A

Project Report On

Online Thrift Shop

By

Anisha Malhotra (CE014) (20CEUXG003)

Priyanshi Bhadresha (CE017) (20CEUOG016)

Manav Mistry (CE070) (20CEUBG074)

B.Tech CE Semester-VI

Subject: System Design Practice

Guided By

Dr. Brijesh S. Bhatt

Professor

Department of Computer Engineering

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

CERTIFICATE

This is to certify that the practical / term work carried out in the subject of System Design Practice and recorded in this journal is the bonafide work of

Anisha Malhotra (CE014) (20CEUXG003)

Priyanshi Bhadresha (CE017) (20CEUOG016)

Manav Mistry (CE070) (20CEUBG074)

of B.Tech semester VI in the branch of Computer Engineering during the academic year 2022-2023.

Dr. Brijesh S. Bhatt
Professor
Dept. of Computer Engg.
Faculty of Technology
Dharmsinh Desai University, Nadiad

Dr. C. K. Bhensdadia,
Head, Professor
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

INDEX

SR No.	Topic	Page No.
1	INTRODUCTION	4-5
2	ANALYSIS	6-11
	- Problem	7
	- Scope	8
	- Software Requirements Specifications	9
	- Use-Case Diagram	11
3	DESIGN	12
	- Class Diagram	12
4	IMPLEMENTATION	13-32
	- Tools And Technology Used	13
	- Major Modules	14
	- Data Dictionary	32
5	TESTING	33-43
6	CONCLUSION	44
7	LIMITATIONS AND FUTURE EXTENSIONS	45
8	REFERENCES	46

INTRODUCTION

Abstract:

ReTagIt is a Free Store where various clothing and household goods are available to the community at no charge. This service helps families stay afloat financially. They can use their income to pay for shelter, food and transportation while "shopping" for free clothing, shoes, bed linens, dishes, cookware, toys—even beds, couches, and refrigerators.

Inroduction:

ReTagIt :

Well, thrifting is the of process reusing pre-owned and upcycled or thrifted items of clothing and accessories. In other words means shopping at a thrift store, garage sale, or flea market for gently used products at a reduced price.

India is no stranger to thrift shopping; we may not have large stores, but there are always small pop-ups where you can find something of your liking. Thrifted objects have been cherished by a prior owner, but they are usually in decent condition and have enough life still in them to be useful to a new owner. Online thrift stores exploded in popularity in 2020, and it was a smash for all the right reasons.

ReTagIt facilitates the same purpose for a user. It allows customers to sell and purchase used but reusable goods at no cost. Customers can directly add their used item for sale with the required information and then on request communicate with the users who want to take it. In the same way, one can purchase the items available.

In summary, this easy-to-use application is best for everyone, especially youngsters of low and middle-class groups who want to use the latest items at low prices.

ReTagIt is the perfect thrifting destination for you if you're looking for some casual but trendy reusable items!

ANALYSIS

Problem:

One important detected market need is that middle and low class customers need clothing and other necessary items at low prices. In addition, a lot of individuals want to sell their used, or even new clothes or items that are not in use to them, reduce unnecessary waste, and preserve the environment. Those determined market needs can be satisfied by introducing Thrift Shops.

Scope:

ReTagIt is a web based online thrift store built to decrease the unnecessary waste of reusable items, enabling middle and low class people to buy goods at low prices.

The above goals are achieved by providing customers with a platform where they can sell and buy goods at no charge i.e. free of cost.

- The customers can also personally communicate with each other regarding the quality, size and use of the resources.

- Also the means of delivery is decided by them personally.

Software Requirements Specifications:

ONLINE THRIFT STORE

Functional requirements :

(1) Manage User

R.1.1: Register User

Description: To register any user, the details such as the name, type of user, age, mobile number, email, password, etc. will be taken from the user and it will be stored in the database and a unique user id would also be generated. Users can register either as a student or a quiz organiser.

Input: User details

Output: Unique user id

R.1.2: Login

Description: The details username/email and password would be taken from the user, then it would be validated from the record in the database and then the user would be given access to his/her account if the credentials are true.

Input: Username, email, password

Output: Account access

(2) Manage Item

R.2.1: Add Item

Description: User can add an item after login to the website

Input: Item Title, image, description, address

Output: Success message

R.2.2: Delete Item

Description: User can delete an item that hi/she added after login to the website

Input: User Selection

Output: Success message

R.2.3: Make a request to item

Description: User can make a request to the item owner

Input: User Selection

Output: Success message

R.2.4: Accept item request

Description: User can accept a request of the requested user

Input: User Selection

Output: Item deleted from the list

R.2.5: Accept item request

Description: User can reject a request of the requested user

Input: User Selection

Output: Success message

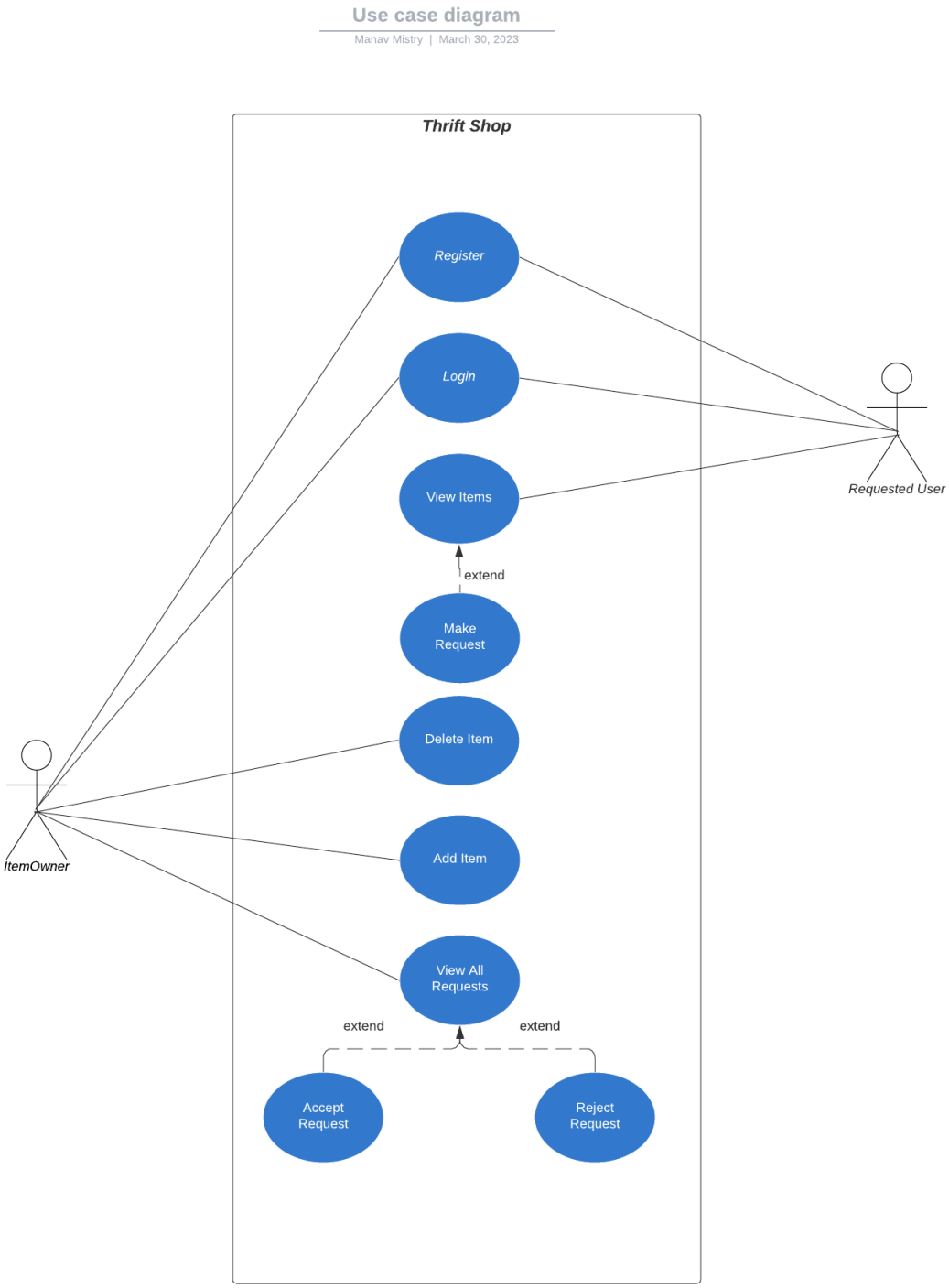
R.2.5: View items

Description: User can view all added items

Input: User Selection

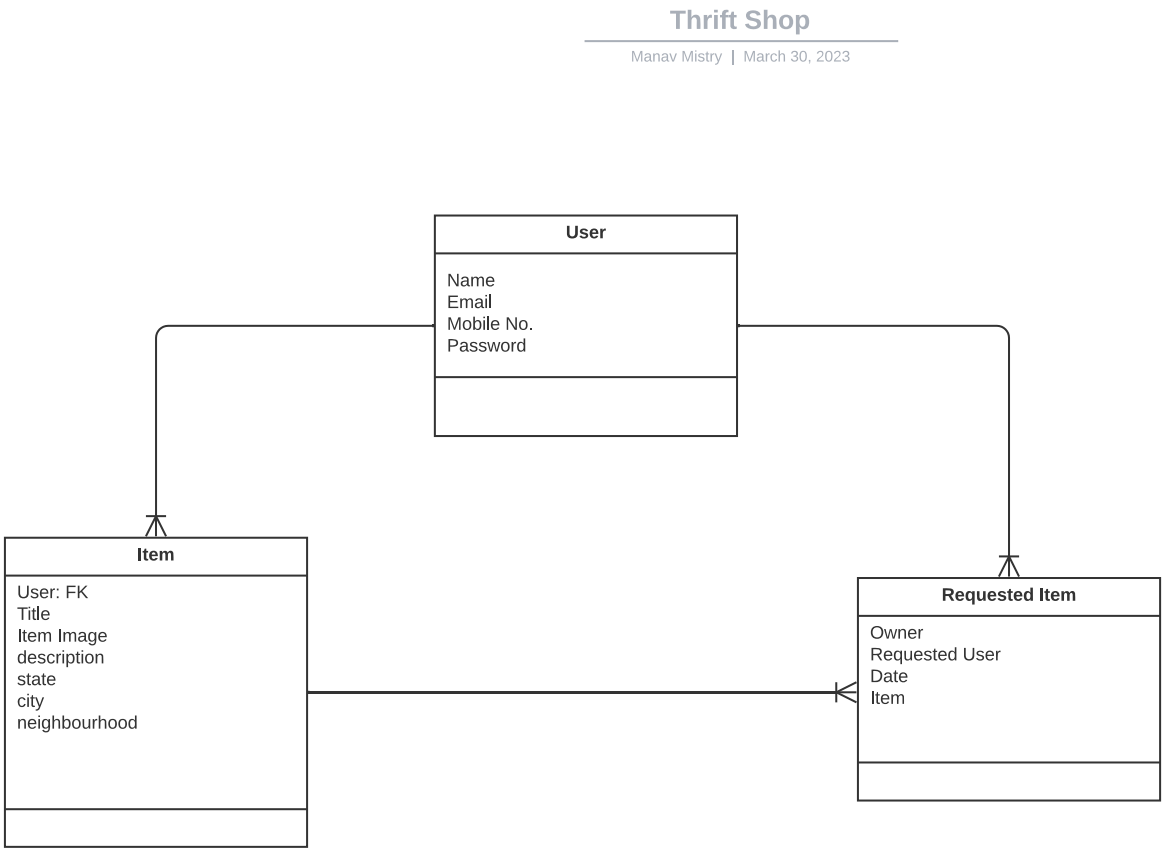
Output: items

Use-Case Diagram:



DESIGN

Class Diagram:



Describing classes:

User : User can have two roles. Item owner (if item is added by him), and requestedUser (if item is requested by user).

Item : This entity is simply an ‘item’ which is added by the user.

RequestedItem : When item is requested by any user it will be stored as ‘RequestedItem’ entity to keep track for all item that are requested. It contains details for item owner and requested user from ‘User’ entity, and item from ‘Item’ entity.

IMPLEMENTATION

Technology And Tools Used:

- ***Technology:***

  - React js
  - Redux
  - Express js
  - Node js
  - MongoDb

- ***Tools:***

  - Visual Studio Code
  - Redux DevTools
  - Git
  - GitHub
  - MonogoDB Compass
  - Postman

Major Modules:

User Module:

authSlice.js (at frontend)

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import { authService } from "./authService";

const user = JSON.parse(localStorage.getItem("user"))

const initialState = {
  user: user ? user : null,
  isError: false,
  isSuccess: false,
  isLoading: false,
  message: "",
};

// register
export const register = createAsyncThunk(
  "auth/register",
  async (user, thunkAPI) => {
    try {
      console.log(user);
      return await authService.register(user);
    } catch (error) {
      const message =
        (error.response &&
          error.response.data &&
          error.response.data.message) ||
        error.message ||
        error.toString();
      return thunkAPI.rejectWithValue(message);
    }
  }
);
```

```
// login
export const login = createAsyncThunk("auth/login", async (user,
thunkAPI) => {
    console.log(user);
    try {
        return await authService.login(user);
    } catch (error) {
        const message =
        (error.response && error.response.data &&
error.response.data.message) ||
        error.message ||
        error.toString();
        return thunkAPI.rejectWithValue(message);
    }
});

    // logout
export const logout = createAsyncThunk("auth/logout", async () => {
await authService.logout()
} )

export const authSlice = createSlice({
name: "auth",
initialState,
reducers: {
    reset: (state) => {
    state.isError = false;
    state.isLoading = false;
    state.isSuccess = false;
    state.message = "";
    },
},
extraReducers: (builder) => {
    builder
    // Register Event
    .addCase(register.pending, (state) => {
        state.isLoading = true;
    })
```

```
    .addCase(register.fulfilled, (state, action) => {
        state.isLoading = false;
        // state.isError = false
        state.isSuccess = true;
        state.user = action.payload;
    })

    .addCase(register.rejected, (state, action) => {
        state.isError = true;
        state.isLoading = false;
        state.message = action.payload;
        state.user = null;
    })

    // Login Event
    .addCase(login.pending, (state) => {
        state.isLoading = true;
    })

    .addCase(login.fulfilled, (state, action) => {
        state.isLoading = false;
        // state.isError = false
        state.isSuccess = true;
        state.user = action.payload;
    })

    .addCase(login.rejected, (state, action) => {
        state.isError = true;
        state.isLoading = false;
        state.message = action.payload;
        state.user = null;
    })

    .addCase(logout.fulfilled, (state) => {
        state.user = null
    })
},
```

```
});

export const { reset } = authSlice.actions;
export default authSlice.reducer;
```

Description:

```
const initialState = {
    user: user ? user : null,
    isError: false,
    isSuccess: false,
    isLoading: false,
    message: "",
  };
```

user : if user is logged in then it has a value of user email otherwise null

isError, isSuccess, isLoading, message : all this variables will change according to each request to backend

register request to backend:

```
return await authService.register(user);
```

above line will call the register method of **authService.js**

```
const API_URL = "http://localhost:5000/api/users"
const register = async (userData) => {
    const resp = await axios.post(API_URL, userData);
    if(resp.data){
        localStorage.setItem("user", JSON.stringify(resp.data));
        localStorage.setItem("token", resp.data.token);
    }
    return resp.data
}
```

➔ The register method of authService.js will make the post request to backed by sending the ‘userData’.

Backend controller ‘**userController.js**’:

```
const registerUser = asyncHandler(async (req, res) => {
    const {name, email, mobile, password} = req.body

    if(!name || !email || !password || !mobile) {
       return res.status(400).json({message: "please include all
fields"})
    }

    // find if already exist
    const UserExist = await User.findOne({email})

    if(UserExist) {
        res.status(400)
        throw new Error("User already exist")
    }

    //hash the password
    const salt = await bcrypt.genSalt(10)
    const hashedPassword = await bcrypt.hash(password, salt)

    //create user
    const user = await User.create({
        name,
        email,
        mobile,
        password: hashedPassword
    })

    if(user) {
        res.status(201).json({
            _id: user._id,
            name: user.name,
            email: user.email,
            mobile: user.mobile,
            token: generateToken(user._id)
        })
    } else {
        res.status(400)
        throw new Error("Invalid user data")
    }

}
```

➔ In userController.js if userData is valid and there is no such user exist with the same email then user will created to the database with hashed password and success status will return with the json object containing user details.

The return data will come in 'authSlice.js'

```
try {
    console.log(user);
    return await authService.register(user);
  } catch (error) {
    const message =
      (error.response &&
        error.response.data &&
        error.response.data.message) ||
      error.message ||
      error.toString();
    return thunkAPI.rejectWithValue(message);
  }
```

If there is status code for success then according method of extrareducer will be executed.

And the variable isLoading, isSuccess, isError, message will be changed accordingly.

```
  .addCase(register.pending, (state) => {
    state.isLoading = true;
  })

  .addCase(register.fulfilled, (state, action) => {
    state.isLoading = false;
    // state.isError = false
    state.isSuccess = true;
    state.user = action.payload;
  })

  .addCase(register.rejected, (state, action) => {
    state.isError = true;
    state.isLoading = false;
    state.message = action.payload;
    state.user = null;
  })
```

Item module:

ItemSlice.js (from frontend):

Initial variables:

```
const initialState = {
    items: [], // all added items
    isError: false,
    isSuccess: false,
    isLoading: false,
    message: "",
    itemAdded: false, // means change in number of items
    userItems: [] // logged user's added items
};
```

itemService.js (requests from frontend):

```
import axios from "axios"

const API_URL = "http://localhost:5000/api/item"
const API_URL_IMG = "http://localhost:5000/api/image"

const getAllItems = async () => {
    // get all items
    // TODO: create backend for get items
    // console.log("above get request")
    const items = await axios.get(`${API_URL}/`);
    // console.log("IN item service", items.data)
    return items.data;

}

const deleteItem = async (item) => {
    const resp = await axios.delete(`${API_URL}/${item._id}`, {
        headers: {
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })

    return resp.data
    // console.log("delete", item)
}

const getAllUserItems = async () => {
```

```
    // console.log("above get request")
    const items = await axios.get(`${API_URL}/user`, {
        headers: {
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })
    // console.log("IN item service", items.data)
    console.log(items.data)
    return items.data
}

// TODO: NEW
const addImage = async (formdata) => {
    console.log("asdasd sdasda")
    console.log(formdata)
    const resp = await axios.post(`${API_URL_IMG}/upload`, formdata,
{
        headers: {
            'Content-Type': 'multipart/form-data',
        }
    });
    console.log("response data",resp.data)
    return resp.data
}

const addItem = async (item) => {
    const user = JSON.parse(localStorage.getItem("user"))
    item.user = user.email
    console.log("item in addItem", item)

    const resp = await axios.post(`${API_URL}/`, item, {
        headers:{
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    });

    return resp.data
}

const itemService = {
    getAllItems,
    addItem,
    addImage,
    getAllUserItems,
    deleteItem
}
```

```
export default itemService
```

itemController.js (backend):

```
const asyncHandler = require("express-async-handler")

const Item = require("../models/itemModel")

const addItem = asyncHandler( async (req, res) => {
    const {user, title, description, state, city, neighbourhood,
selectedFile} = req.body

    // backend validation
    if(!user || !title || !description || !state || !city ||
!neighbourhood || !selectedFile) {
        console.log("user: ",user)
       return res.status(400).json({message: "please include all
fields"})
    }

    const item = await Item.create( {
        user,
        title,
        description,
        state,
        city,
        neighbourhood,
        selectedFile
    })

    if(item) {
        res.status(201).json({message : "Item is successfully
Added"})
    } else {
        res.status(400)
        throw new Error("Invalid user data")
    }

} )

const getItems = asyncHandler( async (req, res) => {
    // console.log("In get Items controller")
    const items = await Item.find()
    // console.log(items)
    if(items) {
        // return items;
        res.status(200).json({
```

```
            items: items
        })
    } else {
        res.status(400)
        throw new Error("No Item Found")
    }

})

const getItemsByUser = asyncHandler(async (req, res) => {
    const itemUser = req.user
    const items = await Item.find({"user": itemUser.email})
    // console.log("--- === items ",items)

    if(items) {
        res.status(200).json({
            items: items
        })
    } else {
        res.status(400)
        throw new Error("No Item By You")
    }
})

const deleteItem = asyncHandler(async (req, res) => {

    console.log(req.params.itemId)
    const deletedItem = await Item.deleteOne({"_id":
req.params.itemId})

    res.status(204).json({message: "deleted successfully"})
})

module.exports = {
    addItem,
    getItems,
    getItemsByUser,
    deleteItem
}
```

RequestedItem Model:

requestedItemSlice.js (from frontend):

initial variables:

```
const initialState = {
    r_items_all : [], // all requested items
    r_items: [], // pending items
    r_items_accepted: [], // accepted items
    isError: false,
    isSuccess: false,
    isLoading: false,
    message: "",
    item_swaped: false,
    requestSuccess: false,
}
```

requestedItemService.js (request from frontend):

```
import axios from "axios"

const API_URL = "http://localhost:5000/api/requestedItem"

const addRequestedItem = async (r_item_user) => {
    console.log("In Service",r_item_user)
    const r_item_user_seperated = {
        "item" : r_item_user.item,
        "user" : r_item_user.user
    }
    console.log(r_item_user.item, r_item_user.user)
    console.log(localStorage.getItem("token"))

    const resp = await axios.post(`${API_URL}/`,
r_item_user_seperated, {
        headers: {
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })
    return resp.data
}

const getAllPendingRequests = async () => {
    console.log("in service")
    const resp = await axios.get(`${API_URL}/pending`, {
        headers: {
```

```
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })
    return resp.data
}

const getAllRequests = async () => {
    console.log("in service")
    const resp = await axios.get(`${API_URL}/`, {
        headers: {
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })
    return resp.data
}

const getAllAcceptedRequests = async () => {
    console.log("in service")
    const resp = await axios.get(`${API_URL}/accepted`, {
        headers: {
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })
    return resp.data
}

const acceptRequest = async (r_item_id) => {
    console.log("in service",r_item_id)
    const resp = await axios.post(`${API_URL}/accept/`, r_item_id, {
        headers : {
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })
    return resp.data
}

const deniedRequest = async (r_item_id) => {
    console.log("in service",r_item_id)
    const resp = await axios.post(`${API_URL}/denied/`, r_item_id, {
        headers : {
            "authorization" : `Bearer
${localStorage.getItem("token")}`
        }
    })
```

```
    return resp.data
}

const requestedItemService = {
    addRequestedItem,
    getAllPendingRequests,
    acceptRequest,
    getAllAcceptedRequests,
    deniedRequest,
    getAllRequests
}

export default requestedItemService
```

requestedItemController.js (backend):

```
const asyncHandler = require("express-async-handler")

const RequestedItem = require("../models/requestedItemModel")
const requestedItemModel = require("../models/requestedItemModel")
const User = require("../models/userModel")

const addRequestedItem = asyncHandler(async (req, res) => {

    const {item, user} = req.body

    if( user != null || item != null) {
        console.log("In r_item controller: ",item, user)
        const date = new Date()
        const permission = false
        const isRequestComplete = false
        const owner = await User.findOne({"email": item.user})

        const requestedItem = await RequestedItem.create({
            owner: owner,
            requestedUser: user,
            item,
            date,
            permission,
            isRequestComplete
        })

        if(requestedItem) {
            console.log("request successful")
            res.status(201).json("successfully requested")
```

```
        } else {
            res.status(400)
            throw new Error("There is some error, please try again")
        }

    } else {
        res.status(400).json({message: "Please include all the
fields"})
    }
})

const getAllPendingRequestedItems = (asyncHandler( async (req, res)
=> {
    const user = req.user

    const r_items = await requestedItemModel.find({"owner.email":
user.email, isRequestComplete: false})
    res.status(200).json(r_items)
}))

const getAllRequestedItems = (asyncHandler( async (req, res) => {
    const user = req.user

    const r_items = await
requestedItemModel.find({"requestedUser.email": user.email})
    res.status(200).json(r_items)
}))

const getAllAcceptedRequestedItems = (asyncHandler( async (req, res)
=> {
    const user = req.user

    const r_items = await requestedItemModel.find({"owner.email":
user.email, permission: true})
    res.status(200).json(r_items)
}))

const acceptRequest = (asyncHandler(async (req, res) => {
    const { r_item_id } = req.body
    const r_item = await requestedItemModel.findOne({_id:
r_item_id});

    r_item.isRequestComplete = true
    r_item.permission = true
    await requestedItemModel.updateOne({_id:r_item_id},r_item)
    res.status(201).json({message: "successfuly accepted"})
}))
```

```
const deniedRequest = (asyncHandler(async (req, res) => {
    const { r_item_id } = req.body

    // get the r_item with id r_item_id
    const r_item = await requestedItemModel.findOne({_id:
r_item_id});

    r_item.isRequestComplete = true
    r_item.permission = false
    await requestedItemModel.updateOne({_id:r_item_id},r_item)
    res.status(201).json({message: "successfuly accepted"})

}))

module.exports = {
    addRequestedItem,
    getAllPendingRequestedItems,
    acceptRequest,
    getAllAcceptedRequestedItems,
    deniedRequest,
    getAllRequestedItems
}
```

Data Dictionary:

User

Field	Datatype	Constrain
user_id	String	Primary Key
Name	String	-
Email	String	-
Mobile	Number	-
Password	String	-

Item:

Field	Datatype	Constrain
item_id	String	Primary Key
user	User	Foreign Key
Title	String	-
Description	String	-
State	String	-
City	String	-
Neighbourhood	String	-

Requested Item:

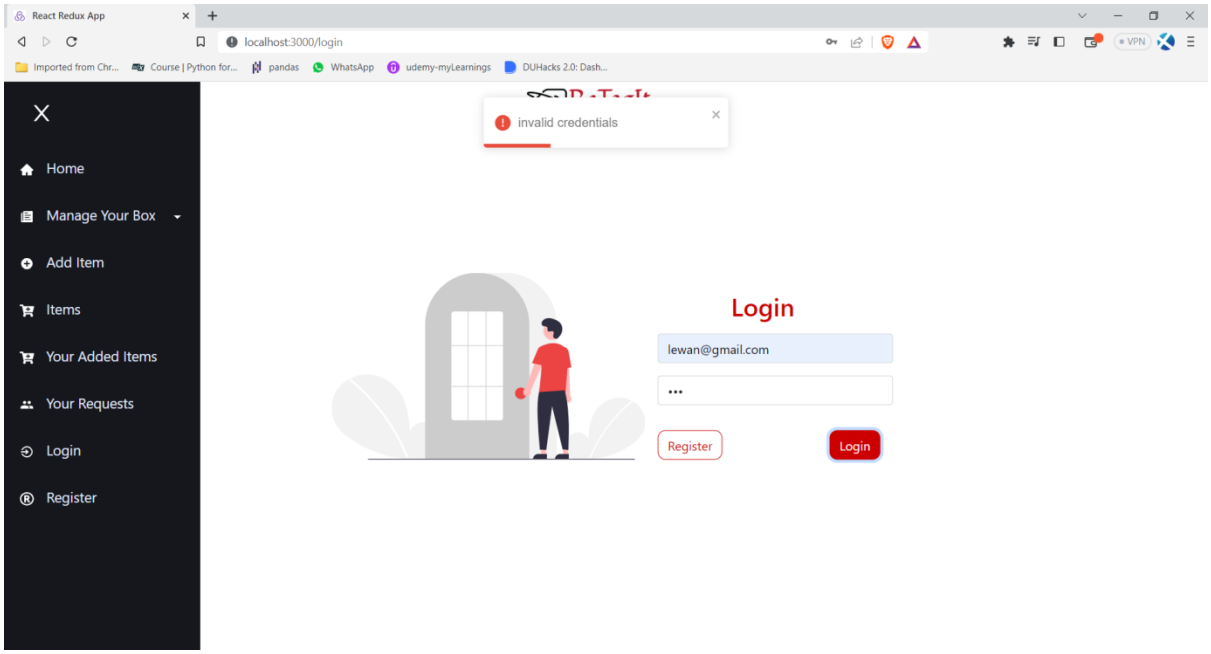
Field	Datatype	Constrain
Requested_item_id	String	Primary Key
owner	User	Foreign Key
Requested User	User	Foreign Key
Date	Date	-
Item	Item	Foreign Key
Permission	Boolean	-

TESTING

Test case name	Description	Precondition	Steps	Test data	Expected result	Actual result
Login user	Wrong credentials	-	Enter the value for email and password	Email and password	Invalid credentials error	Invalid credentials error
Login user	Correct credentials	-	Enter the value for email and password	Email and password	Redirect to home page	Redirect to home page
Register User	Already registered email	-	Enter registration details	Registration details	Error: already exist emailId	Error: already exist emailId
Register User	On valid registration details	-	Enter registration details	Registration details	Redirect to home page with login (No need for login)	Redirected to home page
Register User	When Password and confirm password not matched	-	Enter password and confirm password	Password and confirm password	Error: passwords are not matching	Error: passwords are not matching
Add Item	On valid Item details	User must be logged in	Enter the item details	Item details	Item successfully added	Success message
View your added items	View all items added by logged user	User must be logged in	Button click	-	Display all items	Displaying all items
View your added items	View all items added by user (not logged in)	-	Button click	-	Error: permission denied	Error: Please login first to access
Delete your added items	Item owner can delete the item if he wants to make item unavailable for request	User must be logged in	Button click	-	Item deleted successfully	Item deleted successfully
View your requests	View all requests by logged user	User must be logged in	Button click	-	Display all requested items	Displaying all requested items

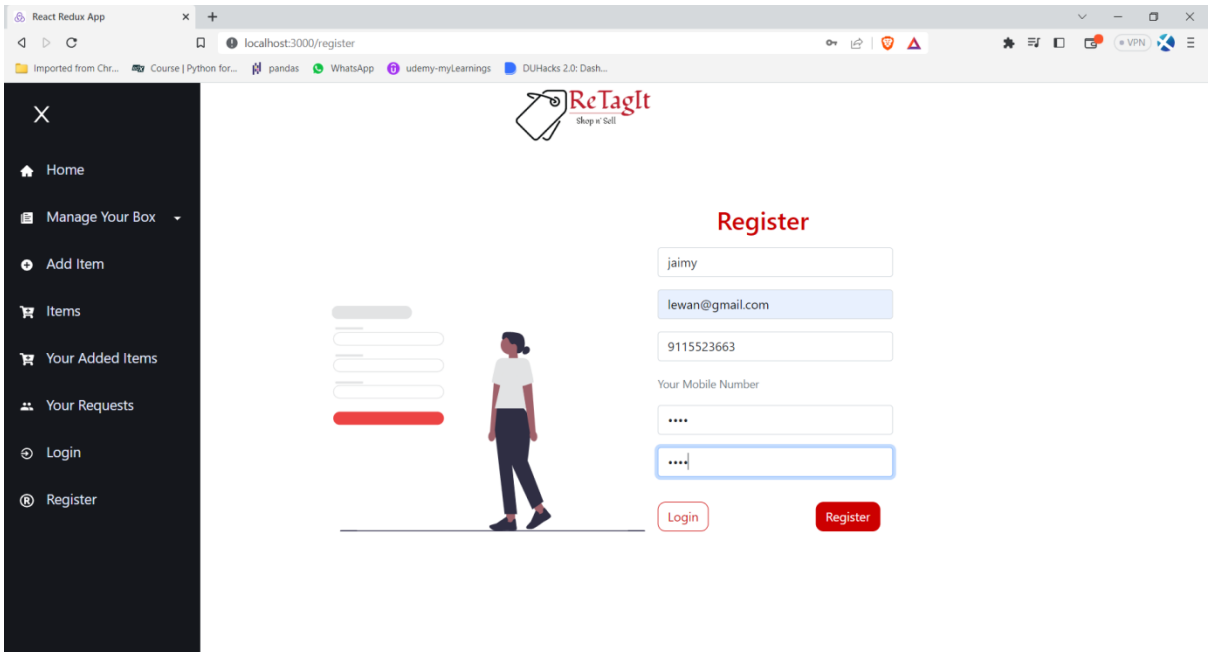
Test case name	Description	Precondition	Steps	Test data	Expected result	Actual result
View your requests	View all requests by user (not logged in)	-	Button click	-	Error: permission denied	Error: Please login first to access
Accept request	Item owner can accept request	User must be logged in	Button click		Item successfully accepted and should be removed from pending requests list, and removed from the `view all item list ` (items which are available for request)	Item successfully accepted and removed from pending requests list, and removed from the `view all item list ` (items which are available for request)
Reject request	Item owner can reject the request	User must be logged in	Button click		Item successfully reject, and removed from the pending requests of user, and still available for request for other users	Item successfully reject, and removed from the pending requests of user, and still available for request for other users

Login user testing:

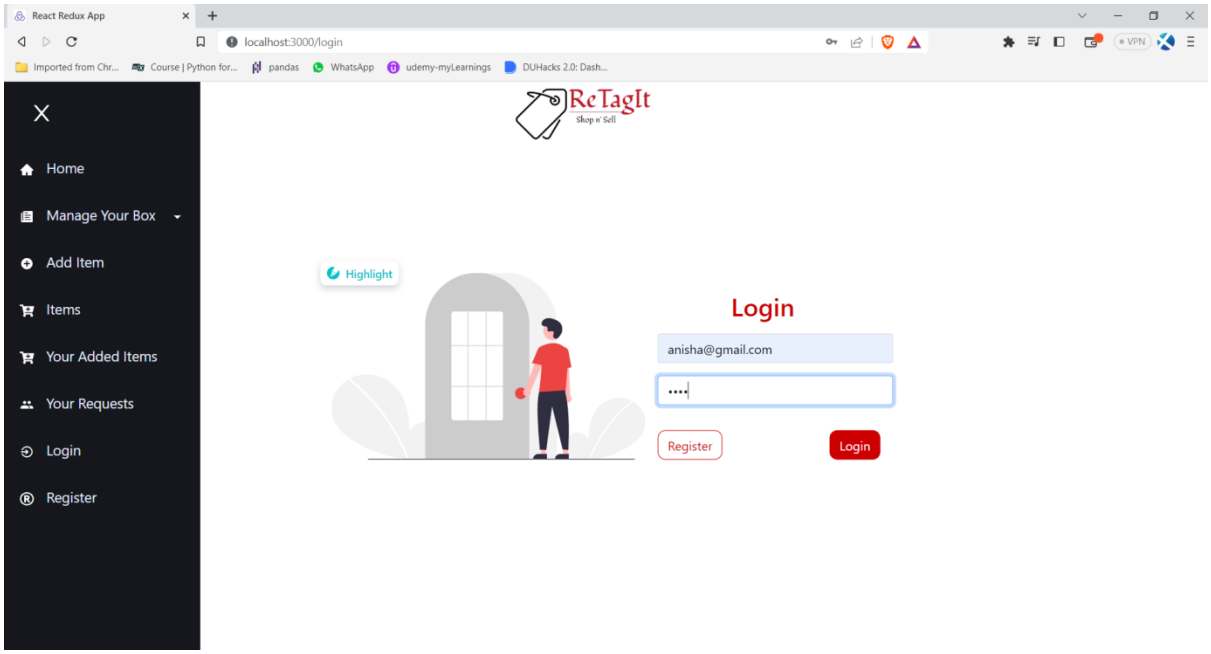


After successful login or registration:

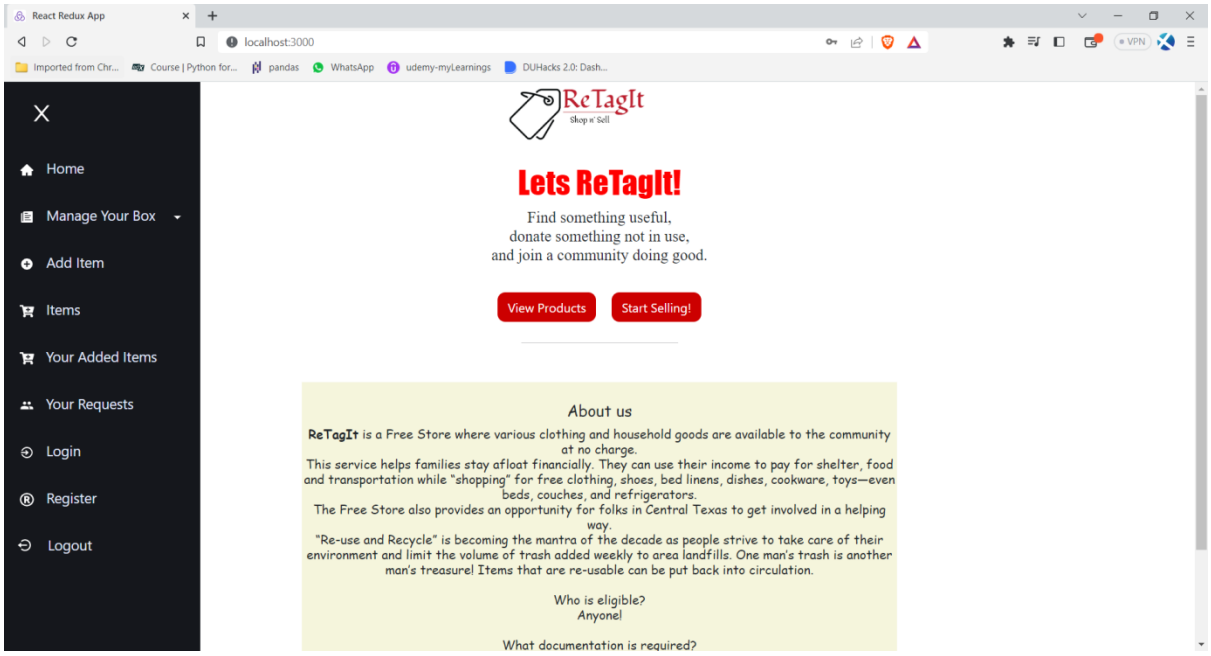
Successful registration:



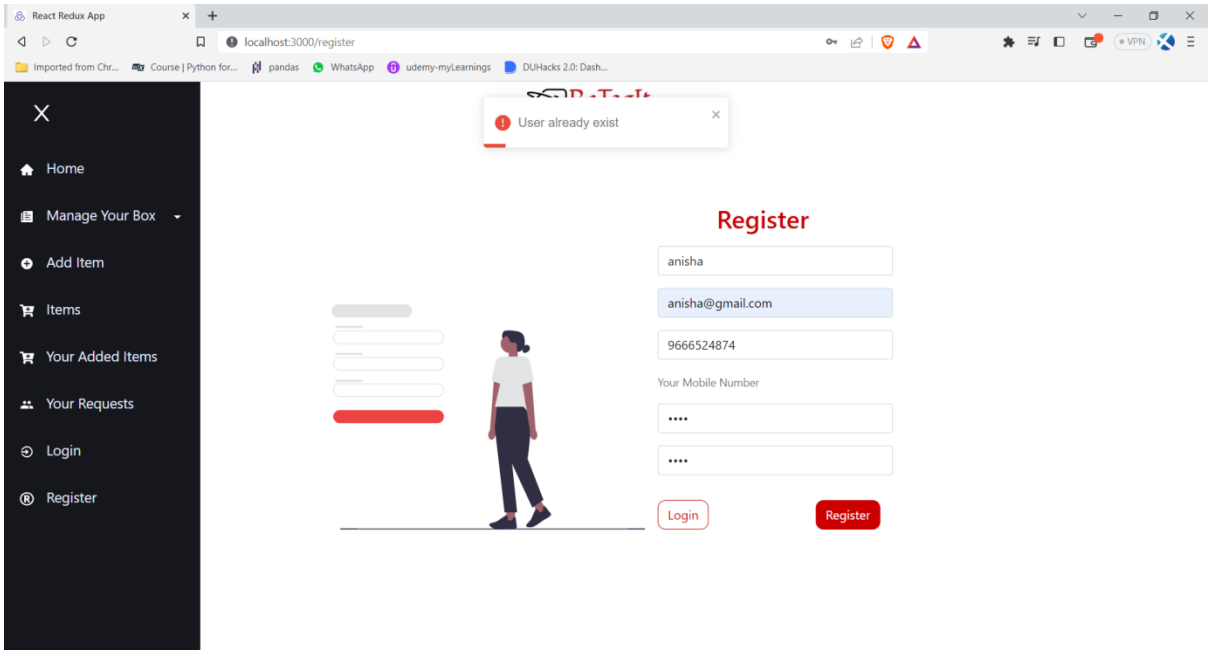
Successful login:



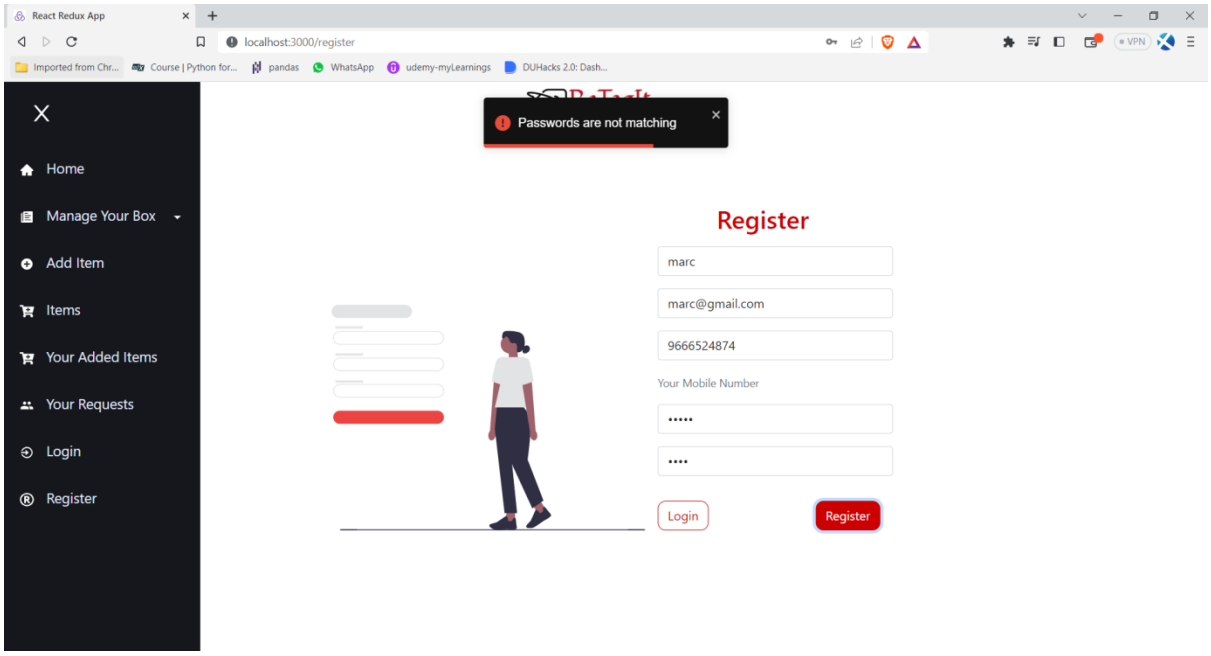
Redirect to home page:



Register with already existed user email:

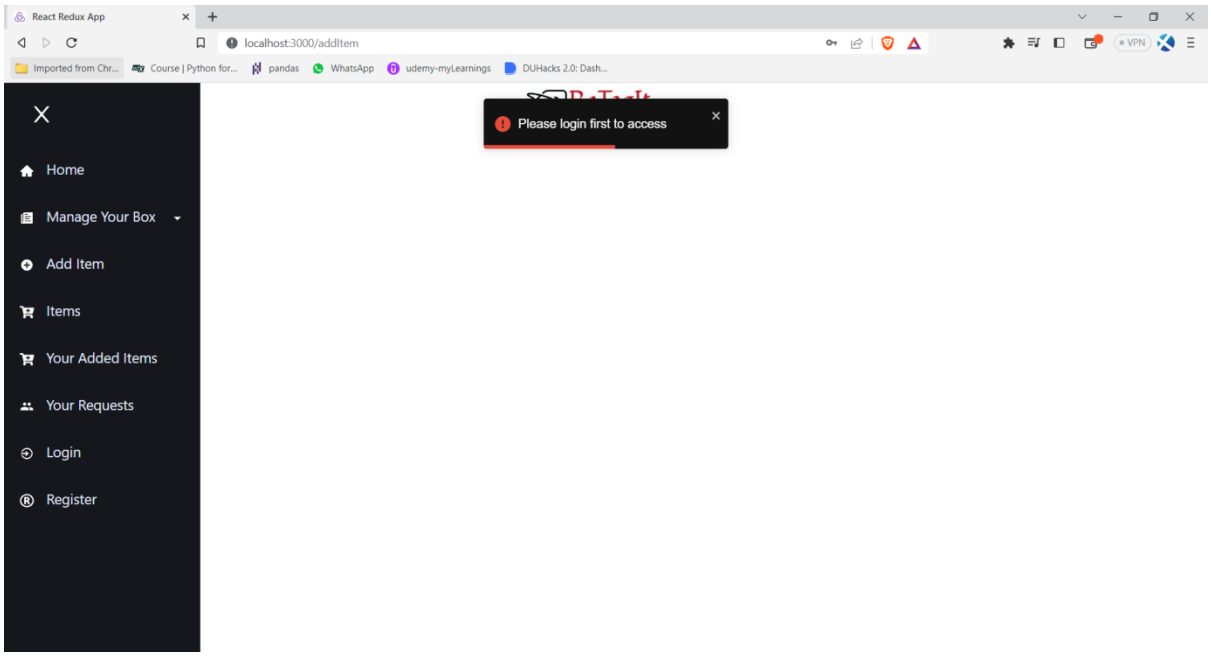


Password match testing in register user:

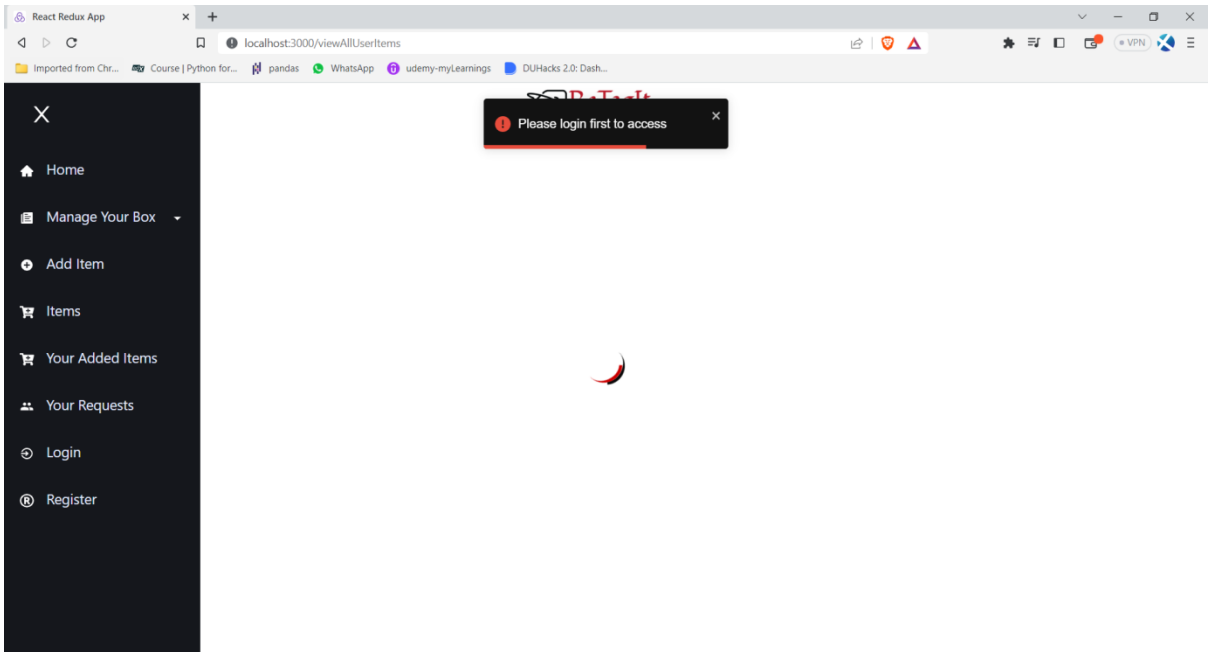


All page request without login:

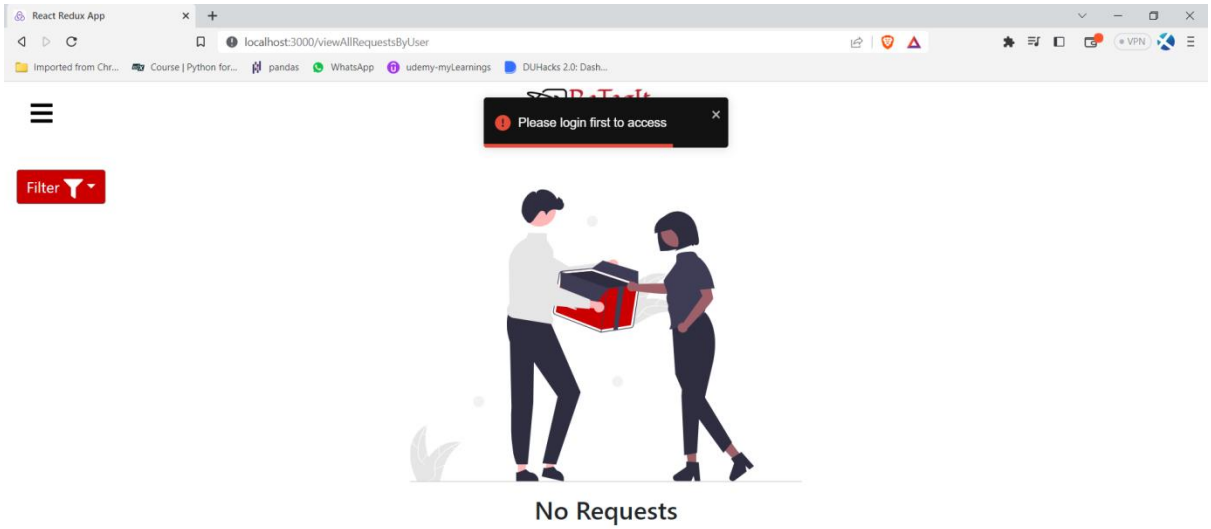
Accessing Add Item page without login:



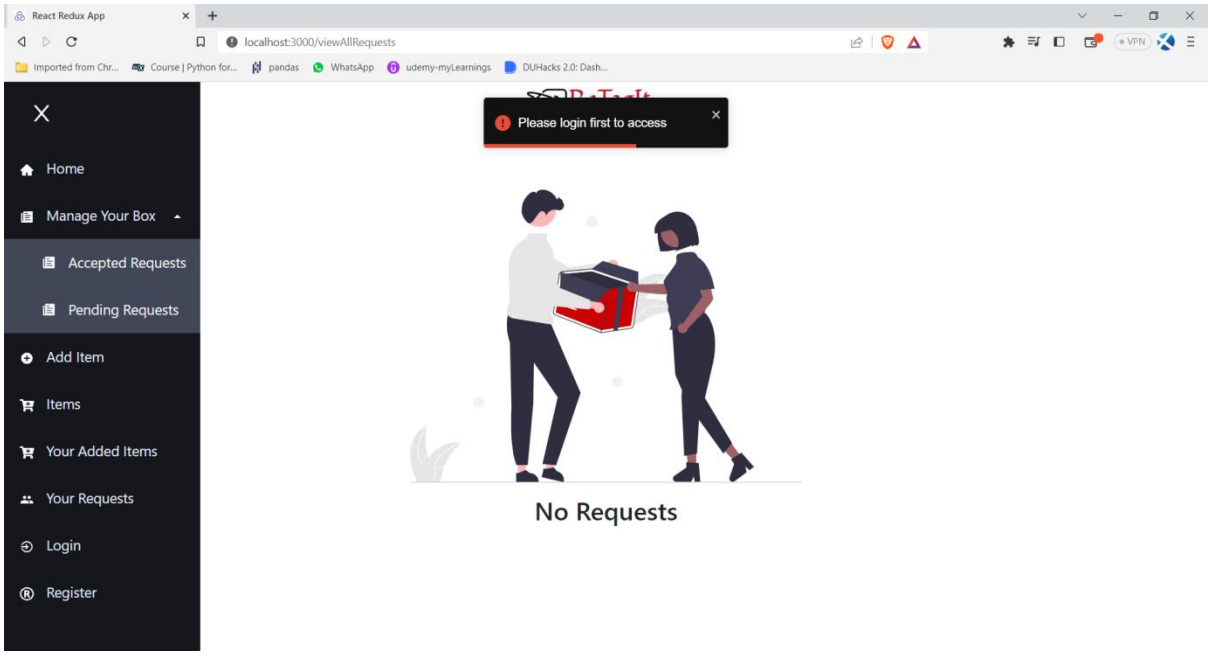
Accessing Your ‘Added Item’ page without login:



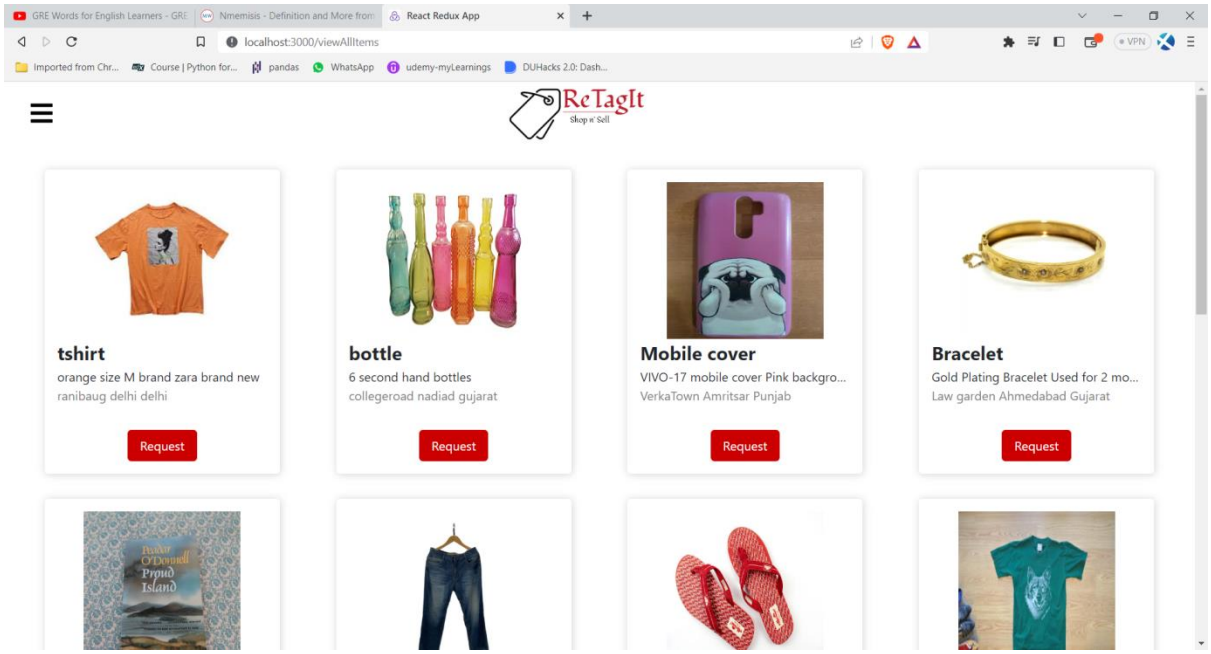
Accessing ‘own Requests’ page without login:



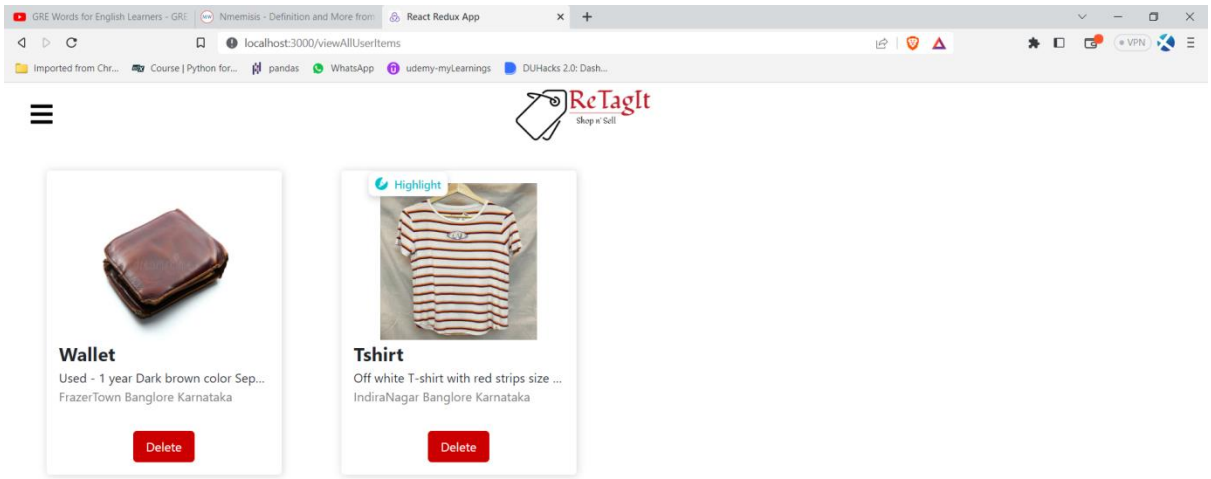
Accessing ‘other user request’ without login:



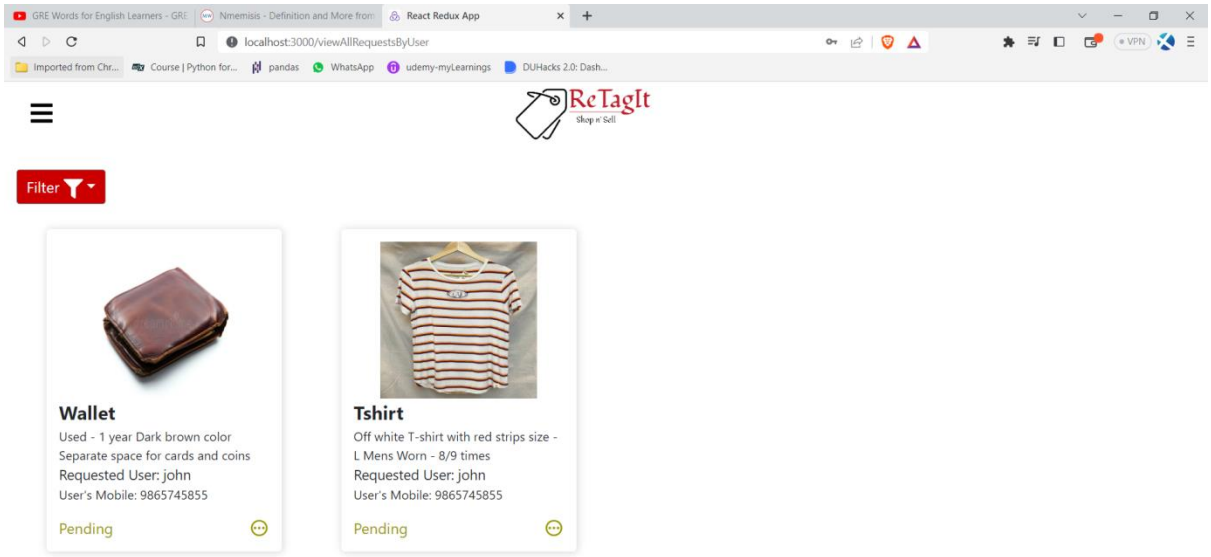
‘View items’ page (login is required):



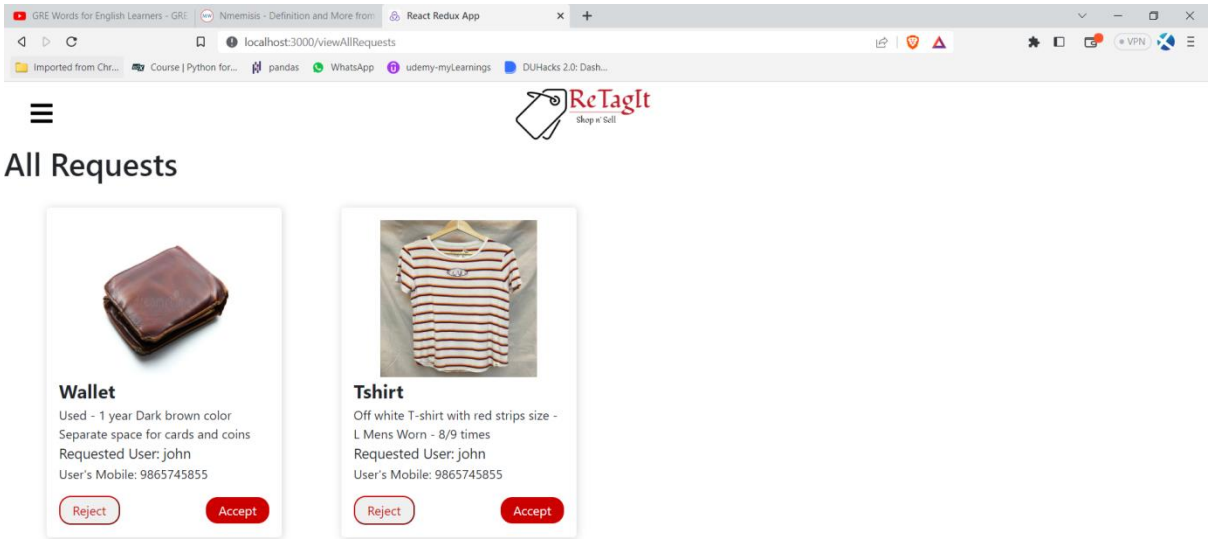
‘Your added items’ page (owner’s):



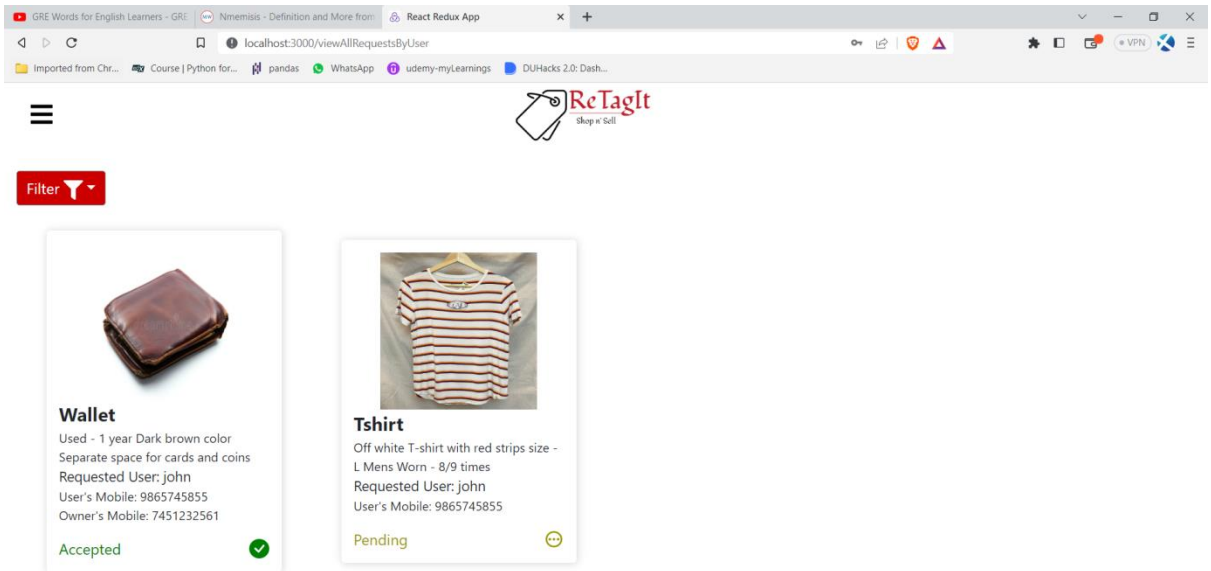
‘your requests’ page (requested user’s):



‘view all requests’ page (owner’s) :



‘your requests’ page (requested user’s):



Conclusion:

- By implementing the above project, the main aim of thrifting reusable items is achieved.
- The system is able to increase the chance of middle, and low-income people to buy, sell and exchange various reusable items at almost no cost.
- The system is successful in encouraging our customers to resell their used or even unused items to the ones in need.
- Functionalities:
    - Users can request the item from the item owner
    - Item owner can accept or reject the request
    - The contact details of the item owner can only be displayed to the requested user when the request is accepted by the item owner

Limitations and future extension of system:

Limitations:

- User can not add quantity
- No track of item is actually received or not

Future extension:

- Chat system will be added between owner and requested user
- Search system will be implemented

References:

https://stackoverflow.com/

https://github.com/devconcept/multer-gridfs-storage

https://github.com/aheckmann/gridfs-stream