# Lab 3

## Introduction:

The lab concentrates on the use of graphql instead of Rest Api calls in service.

Grubhub application is developed with graphql and express in backend server and React along with apollo client as frontend client.

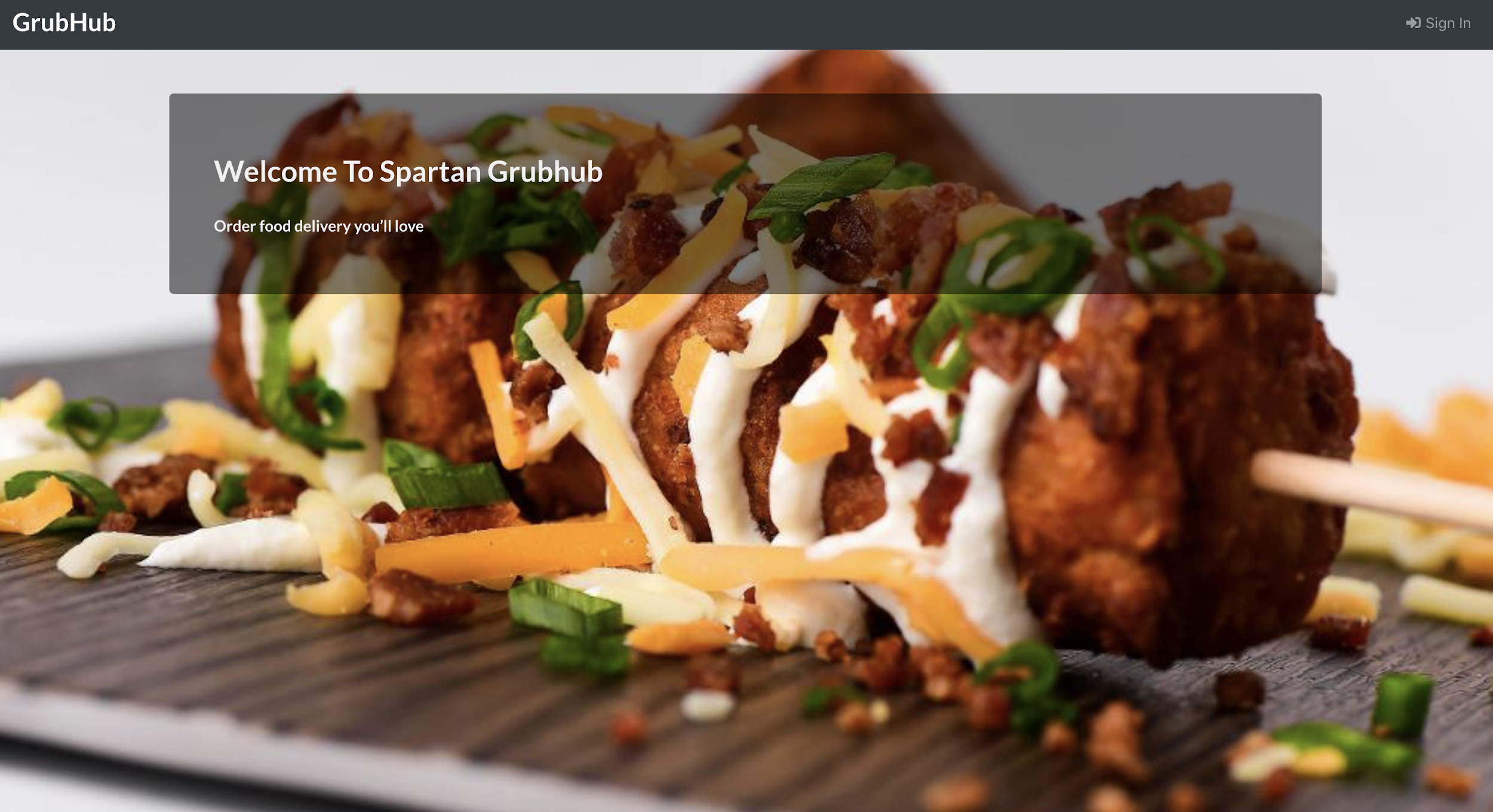
## System Design:



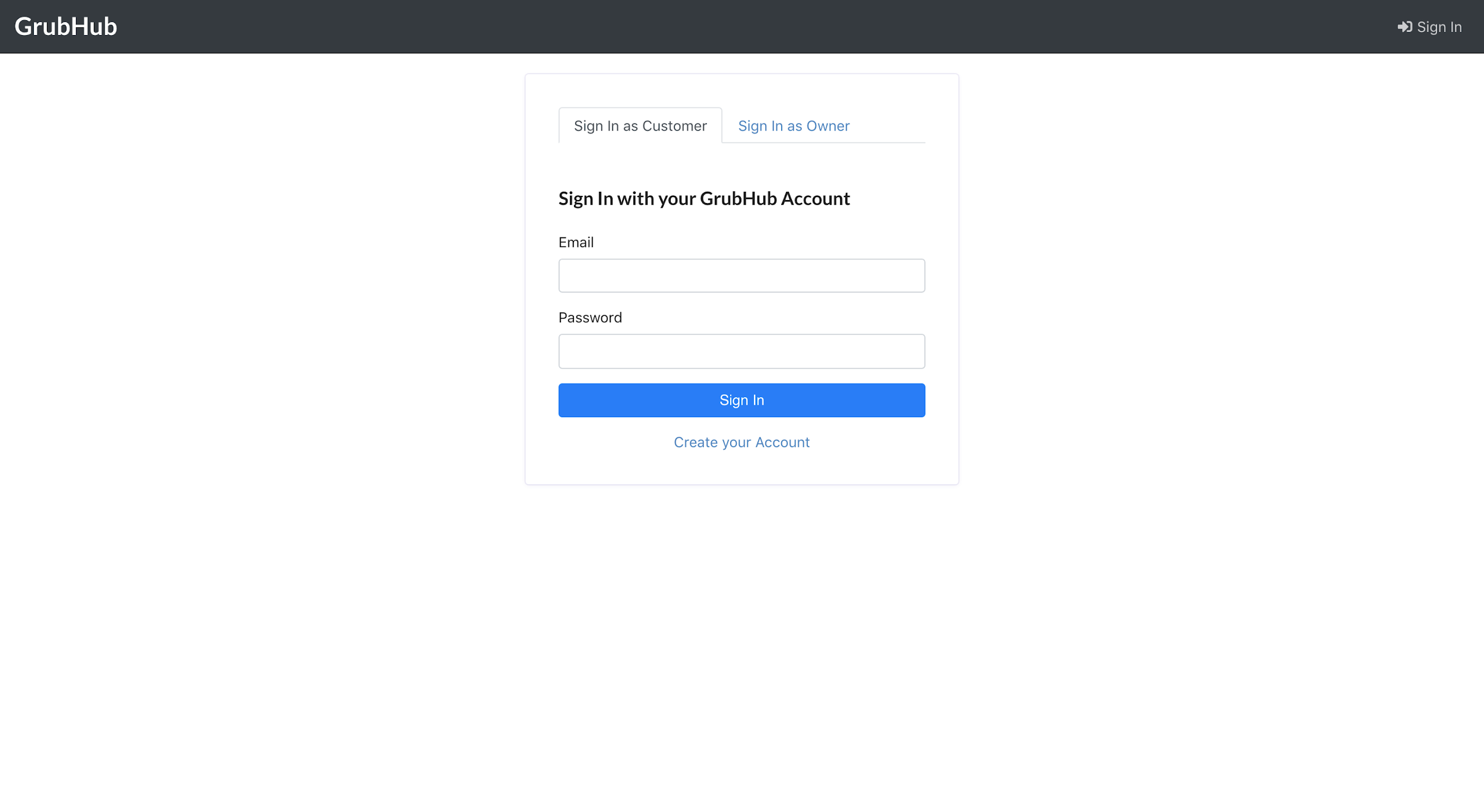
Frontend client is built using react Js along with react-apollo client and backend service with node js, express Js, mongoose, and graphql. MongoDb is used for data storage.

## Results:

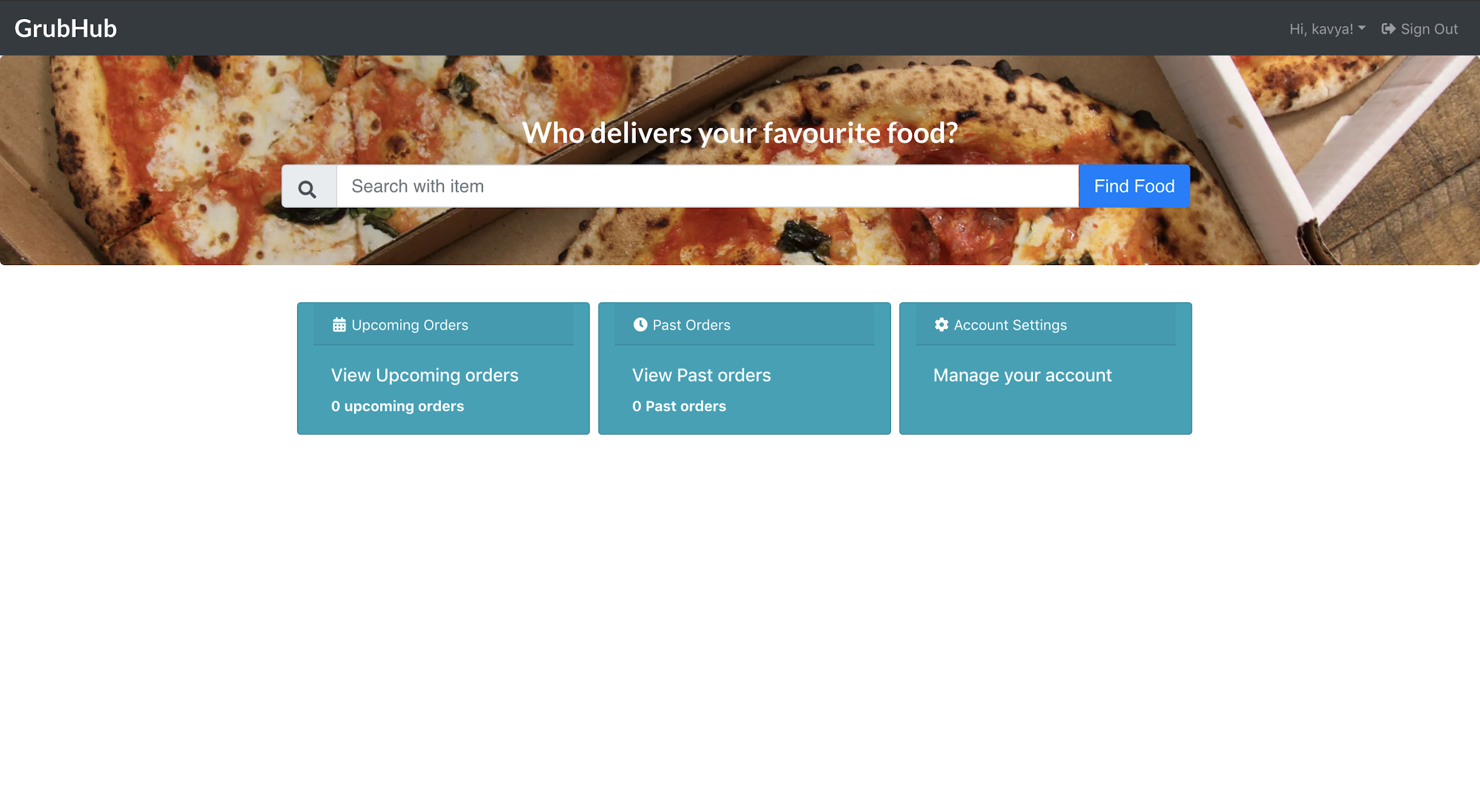
Start Page:



User Login Page:



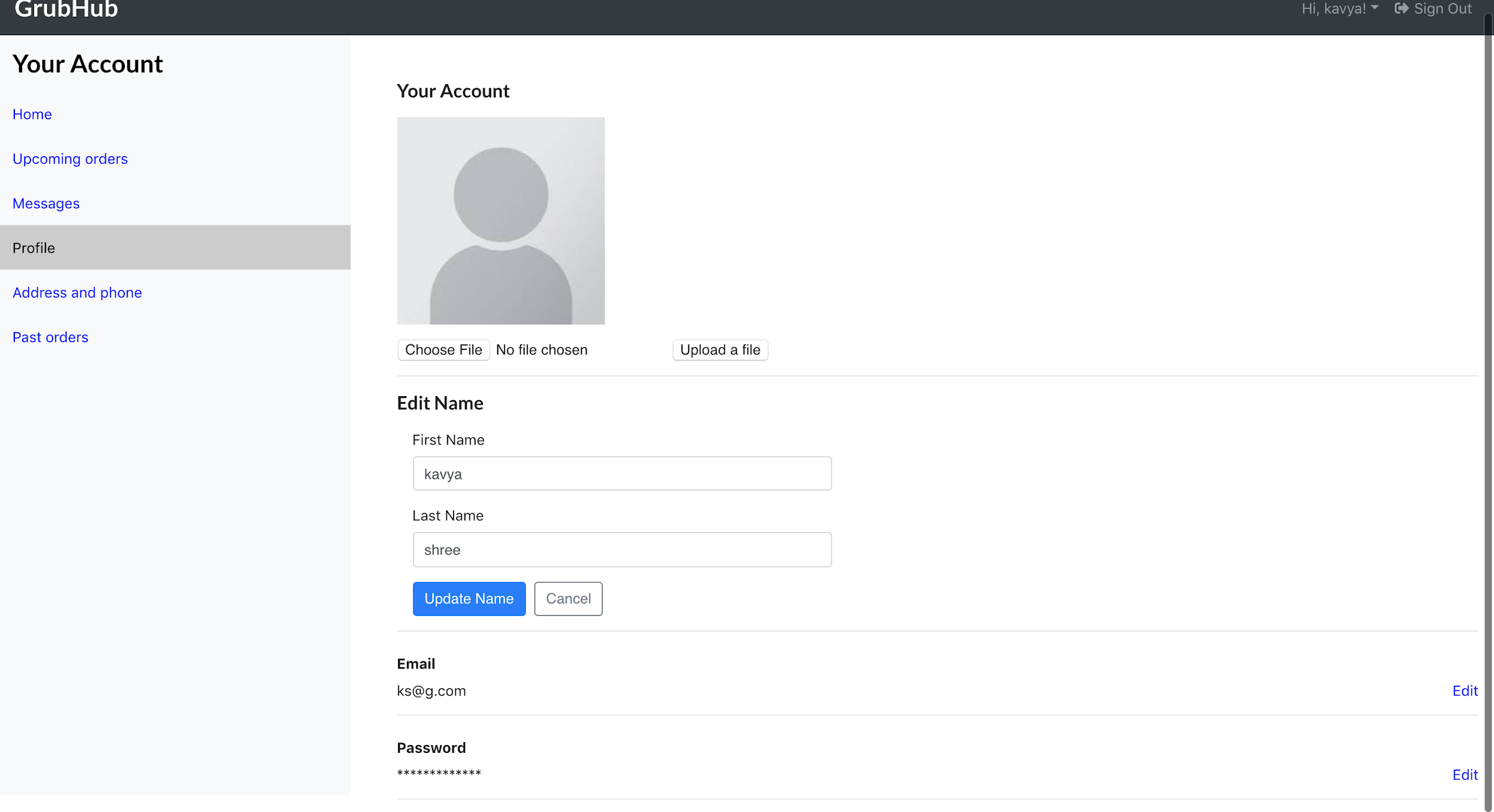
UserHome Page:



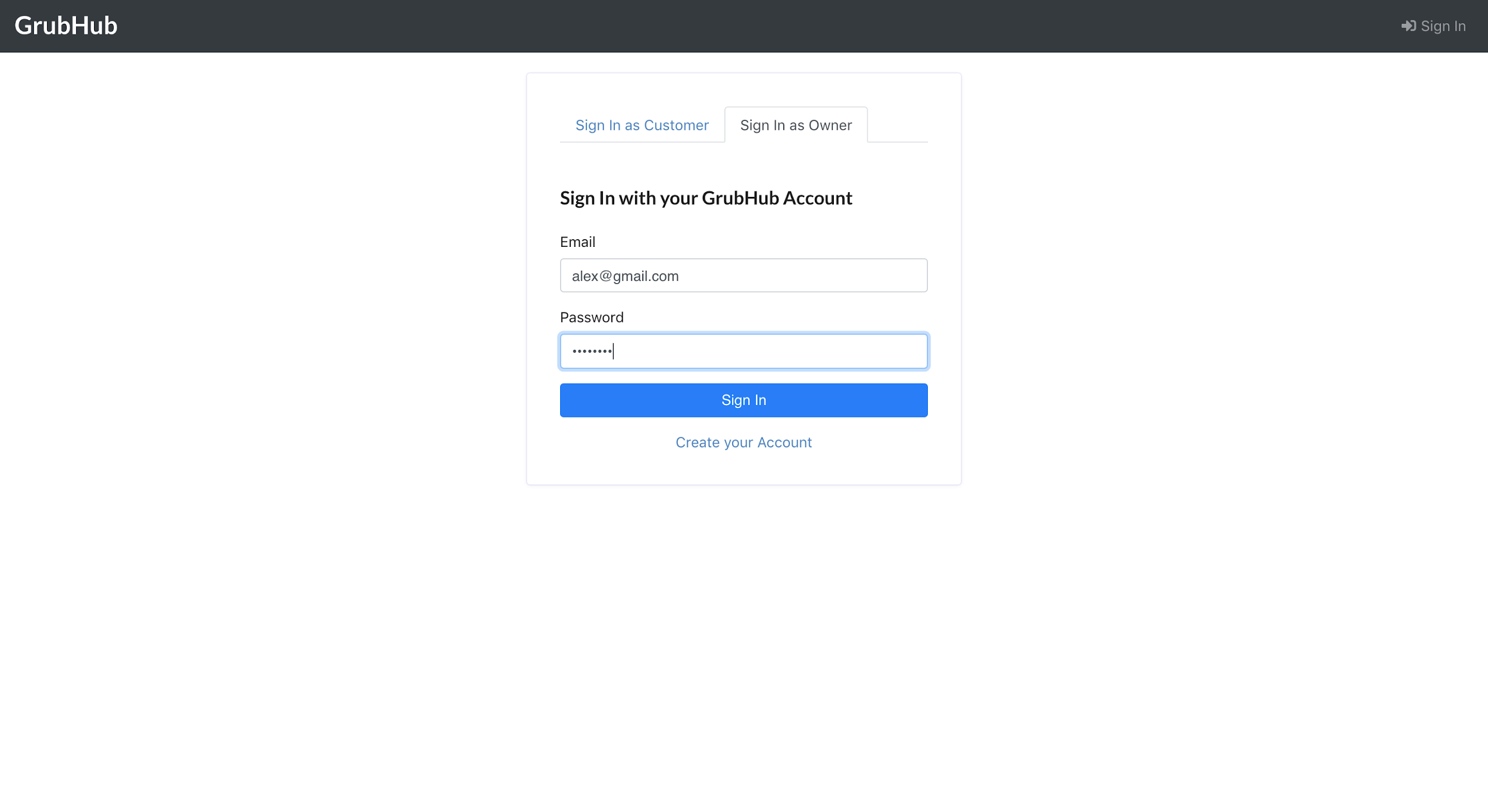
User Profile Page:



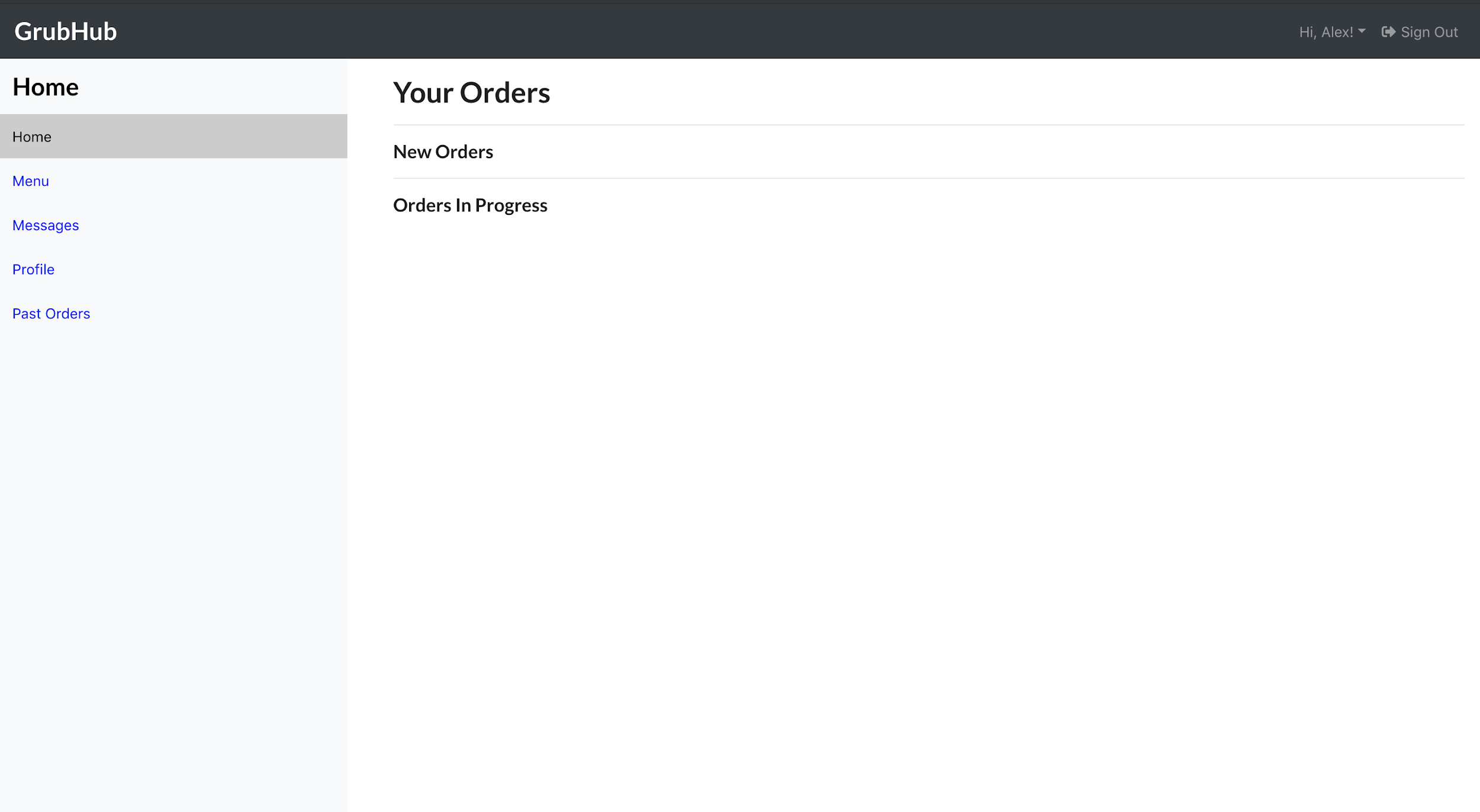
User Profile Update:



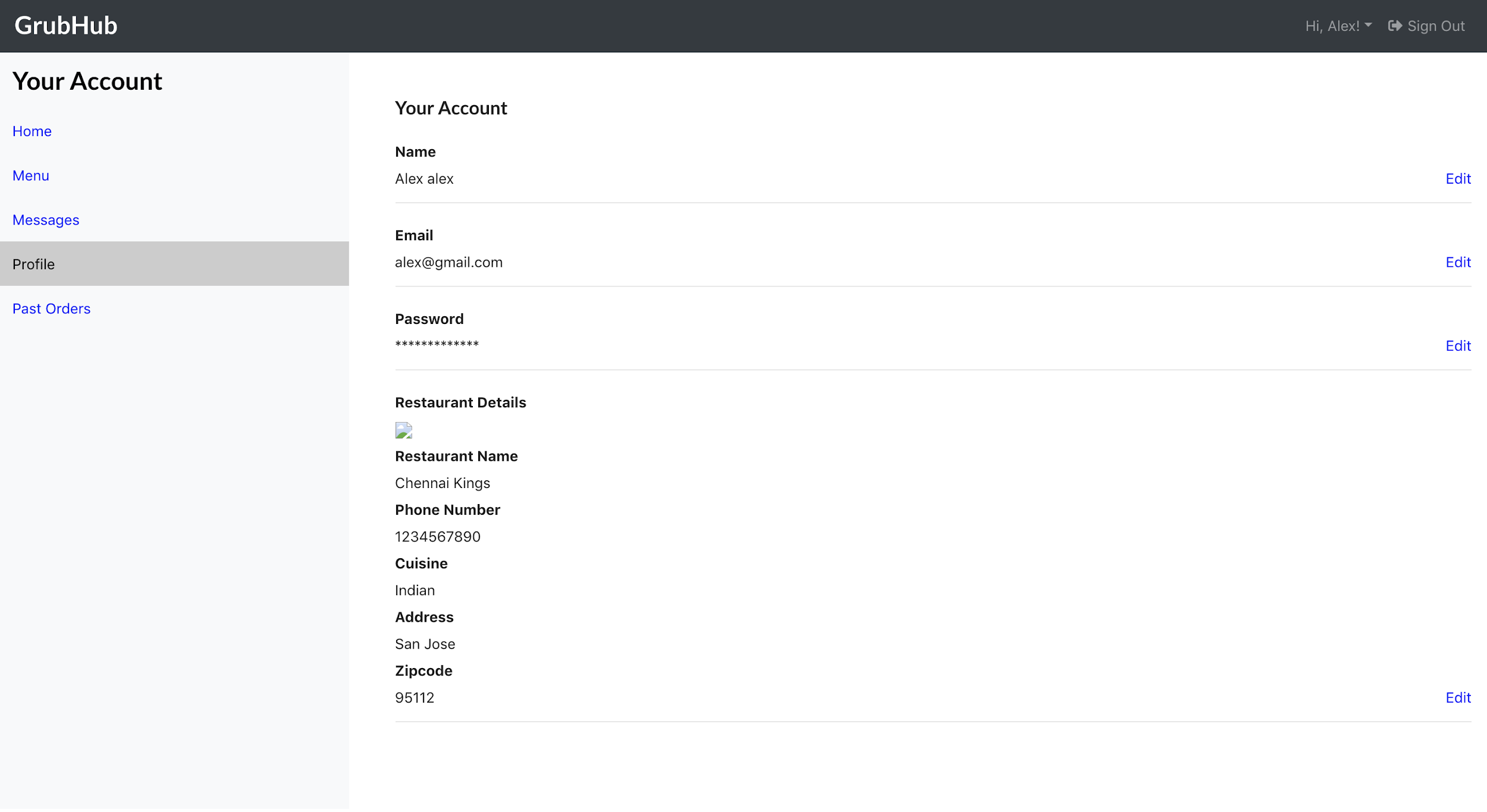
Owner Login:



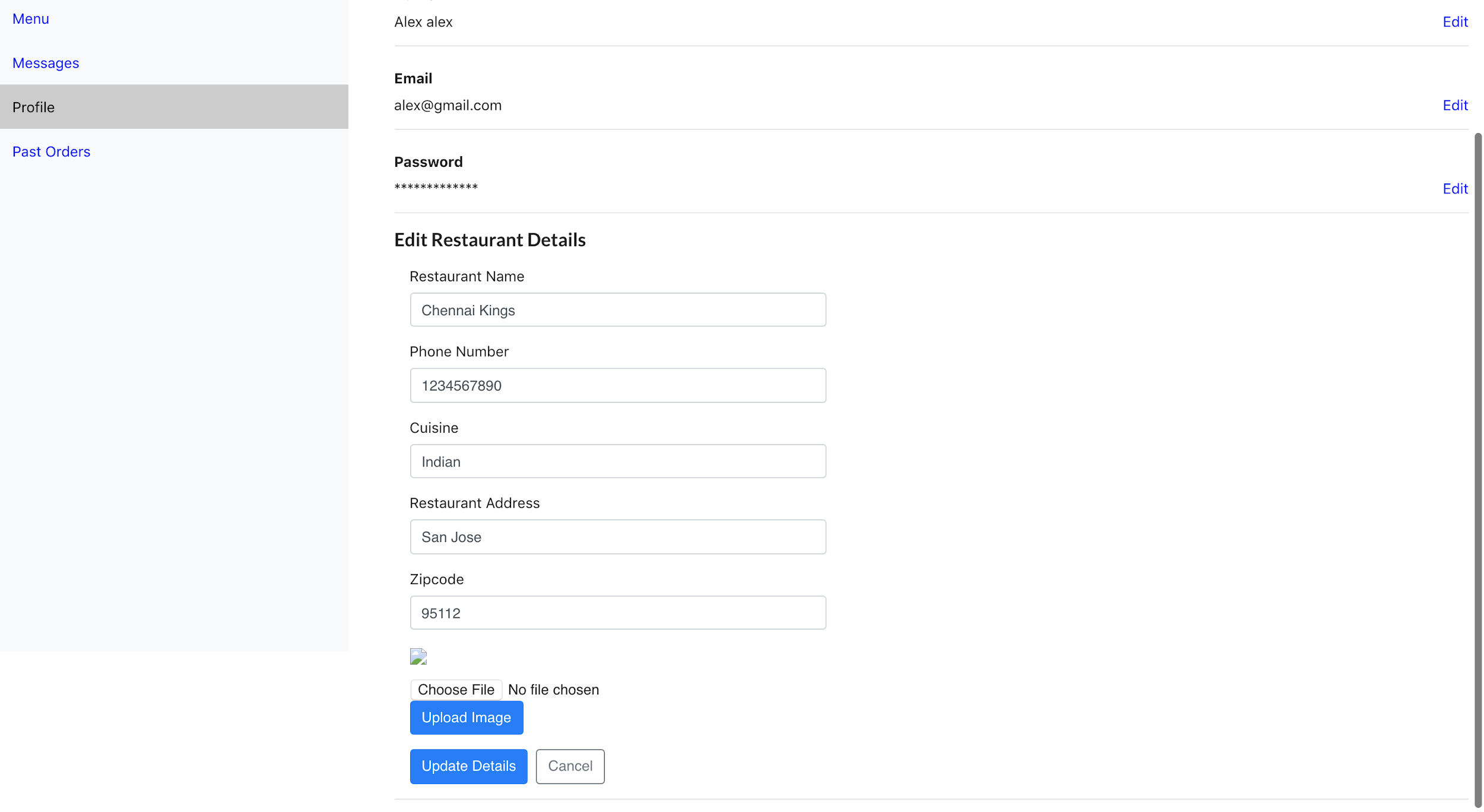
Owner Home Page:

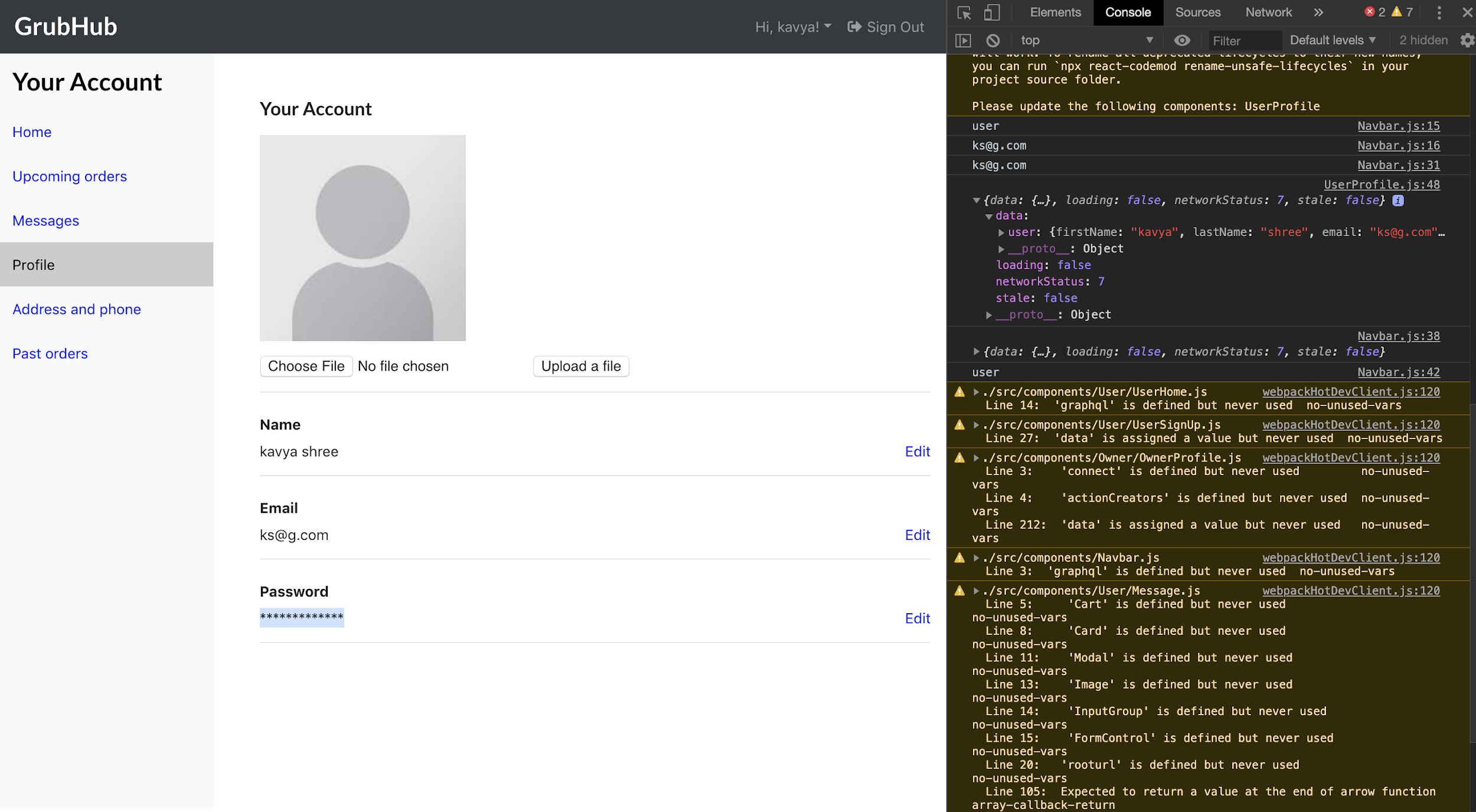


Owner Profile Page:



Owner Profile Update:





## Code Listing:

backend/Schema.js

const graphql = require('graphql');

const \_ = require('lodash');

var { user } = require("../models/UserSchema")

var { owner } = require("../models/OwnerSchema")

var { restaurant } = require("../models/RestaurantSchema")

const bcrypt = require("bcryptjs");

const saltRounds = 10;

const {

GraphQLObjectType,

GraphQLString,

GraphQLSchema,

GraphQLID,

GraphQLInt,

GraphQLList,

GraphQLNonNull

} = graphql;

const UserType = new GraphQLObjectType({

name: 'user',

fields: () => ({

firstName: { type: GraphQLString },

lastName: { type: GraphQLString },

email: { type: GraphQLString },

password: { type: GraphQLString },

address: { type: GraphQLString },

phoneNo: { type: GraphQLString },

profilePic: { type: GraphQLString }

})

})

const RestaurantType = new GraphQLObjectType({

name: 'restaurant',

fields: () => ({

name: { type: GraphQLString },

address: { type: GraphQLString },

cuisine: { type: GraphQLString },

phoneNo: { type: GraphQLString },

zipcode: { type: GraphQLString },

image: { type: GraphQLString }

})

})

const OwnerType = new GraphQLObjectType({

name: 'owner',

fields: () => ({

firstName: { type: GraphQLString },

lastName: { type: GraphQLString },

email: { type: GraphQLString },

password: { type: GraphQLString },

restaurant: { type: RestaurantType }

})

})

const RootQuery = new GraphQLObjectType({

name: 'RootQueryType',

fields: {

user: {

type: UserType,

args: { email: { type: GraphQLString } },

resolve(parent, args) {

let gbUser = user.findOne({ email: args.email })

console.log(gbUser)

return gbUser

}

},

users: {

type: new GraphQLList(UserType),

resolve(parent, args) {

let gbUsers = user.find({});

console.log(gbUsers)

return gbUsers

}

},

owner: {

type: OwnerType,

args: { email: { type: GraphQLString } },

resolve(parent, args) {

let gbOwner = owner.findOne({ email: args.email })

console.log(gbOwner)

return gbOwner

}

},

}

});

const Mutation = new GraphQLObjectType({

name: 'Mutation',

fields: {

addUser: {

type: UserType,

args: {

firstName: { type: GraphQLString },

lastName: { type: GraphQLString },

email: { type: GraphQLString },

password: { type: GraphQLString }

},

resolve(parent, args) {

let hashPassword = bcrypt.hashSync(args.password, saltRounds);

console.log(hashPassword.length);

var newUser = new user({

firstName: args.firstName,

lastName: args.lastName,

email: args.email,

password: hashPassword,

phoneNo: "",

address: "",

profilePic: ""

});

var result = newUser.save(function (err, user) {

if (err) return err;

});

return result;

}

},

addOwner: {

type: OwnerType,

args: {

firstName: { type: GraphQLString },

lastName: { type: GraphQLString },

email: { type: GraphQLString },

password: { type: GraphQLString },

restaurantName: { type: GraphQLString },

zipcode: { type: GraphQLString }

},

resolve(parent, args) {

let hashPassword = bcrypt.hashSync(args.password, saltRounds);

console.log(hashPassword.length);

var newRestaurant = new restaurant({

name: args.restaurantName,

address: "",

phoneNo: "",

cuisine: "",

zipcode: args.zipcode,

image: ""

});

var newOwner = new owner({

firstName: args.firstName,

lastName: args.lastName,

email: args.email,

password: hashPassword,

restaurant: newRestaurant

});

var result = newOwner.save(function (err, user) {

if (err) return err;

});

return result;

}

},

updateUserName: {

type: UserType,

args: {

firstName: { type: GraphQLString },

lastName: { type: GraphQLString },

email: { type: GraphQLString }

},

async resolve(parent, args) {

let useru = await user.findOneAndUpdate({ email: args.email }, { firstName: args.firstName, lastName: args.lastName }, { new: true })

return useru;

}

},

updateUserEmail: {

type: UserType,

args: {

newEmail: { type: GraphQLString },

email: { type: GraphQLString }

},

resolve(parent, args) {

return user.findOneAndUpdate({ email: args.email }, { email: args.newEmail }, { new: true })

}

},

updateUserPassword: {

type: UserType,

args: {

password: { type: GraphQLString },

email: { type: GraphQLString }

},

resolve(parent, args) {

let hashPassword = bcrypt.hashSync(args.password, saltRounds);

user.findOneAndUpdate({ email: args.email }, { password: hashPassword }, (err, rows) => {

if (err) return err;

else return rows;

})

}

},

updateOwnerName: {

type: OwnerType,

args: {

firstName: { type: GraphQLString },

lastName: { type: GraphQLString },

email: { type: GraphQLString }

},

async resolve(parent, args) {

let useru = await owner.findOneAndUpdate({ email: args.email }, { firstName: args.firstName, lastName: args.lastName }, { new: true })

return useru;

}

},

updateOwnerEmail: {

type: OwnerType,

args: {

newEmail: { type: GraphQLString },

email: { type: GraphQLString }

},

resolve(parent, args) {

return owner.findOneAndUpdate({ email: args.email }, { email: args.newEmail }, { new: true })

}

},

updateOwnerPassword: {

type: OwnerType,

args: {

password: { type: GraphQLString },

email: { type: GraphQLString }

},

resolve(parent, args) {

let hashPassword = bcrypt.hashSync(args.password, saltRounds);

return owner.findOneAndUpdate({ email: args.email }, { password: hashPassword }, { new: true })

}

},

updateOwnerRestaurant: {

type: OwnerType,

args: {

email: { type: GraphQLString },

name: { type: GraphQLString },

cuisine: { type: GraphQLString },

address: { type: GraphQLString },

phoneNo: { type: GraphQLString },

zipcode: { type: GraphQLString }

},

resolve(parent, args) {

var newRestaurant = new restaurant({

name: args.name,

address: args.address,

phoneNo: args.phoneNo,

cuisine: args.cuisine,

zipcode: args.zipcode,

image: ""

});

return owner.findOneAndUpdate({ email: args.email }, { restaurant: newRestaurant }, { new: true })

}

}

}

});

module.exports = new GraphQLSchema({

query: RootQuery,

mutation: Mutation

});

Backend/index.js

const express = require('express');

const graphqlHTTP = require('express-graphql');

const schema = require('./schema/schema');

const mongoose = require('mongoose')

const cors = require('cors');

var bodyParser = require("body-parser");

const app = express();

app.use(cors({ origin: 'http://localhost:3000', credentials: true }));

app.use(function (req, res, next) {

res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');

res.setHeader('Access-Control-Allow-Credentials', 'true');

res.setHeader('Access-Control-Allow-Methods', 'GET,HEAD,OPTIONS,POST,PUT,DELETE');

res.setHeader('Access-Control-Allow-Headers', 'Access-Control-Allow-Headers, Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers');

res.setHeader('Cache-Control', 'no-cache');

next();

});

app.use(bodyParser.json());

mongoose.connect('mongodb+srv://root:root@clusterkc-cr6mm.mongodb.net/grubhub?retryWrites=true&w=majority', { useNewUrlParser: true, poolSize: 10 }, function (err) {

if (err) throw err;

else {

console.log('Successfully connected to MongoDB');

}

});

app.use("/graphql", graphqlHTTP({

schema,

graphiql: true

}));

const login = require("./routes/login")

app.use("/", login)

app.listen(8080, () => {

console.log("GraphQL server started on port 8080");

})

Frontend/Quesries.js

import { gql } from 'apollo-boost';

const getUsersQuery = gql`

{

users {

firstName

lastName

email

password

}

}

`;

const getUserQuery = gql`

query user($email: String) {

user(email:$email) {

firstName

lastName

email

password

}

}

`;

const getOwnerQuery = gql`

query owner($email: String) {

owner(email:$email) {

firstName

lastName

email

password

restaurant {

name

cuisine

address

phoneNo

zipcode

}

}

}

`;

export { getUsersQuery, getUserQuery, getOwnerQuery };

frontend/mutation.js

import { gql } from 'apollo-boost';

const addBookMutation = gql`

mutation AddBook($name: String, $genre: String, $authorId: ID){

addBook(name: $name, genre: $genre, authorId: $authorId){

name

id

}

}

`;

const addUserMutation = gql`

mutation AddUser($firstName: String, $lastName: String, $email: String, $password: String){

addUser(firstName: $firstName, lastName: $lastName, email: $email, password: $password){

firstName

lastName

email

}

}

`;

const addOwnerMutation = gql`

mutation AddOwner($firstName: String, $lastName: String, $email: String, $password: String, $restaurantName: String, $zipcode: String){

addOwner(firstName: $firstName, lastName: $lastName, email: $email, password: $password, restaurantName: $restaurantName, zipcode:$zipcode){

firstName

lastName

email

}

}

`;

const updateUserNameMutation = gql`

mutation UpdateUserName($firstName: String, $lastName: String, $email: String){

updateUserName(firstName: $firstName, lastName: $lastName, email: $email){

firstName

lastName

email

passsword

}

}

`;

const updateUserEmailMutation = gql`

mutation UpdateEmailName( $newEmail: String, $email: String){

updateUserEmail(newEmail: $newEmail, email: $email){

firstName

lastName

email

password

}

}

`;

const updateUserPasswordMutation = gql`

mutation UpdateUserPassword($password: String, $email: String){

updateUserPassword(password: $password, email: $email){

firstName

lastName

email

password

}

}

`;

const updateOwnerNameMutation = gql`

mutation UpdateOwnerrName($firstName: String, $lastName: String, $email: String){

updateOwnerName(firstName: $firstName, lastName: $lastName, email: $email){

firstName

lastName

email

password

restaurant {

name

cuisine

address

phoneNo

zipcode

}

}

}

`;

const updateOwnerEmailMutation = gql`

mutation UpdateOwnerEmail( $newEmail: String, $email: String){

updateOwnerEmail(newEmail: $newEmail, email: $email){

firstName

lastName

email

password

restaurant {

name

cuisine

address

phoneNo

zipcode

}

}

}

`;

const updateOwnerPasswordMutation = gql`

mutation UpdateOwnerPassword($password: String, $email: String){

updateOwnerPassword(password: $password, email: $email){

firstName

lastName

email

password

restaurant {

name

cuisine

address

phoneNo

zipcode

}

}

}

`;

const updateOwnerRestaurantMutation = gql`

mutation UpdateOwnerRestaurant($email: String, $name: String, $cuisine: String, $address: String, $phoneNo: String, $zipcode: String){

updateOwnerRestaurant(email: $email, name: $name, cuisine: $cuisine, address: $address, phoneNo: $phoneNo, zipcode: $zipcode ){

firstName

lastName

email

password

restaurant {

name

cuisine

address

phoneNo

zipcode

}

}

}

`;

export {

addBookMutation, addUserMutation, addOwnerMutation, updateUserNameMutation,

updateUserEmailMutation, updateUserPasswordMutation, updateOwnerEmailMutation,

updateOwnerNameMutation, updateOwnerPasswordMutation, updateOwnerRestaurantMutation

};

## Questions:

1. We can upload images using two ways,
   1. Uploading using separate calls, first call to upload file, then to save the path using mutation
   2. Encoding in base64 format is the another way, but this will not be as effective as the whole image size needs to be stored in database.
2. Graphql-go-upload is one of the open source library, which is used as middleware to upload multipart data using graphql. This is simple and easy to access and also can be implemented using node Js apollo client server