payment_id: pay_id,
69   batch_changed: 1,
70   batch_changed_date: curr_date
71 }
72 console.log(inputData_pay)
73 console.log(inputData_participants)
74 var sql='SELECT * FROM payments WHERE email=?';
75 conn.query(sql, [email], function (err, data, fields) {
76   if(err) throw err
77   else{
78     if(data.length){
79       //var sql = `INSERT INTO payments (pay_id, payment_date, amount, total_amount, payment_monthly_status, email) VALUES ("${1}","${payment_date},${amount},${total_amount},${payment_monthly_status},${email})`;
80       var sql = `INSERT INTO payments SET ?`;
81       conn.query(sql, [inputData_pay], function (err, data1) {
82         console.log(data1);
83         if (err) throw err;
84         else{
85           console.log('Success to insert the payment data');
86           var sql = `SELECT pay_id FROM payments where email=?`;
87           conn.query(sql, [inputData_pay.email], function (err, data2) {
88             if (err) throw err;
89             else{
90               inputData_participants.payment_id = data2[0].pay_id;
91               console.log(inputData_participants.payment_id)
92               console.log(inputData_participants.batch_id)
93               var sql = `INSERT INTO participants SET ?`;
94               conn.query(sql, [inputData_participants], function (err, data) {
95                 if (err) throw err;
96                 else{
```



## POST API written to update the batch of the participants :-



```
server.js: Yoga_Classes - Visual Studio Code
server > JS server.js > app.post('/updatebatch') callback > conn.query() callback
129
130 })
131
132 //POST api to update batch time of the yoga class participants
133 app.post('/updatebatch',(req, res) => {
134   let email = req.body.email;
135   let batch_id = req.body.batchtime;
136   var date_ob = new Date();
137   let day = ("0" + date_ob.getDate()).slice(-2);
138   let month = ("0" + (date_ob.getMonth() + 1)).slice(-2);
139   let year = date_ob.getFullYear();
140   let curr_date = year + "-" + month + "-" + day;
141
142   var sql='SELECT * FROM participants WHERE email=?';
143   conn.query(sql, [email],function (err, data, fields) {
144     if(err) throw err
145     else{
146       if(!data.length)
147       {
148         console.log("Please register for Yoga Classes First!!")
149         res.status(400).json({message:"Please register for Yoga Classes First!!"});
150       }
151       else{
152         console.log(data[0])
153         if(data[0].batch_changed == 1)
154         {
155           console.log("You're limit to change the batch has exceed i.e 1")
156           res.status(400).json({message:"You're limit to change the batch has exceed i.e 1"})
157         }
158         else{
159           var sql2 = `UPDATE participants SET batch_id = ?, batch_changed = ?, batch_changed_date = ? where email=?`;
160           conn.query(sql2, [batch_id, 1, curr_date, email], function (err, data2) {
161             if (err) throw err;
162             else{
163               console.log('Batch Time updated!!!');
164               res.status(200).json({message:"Batch Time Updated successfully. Please enjoy your yoga class at your comfort :)"}))
165             }
166           }
167         }
168       }
169     }
170   })
171 }
172 }
```



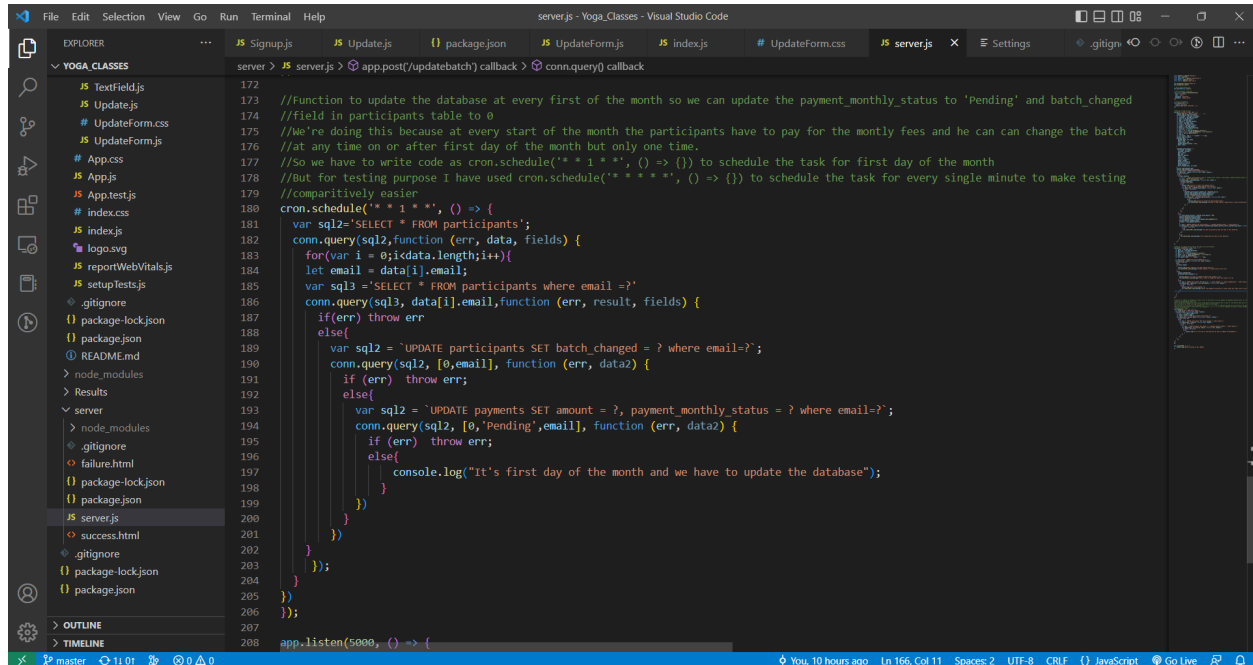
```
server.js: Yoga_Classes - Visual Studio Code
server > JS server.js > app.post('/updatebatch') callback > conn.query() callback
136   var date_ob = new Date();
137   let day = ("0" + date_ob.getDate()).slice(-2);
138   let month = ("0" + (date_ob.getMonth() + 1)).slice(-2);
139   let year = date_ob.getFullYear();
140   let curr_date = year + "-" + month + "-" + day;
141
142   var sql='SELECT * FROM participants WHERE email=?';
143   conn.query(sql, [email],function (err, data, fields) {
144     if(err) throw err
145     else{
146       if(!data.length)
147       {
148         console.log("Please register for Yoga Classes First!!")
149         res.status(400).json({message:"Please register for Yoga Classes First!!"});
150       }
151       else{
152         console.log(data[0])
153         if(data[0].batch_changed == 1)
154         {
155           console.log("You're limit to change the batch has exceed i.e 1")
156           res.status(400).json({message:"You're limit to change the batch has exceed i.e 1"})
157         }
158         else{
159           var sql2 = `UPDATE participants SET batch_id = ?, batch_changed = ?, batch_changed_date = ? where email=?`;
160           conn.query(sql2, [batch_id, 1, curr_date, email], function (err, data2) {
161             if (err) throw err;
162             else{
163               console.log('Batch Time updated!!!');
164               res.status(200).json({message:"Batch Time Updated successfully. Please enjoy your yoga class at your comfort :)"}))
165             }
166           }
167         }
168       }
169     }
170   })
171 }
172 }
```

You, 10 hours ago + first commit

This API will check whether the user that has requested to change the bath is already registered or not. If he is not registered then it will send the message in response that Please register for the yoga class first. If the participants have the chance to change the batch for this particular month it will change the batch time successfully when requested. **If**

the participants have used the chance to change the batch time then it will give a response that the limit to change the batch time has exceeded for this particular month.

### Cron job function :-



```
172 //Function to update the database at every first of the month so we can update the payment_monthly_status to 'Pending' and batch_changed
173 //field in participants table to 0
174 //We're doing this because at every start of the month the participants have to pay for the monthly fees and he can change the batch
175 //at any time on or after first day of the month but only one time.
176 //So we have to write code as cron.schedule('* * * * *', () => {}) to schedule the task for first day of the month
177 //But for testing purpose I have used cron.schedule('* * * * *', () => {}) to schedule the task for every single minute to make testing
178 //comparitively easier
179 cron.schedule('* * * * *', () => {
180   var sql2='SELECT * FROM participants';
181   conn.query(sql2,function (err, data, fields) {
182     for(var i = 0;i<data.length;i++){
183       let email = data[i].email;
184       var sql3='SELECT * FROM participants where email=?'
185       conn.query(sql3, data[i].email,function (err, result, fields) {
186         if(err) throw err
187         else{
188           var sql2 = `UPDATE participants SET batch_changed = ? where email=?`;
189           conn.query(sql2, [0,email], function (err, data2) {
190             if (err) throw err;
191             else{
192               var sql2 = `UPDATE payments SET amount = ?, payment_monthly_status = ? where email=?`;
193               conn.query(sql2, [0,'Pending',email], function (err, data2) {
194                 if (err) throw err;
195                 else{
196                   console.log("It's first day of the month and we have to update the database");
197                 }
198               })
199             }
200           })
201         }
202       })
203     }
204   });
205 })
206 });
207
208 app.listen(5000, () => {
```

I have used the node-cron package in the nodejs to schedule a task. So task is that at every first day of the new month it will update the batch\_changed to 0 means the user will get the option to change the batch for the new month and payment\_monthly\_status field to 'Pending' as mentioned in the assumptions that at for every new month the participants has to pay fees of full month.