

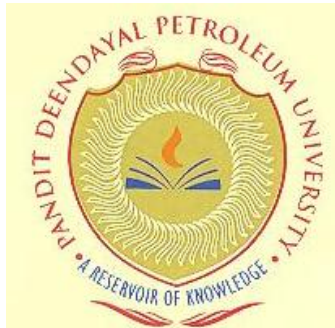
# INVOICE AND PURCHASE ORDER SYSTEM

*Report submitted in partial fulfillment of the requirements for  
the  
B. Tech. degree in Computer Science & Engineering*

By

*1. PAREKH VISHALKUMAR      16BCP026*

**Under the supervision  
Of  
Dr. RUTVIJ JHAVERI**



**SCHOOL OF TECHNOLOGY  
PANDIT DEENDAYAL PETROLEUM UNIVERSITY  
GANDHINAGAR, GUJARAT, INDIA  
May/June 2020**

# CERTIFICATE

This is to certify that the report on “INVOICE AND PURCHASE ORDER SYSTEM” submitted by the students, as a requirement for the degree in Bachelor of Technology (B. Tech) in Computer Science & Engineering, under my guidance and supervision for the session 2019-2020.

**Name of the student**

**Roll No.**

**Signature**

*1. PAREKH VISHALKUMAR      16BCP026*

Date:

Signature of the

Supervisor

Place:

(Name of the Supervisor)

## List OF Tables

Table No	Description	Page No.
3.2.1	Invoice Schema Table	10
3.2.2	Organization Schema Table	11
3.2.3	Purchase Order Schema Table	12
3.2.4	Users Schema Table	13
3.2.5	Group Schema Table	13
3.2.6	Roles Schema Table	13
3.2.7	Rights Schema Table	14

## List OF Figures

Figure No	Description	Page No.
3.1	Administrator Use Case.	5
3.2	Operator Use Case	6
3.3	Approval Manager Use Case	7
3.4	Accountant Use Case	8
3.5	Auditor Use Case	9
3.6	CFO Use Case	10
4.1	Project Explorer Shows Two Microservices	15
4.2	User Services APIs and node_modules contain dependencies	16
4.3	pacakge.json file	17
4.4	app-admin.js files small snapshot which contains reference to SignUp and Login.	18
4.5	Small screenshot of user.js	20

4.6	users.js file which contains database schema for users table in MongoDB	20
4.7	Project Structure of Invoice and PO service	22
4.8	Front-End project set up using React App	24
4.9	App.js describing router	25
4.10	User Registration Form (URL: localhost:3000/userReg)	26
5.1	Result of Test Case 1 in the postman	29
5.2	Result of Test case 2 in Postman showing error: 409 conflict error	30
5.3	Result of Test Case 3 in the postman	30
5.4	Output of Test Case 4	31

## CONTENTS

### Abbreviations

<b>1. Chapter 1 - INTRODUCTION</b>	<b>1</b>
➤ 1.1. Purpose	1
➤ 1.2. Scope	1
<b>2. Chapter 2</b>	<b>2</b>
➤ 2.1. Proposed System	2
➤ 2.2. Specific Requirements	2
➤ 2.3. Tools and Technologies	3
➤ 2.4. Advantages offer by proposed system	4
<b>3. Chapter 3 – Design</b>	<b>5</b>
➤ 3.1. Use case diagrams	5
▪ 3.1.1. Administrator	5
▪ 3.1.2. Operator	6
▪ 3.1.3. Approval Manager	6
▪ 3.1.4. Accountant or Finance Controller	8
▪ 3.1.5. Auditor	9
▪ 3.1.6. CFO	9
➤ 3.2. Data Dictionary	10
▪ 3.2.1. Invoice Schema	10
▪ 3.2.2. Organization Schema	11
▪ 3.2.3. Purchase Order	12
▪ 3.2.4. Users	13
▪ 3.2.5. Group	13
▪ 3.2.6. Roles	13
▪ 3.2.7. Rights	14
<b>4. Chapter 4 – Implementation</b>	<b>15</b>
➤ 4.1. Serve- Side Implementation	15
▪ 4.1.1. User Services	16
▪ 4.1.2. PO_and_Invoice_Service	21
➤ 4.2. UI Implementation	23
<b>5. Chapter 5 – Testing</b>	<b>28</b>

➤ 5.1. Unit Testing	28
▪ 5.1.1. User Microservice Testing	28
▪ 5.1.2. Invoice and PO service Testing	32
➤ 5.2. Integrated Testing	33
<b>6. Chapter 6 – Summary</b>	<b>35</b>

## **Abbreviation**

- AWS – Amazon Web Services
- CFO- Chief Finance Officer
- CSS – Cascading Stylesheet
- CST- Central Sales Tax
- Doc – Documents (P.O and Invoices)
- GST- Goods and Service Tax
- HSN- Harmonized System Nomenclature
- HTML – Hyper Text Markup Language
- NPM – Node Package Manager
- PAN- Permanent Account Number
- P.O – Purchase Order
- TDS- Tax deducted at source
- TIN- Taxpayer Identification Number
- VAT- Value Added Tax

## **Chapter 1 Introduction**

### **1.1 Purpose**

- In this digital era, everyone is using the internet and utilize it to ease their lives. Traditional accounting works using a lot of physical entities like a pen, paper, stamp, etc. To manage all this physical record is a difficult task. Sometimes this physical record might get damage. The approval process of the invoice is time taking. There is a chain of the approval process and sometimes invoice lost or tempered by a person. To overcome this problem with the traditional invoice and P.O approval process, this System transforms all this task digitally using the internet and its services.

### **1.2 Scope**

- Organization Registration and its employee registration.
- Creation of group as per the Organization's requirement and volume.
  - Role creation for each group.
  - Role assignment to employees.
  - Assignments of rights and authentication to each employee based on their role.
- Generate dynamic template for P.O and invoices as per client's and/or Organization requirement.



## Chapter 2

### 2.1 Proposed System

- To develop a functional and user interactive web application that can ease business operations like placing a purchase order and invoice approval and tracking system. It also manages the history of employee performance and provides a helping hand at appraisal time. It also reduces dependences on physical entities and modulates all physical recodes digitally. It is cloud base system which performs all the operation on the cloud so, it improves the security of system and store data on the cloud. In the future, it can collaborate with the marketplace and provide trusted vendors and publish your product on the system. It also monitors all financial activities in specific departments.

### 2.2 Specific Requirements

- Hardware Requirement:
  - This system is a web-application so, users need to have a computer with a stable internet connection.
  - The computer system requires the latest version of the chrome or Mozilla Firefox version.
  - For a better experience of this product, computers have at least 4GB ram.
- Software Requirement:
  - This system is using Node.js and its framework.

- Node.js Version 12.16.3.
- Node.js Dependencies
  1. Express Framework (For Routing Operation)
  2. Mongoose Framework (For Database)
  3. Nodemon Library (For Server Deployment efficient)
  4. Jsonwebtoken (For authentication)
  5. Bcrypt (For Password Protection)
  6. Body-Parser (For Request Body Parsing)
- This system store data using MongoDB.
- MongoDB version 4.2.6.
- For hosting UI the system is using React App.
- React App provide development and hosting environment for single page application.
- Git is used for version control and maintenance of system.
- Git version 2.26.2.windows.1

## 2.3 Tools and Technologies

- Technologies To be Used:
  - For server-side operation and all business, logic is written in Node.js using its express framework.
  - It uses the principle of microservice.
  - This microservice concept makes server-side deployment efficient.
  - The front end of this application developed using React.js and Bootstrap.

- The database of this developed in MongoDB using Node.js's mongoose framework.
- Tools To be Used:
  - Visual Studio Code is used for the development of the system.
  - Postman is used for API testing
  - For Front end testing Google Chrome and Mozilla Firefox used.
  - For React.js development React App used. It is a predefined set up for the development of the React.js project.
  - The version control system is git for local system and for remote storage Git-hub used.

## **2.4 Advantages offered by proposed system**

- This digital system reduces the usage of physical entity like paper, pen, etc.
- The system makes financial operation easier and safer and time efficient.

## Chapter 3 Design

### 3.1 Use Case Diagram

**3.1.1 Administrator:** is responsible for registering employee to the system.

- Create a new account to system and assign to new employee, assign appropriate rights to that role.

Name of Use case: Create a new account.

Description: establish new account for employee.

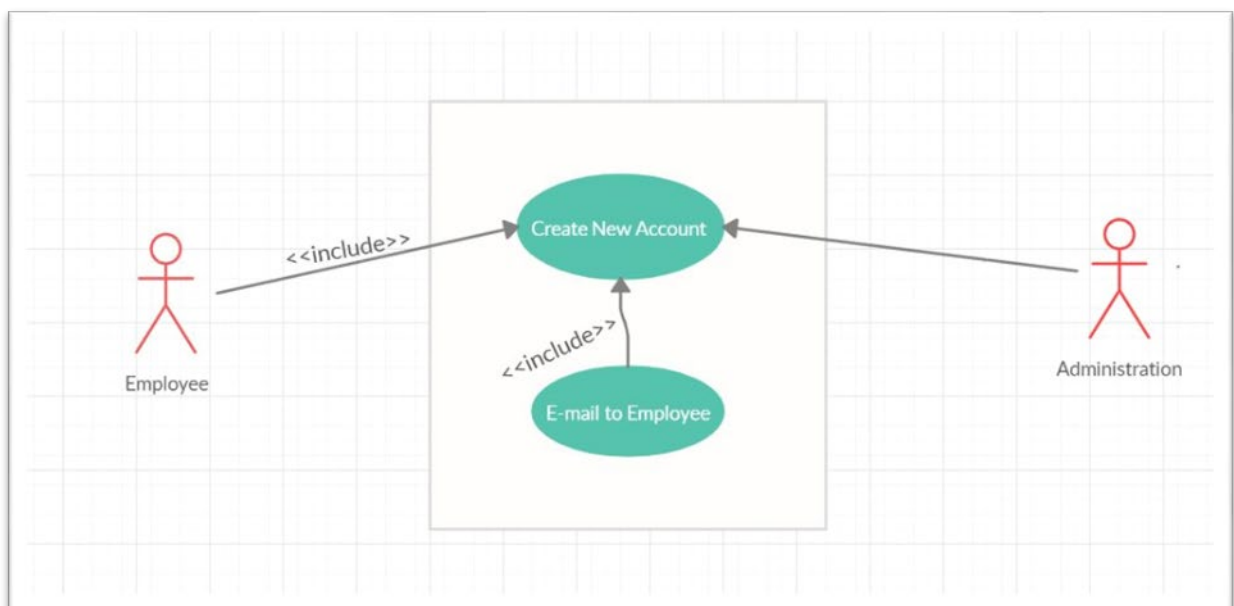
Pre-Conditions: Admin should be logged in the system.

Flow of Events:

- A form will open, enter credential of particular employee and assign role in existing workflow and give appropriate rights to that role.
- A query fired to the database.
- Now employee received welcome e-mail which contains user name and password of account.

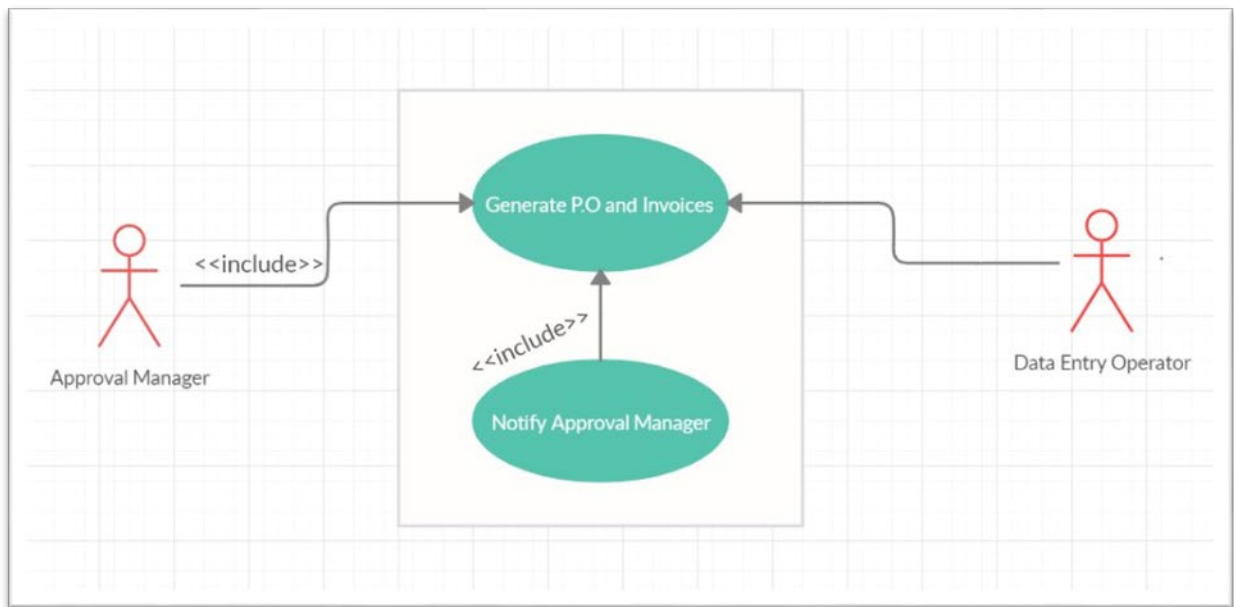
**Diagram 3.1: Administrator Use Case.**

#### 3.1.2 Operator:



- Operator is responsible for create template for P.O and invoices and add data to the system.
- Their main objective is data entry.
- They generate P.O and invoices and send to Approval Manager.

**Diagram 3.2 Operator Use Case**



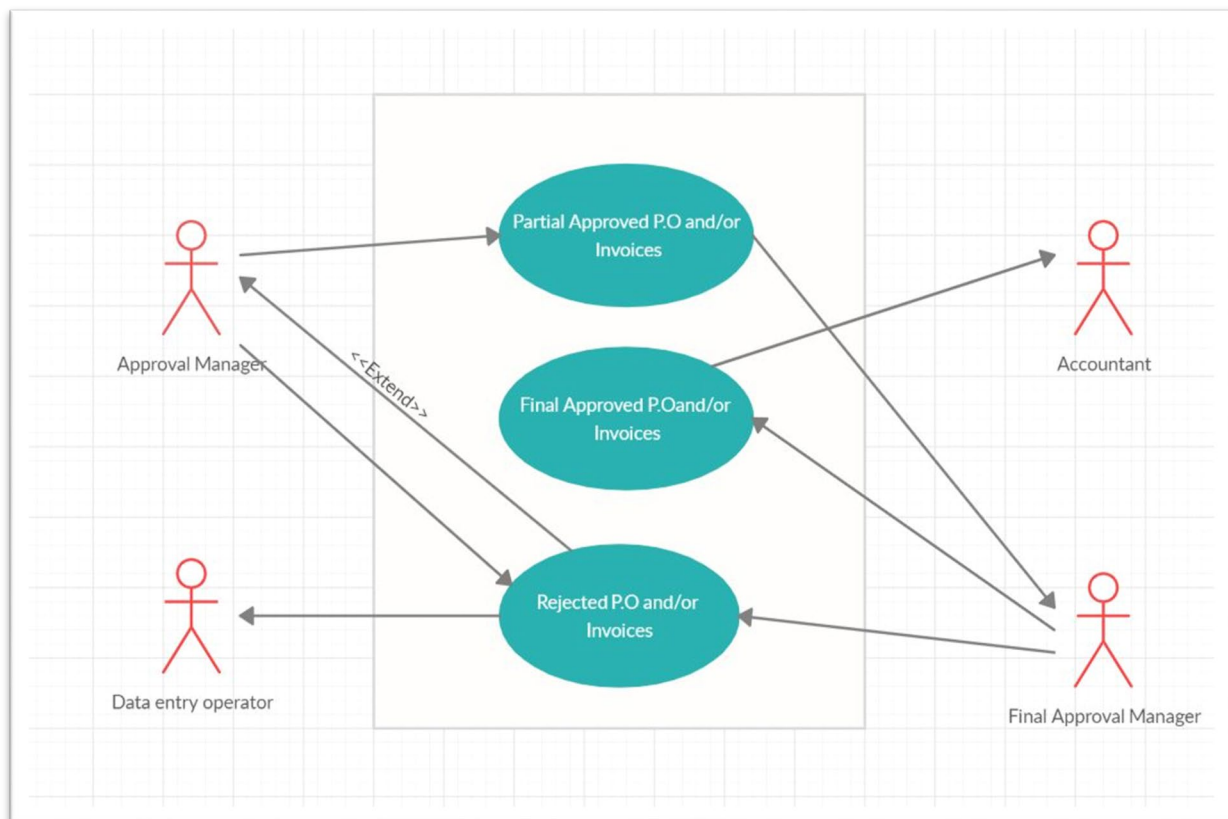
### 3.1.3 Approval Manager:

- Approval manager have authority to approve P.O and/or invoice.
- Number of Approval Manager vary across organizations. Some organization have 2 approval-manager.  
e.g. 1. Junior manager approves P.O and invoices of 0-50,000 Rs. He has full authority till amount of 50,000.
  - If amount exceeds 50,000 then he needs to send docs to the senior manager who have further rights on specific interval say

50,000-500,000 and so on there could be chain of approval managers.

- It is an example of joint authority.
- 
- Approval manager may have absolute authority or joint authority with other managers.
  - Absolute authority means managers have specific category for example IT related P.O then they need not to consult with other managers.
  - Once manager approve docs then they send docs to account department.
  - If is there query in docs then they sends back to data entry operators or reject P.O.

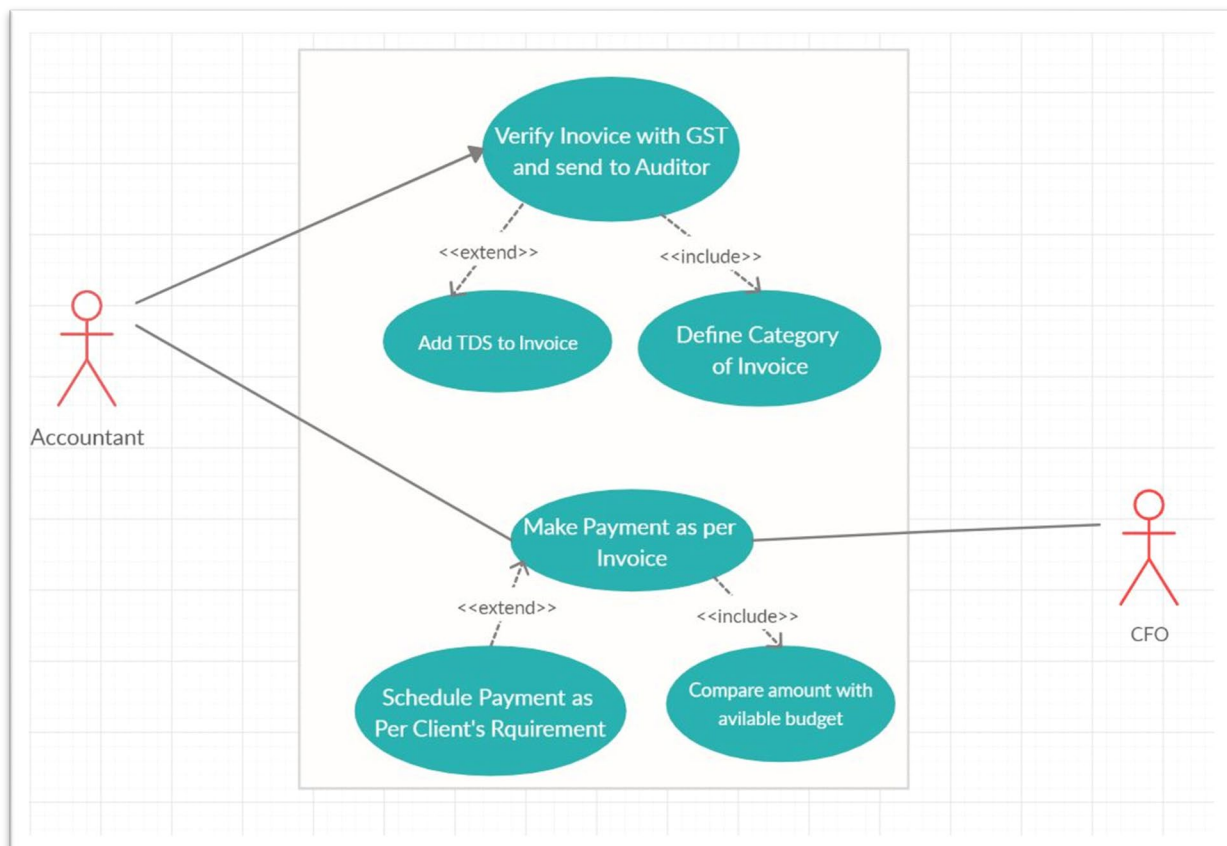
**Diagram 3.3: Approval-Manager Use Case.**



### 3.1.4 Accountant or Finance Controller:

- Once account received docs form approval manager then he needs to add TDS amount and assign that docs to appropriate category.
- Then sends those docs to auditor for further processing.
- When CFO clear docs for payment then accountant make payment as per invoice using specific method of payment mentioned in invoice.
- Account gets record of pending payment from system and compare it with a amount of money that he entitle for use.
- So, accountant schedule payment in system as per requirement of clients.

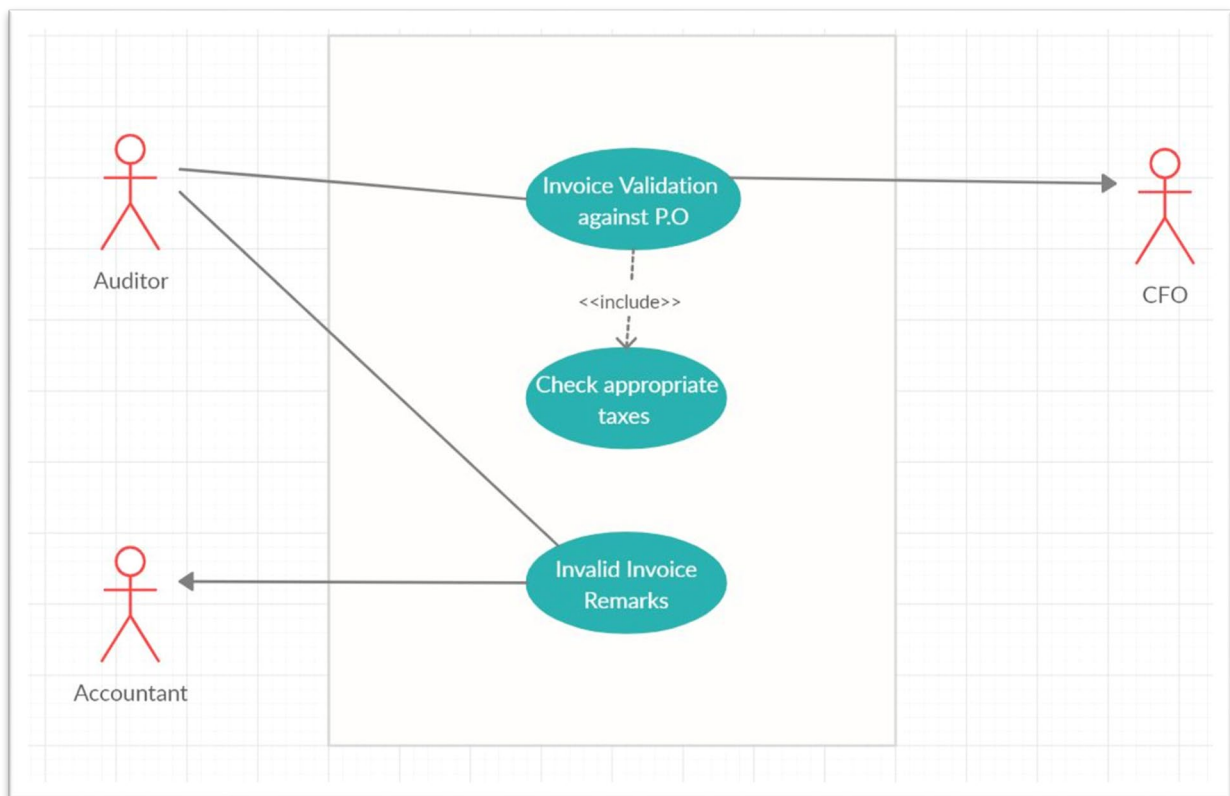
**Diagram 3.4: Accountant Use Case**



### 3.1.5 Auditor:

- Auditor checks invoices against P.O and evaluate that is there any extra amount that added to invoice also check the taxes that included in invoice.
- If auditor found any malicious amount in invoice against P.O or found that some taxes that account missed out then reject invoice and send back to accountant for clarification.
- If auditor clears invoice then CFO received copy of invoice.

**Diagram 3.5: Auditor Diagram**



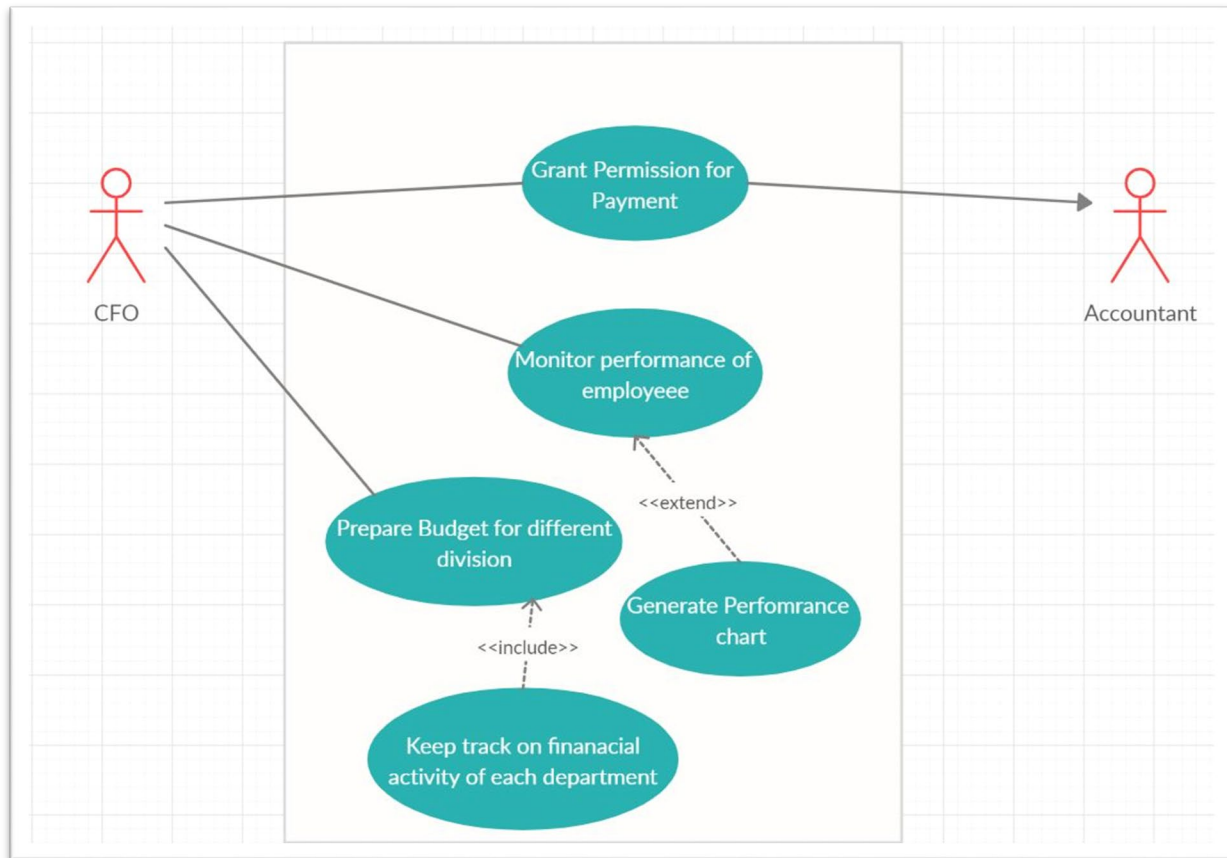
### 3.1.6 CFO

- CFO is a final entity in this workflow who granted payment for particular invoice or take decision about P.O.
- CFO also monitor dashboard of employees and view annual performance of employee.



- He can pass financial budget of numbers of department based on their requirement and also assign same rights to lower authority say regional head.

**Diagram 3.6: CFO Use Case**



## 3.2 Data Dictionary

### 3.2.1 Invoice Schema:

Attributes	Type
orgId	System Generated
user_id	System Generated
orgName	String
username	String
inv_no	String

supplier_name	String
supplier_id	Schema.Types.ObjectId
PO_number	String
PO_id	System Generated
bill_number	String
isExpense	Boolean
Product_Nature	String
HSN_code	String
Quantity	Number
Rate	Number
additionalRate	Number
VAT	Number
CST	Number
GST	Number
service_tax	Number
excise	Number
amount	String
create_date	Date, default : Date.Now
Payment_due_date	String
vendor_selection:selected_by	String
vendor_selection:divison	String
doc_attachment:invoice	String
doc_attachment:PO	String
doc_attachment:other doc	String
iv_status:status	String
iv_status:description	String
iv_status:distribureTo	String
iv_status:changeDate	String
iv_status:changedBy	String

### 3.2.2 Organization Schema:

Attributes	Types
orgName	String
ceoName	String
orgPAN	String
orgGST	String
orgTIN	String
service tax no	String
Export import regd no	String
Form_regd_no	String

orgAddr1	String
orgAddr2	String
orgCity	String
orgDistrict	String
orgState	String
orgCountry	String
orgStatus	String
Join date	Date
subscriptionId	String

### 3.2.3 Purchase Order:

Attribute	Types
orgId	System Generated
userId	System Generated
orgName	String
supplier name	String
PO number	String
supplierId	Schema.Types.ObjectId
supplier_name	String
po_status:status	String
po_status:status_description	String
po_status:status_changedBy	String
po_status:distributeTo	String
po_status:status_changeDate	String
userName	String
create date	Date, default: Date.now
product information:product name	String
product information:poCategory	String
product information:product_description	String
product information:HSN_code	Number
product information:quality	Number
product information:rate	Number
product information:VAT	Number
product information:CST	Number
product information:GST	Number
product information:service tax	Number
product information:excise	Number
product information:amount	String
product information:payment terms	String
vendor selection:selectedBy	String
vendor selection:division	String

budget_and_approvals:location	String
budget_and_approvals:budget head	String
budgets_and_approvals:period	String
doc_attachment:cancelled cheque	String
doc_attachment:quotation	String

### 3.2.4 Users:

Attributes	Types
firstName	String
lastName	String
email	String
password	String
organization	String
orgId	System Genertated
role	String
join_date	Date,default:Date.now
subscriptionId	String

### 3.2.5 Group

Attributes	Types
grpId	System Generated
grpName	String
status	String
Regd_date	String
organization	String
uid	String

### 3.2.6 Roles

Attributes	Types
roleId	System Generated
roleName	String

### 3.2.7 Rights

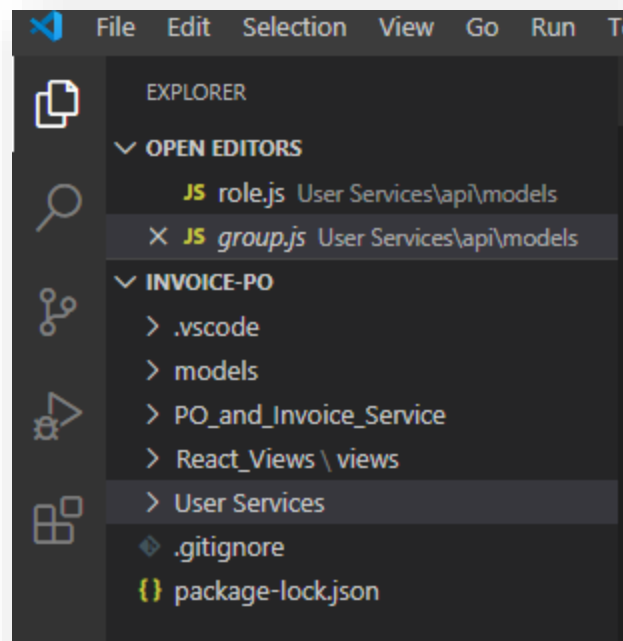
Attributes	Types
rights id	String
rights description	String

## Chapter 4 Implementation

### 4.1. Serve- Side Implementation

- The server-side development uses the principle of microservice.
- Microservice Principle: Microservice is an architectural style that structures an application as collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small-team
- This system currently running on two microservice.
  1. User Services
  2. PO\_and\_Invoice\_Service

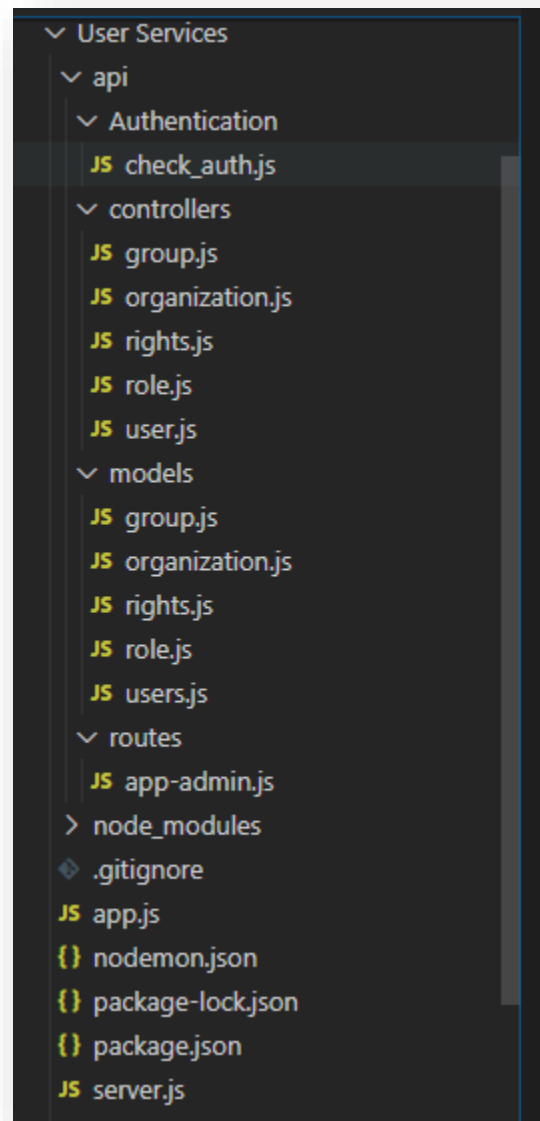
**Diagram 4.1 Project Explorer Shows Two Microservices.**



**4.1.1. User Services:** This microservice consist of all user related operation.

- This service contains all APIs, dependencies, and server configuration related users.

**Diagram 4.2 User Services APIs and node\_modules contain dependencies**



- Here, the API folder contains APIs and authentication logic for users.
- server.js file contains server configuration and defines a port for User Services.
- package.json file contains a list of dependencies, project description, server deployment command, etc.

**Diagram 4.3** pacakge.json file

```
User Services > {} package.json > ...
1  {
2    "name": "user-services",
3    "version": "1.0.0",
4    "description": "It is user microserice",
5    "main": "User.js",
6    "dependencies": {
7      "bcrypt": "^4.0.1",
8      "body-parser": "^1.19.0",
9      "express": "^4.17.1",
10     "jsonwebtoken": "^8.5.1",
11     "mongoose": "^5.9.15",
12     "morgan": "^1.10.0"
13   },
14   "devDependencies": {
15     "nodemon": "^2.0.4"
16   },
17   "scripts": {
18     "test": "echo \"Error: no test specified\" && exit 1",
19     "start": "nodemon server.js"
20   },
21   "keywords": [
22     "node",
23     "microservice"
24   ],
25   "author": "Vishal Parekh",
26   "license": "ISC"
27 }
28
```



- app.js file is the main module which creates APIs, connects database, handles various errors, parse request body.
- app.js works as a middleware and forward all the request to /app-admin.
- Any request to this service looks like **'http://localhost:port/app-admin/(specifird operation)'**
- app-admin is the main router of this service which contain all references to functionalities.

**Diagram 4.4 app-admin.js files small snapshot which contains reference to SignUp and Login.**

```
User Services > api > routes > JS app-admin.js > ...
1  const express = require('express');
2  const route = express.Router();
3  const user = require('../controllers/user');
4  const organization = require('../controllers/organization');
5  const group = require('../controllers/group');
6  const role = require('../controllers/role');
7  const right = require('../controllers/rights');
8  const checkAuth = require('../Authentication/check_auth');
9
10 //User SignUP
11 route.post('/signup',user.addUser);
12
13 //Get all User Info
14 route.get('/userList',checkAuth,user.userData);
15
16 //User LogIn
17 route.post('/login',user.logUser);
18
19 module.exports = route;
```

- As we can see in Diagram 4.4 that all the references fetch from /controllers folder. This is the folder which consists of all the modules with business logic for users operation like signup, login, etc.
- In diagram 4.4 we can see one more reference to /Authentication folder. This is the folder that contains logic for user validation.
- When a user performs successful login operation then system generates token using the “**jsonwebtoken**” library and, adds it to the response.
- Every time user sends a request to this service, the browser automatically adds this token to request body, and this token verified by check\_auth.js module.
- This token is session-specific, it means over new session users need to get new token via login operation.
- In diagram 4.5 we can see another module that is imported from /models. This model folder contains the schema for the database.
- The following diagram 4.6 depicts the users-model.
- Diagram 4.6: the email field which, contains one property “match” is a regex of email. It checks for an appropriate email format.
- Till now, I describe the overall structure of user microservice.

**Diagram 4.5 Small screenshot of user.js**

```

User Services > api > controllers > JS user.js > ...
1  const User = require('../models/users');
2  const bcrypt = require('bcrypt');
3  const jwt = require('jsonwebtoken');
4  //This function create new use and save it to the database
5  exports.addUser = (req, res, next) => {
6      User.find({email : req.body.email}).exec().then(user =>{
7          if(user.length >=1){
8              return res.status(409).json({
9                  message : "User already exist"
10             });
11         }
12         else{
13             bcrypt.hash(req.body.password,10,(err,hash)=>{
14                 if(err){
15                     return res.status(500).json({
16                         error : err
17                     });
18                 }

```

**Diagram 4.6 users.js file which contains database schema for users table in MongoDB**

```

User Services > api > models > JS users.js > ...
1  const mongoose = require('mongoose');
2  const userSchema = mongoose.Schema({
3      user_id : String,
4      firstName : {type:String, required : 'Please enter your first name'},
5      lastName : {type: String},
6      email : {type:String, required: 'Please enter your email',
7      match : /[a-z0-9!#$%&'*/+=?^_`{|}~]+(?:\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*@(?
8      password : {type:String, required: 'Please enter your password.'},
9      organization : {type:String, required: 'Please enter your organization.'},
10     orgId : {type: String},
11     role : {type: Array},
12     join_date : {type: Date, default: Date.now},
13     subscriptionId : {type : String},
14     rights : {type : Array}
15 });
16
17 //var user = mongoose.model('User' , userSchema);
18 module.exports = mongoose.model('User' , userSchema);

```

- The following functionalities offer by user microservices:

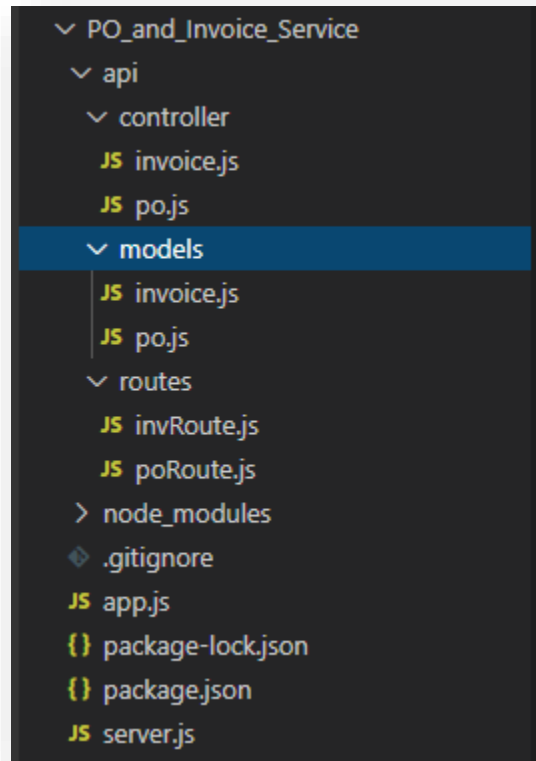
Note: All these operations performed by organizations admin.

1. Organization Registration.
2. User Registration.
3. User Log In (This operation performs by user itself and admin also can).
4. Group Creation.
5. New Rights add to system.
6. New Role generation.
7. Role Assignment to users.
8. Rights Assignments to specific role.
9. Display user data to users-dashboard.

**4.1.2. PO\_and\_Invoice\_Service:** This microservice contains all features related to Invoice and Purchase Order.

- The project structure of this service is same as user service.
- The server configuration of this service is different than user service.
- Both services running on different ports.
- But both the services share same Database server.
- Here, database server is “mongodb://localhost:27017/POIDB”.
- In both services, there is one file .gitignore file. This file is a text file that tells Git which files or folders to ignore in a project.

**Diagram 4.7 Project Structure of Invoice and PO service**



- There is one more file present in both services package-lock.json. It is automatically generated file by npm where npm modifies either node\_modules tree, or package.json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.
- The following functionalities offer by user microservices:

Note: All the operation performed by users according to rights.

1. Invoice Generation.
2. Purchase Order Generation.
3. Retrieve all Invoice of organization.

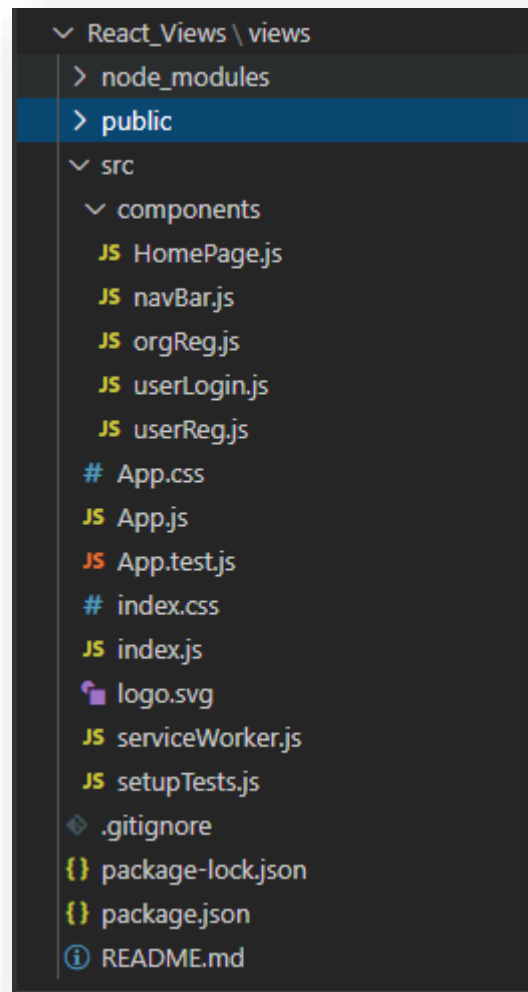
4. Find all Invoice of particular P.O.
5. Update Status of Invoice in approval process.
6. Retrieve all P.O associated with specific group.
7. Update P.O (Only for Super User, i.e. CFO)

Note: Invoice verification process comes under Update status.

#### **4.2. UI Implementation**

- User Interface developed using React. React is an open-source JavaScript library for building user interface.
- It is maintained by Facebook.
- React can be used as a base in the development of single-page application.
- Here, single page application is a web application or website that interacts with web browser by dynamically rewriting the current web page with new data from web server, instead of the default method of the browser loading entire new page.
- Here, I used React App to develop UI, it sets up development environment so that user can use latest JavaScript features.
- Diagram 4.8 describes project structure of React App.
- Here, in this project structure everything is readily available for development including dependences.
- We have to edit only App.js file to add user defined components to render.

**Diagram 4.8 Front-End project set up using React App.**



- To add a user defined component, the user can use app.js file or define it in the src/component folder and add explicitly in app.js. We use a later method.
- Here, we have five components,
  1. HomePage.js
  2. NavBar.js
  3. orgReg.js

4. userLogin.js

5. userReg.js

- Since it is a single page application, we need to have a router to toggle between different components.
- For that, we import 'react-router-dom' in the App.js file and use it as an Html tag to create a router.
- Diagram 4.9 describe that router in details.

**Diagram 4.9 App.js describing router.**

```

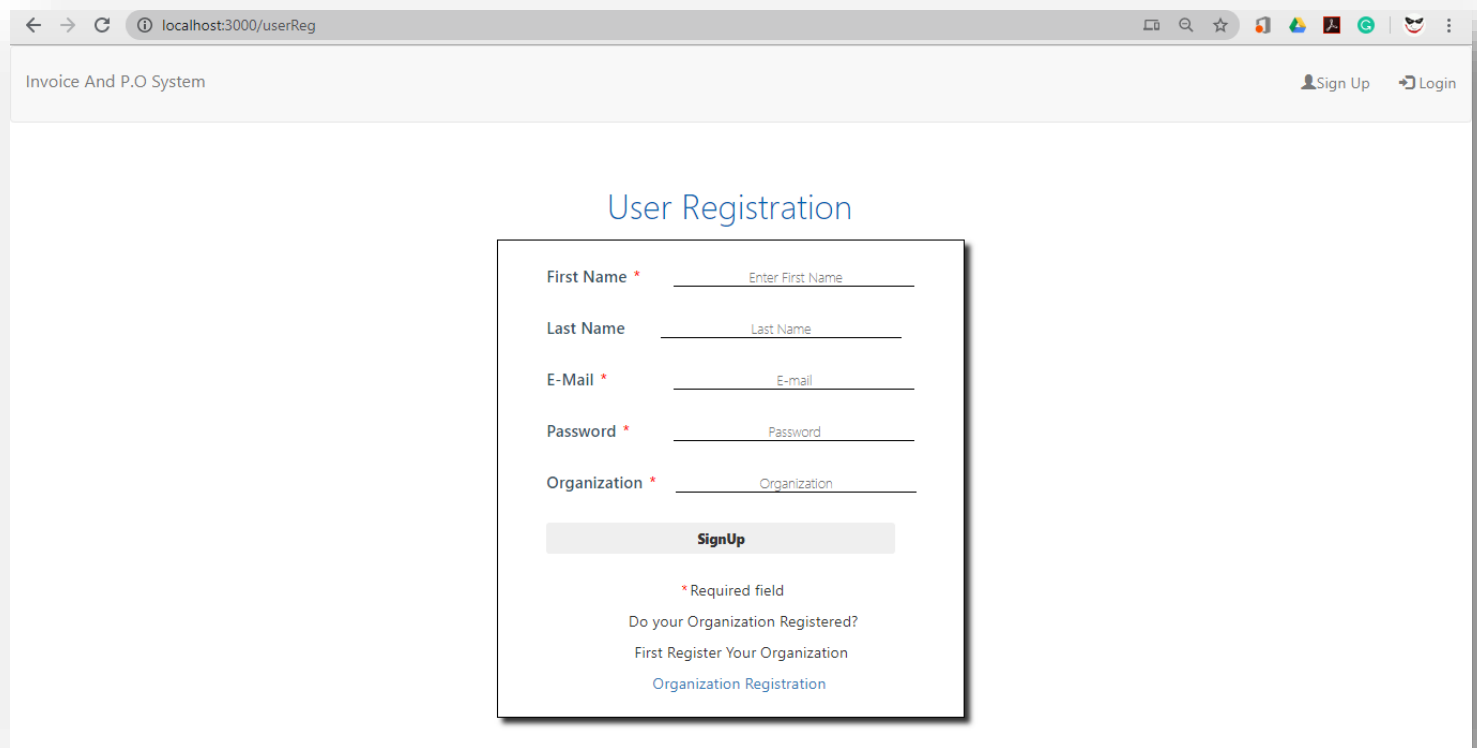
React_Views > views > src > JS App.js > App
1  import React from 'react';
2  import './App.css';
3  import { BrowserRouter as Router } from 'react-router-dom';
4  import Route from 'react-router-dom/Route';
5  import User from '../../views/src/components/userReg';
6  import Home from './components/HomePage';
7  import UsrLgn from './components/userLogin';
8  import OrgReg from './components/orgReg';
9  function App() {
10   return (
11     <div className="App">
12       <Router>
13         <Route path="/" exact component={Home}/>
14         <Route path="/userReg" exact strict component={User}/>
15         <Route path="/userLogin" exact strict component={UsrLgn}/>
16         <Route path="/orgReg" exact strict component={OrgReg}/>
17       </Router>
18     </div>
19   );
20 }
21
22 export default App;

```

- Here, the router tag provides different URL for our website, and for specific path particular component render to index.html.



- Diagram 4.10 describe User Registration component



The screenshot shows a web browser window with the address bar at `localhost:3000/userReg`. The page title is "Invoice And P.O System". In the top right corner, there are links for "Sign Up" and "Login". The main content area features a "User Registration" form. The form includes five required fields: "First Name", "Last Name", "E-Mail", "Password", and "Organization", each with a placeholder text. Below these fields is a "SignUp" button. Under the button, there is a note "\* Required field", a question "Do your Organization Registered?", and a link "First Register Your Organization" which points to "Organization Registration".

**Diagram 4.10 User Registration Form (URL: localhost:3000/userReg)**

- The User is a part of an organization, so the organization's name is a required field.
- So, the organization registration is required to use this product.
- When the user presses the SignUp button, form data send to the server in JSON format and stored to the database, if there is any an error alert message display else user redirect to the log in page.
- Here, I used the Axios library to send a post request to the server. The browser sends a request to the `'http://localhost:3001/app-admin/signup'`.

- In diagram 4.10, a single page consists of two components. The first component is the Nav component which, explicitly added to userReg.js from navBar.js. The second component is the User component added to App.js.
- The user login page and organization page is as same as the user registration page. The difference is only in form elements and in target API.
- The target API for the user login page is '**http://localhost:3001/app-admin/login**' and, the organization registration page sends the request to the following link, '**http://localhost:3001/app-admin/orgReg**'.

## Chapter 5 Testing

- Testing is a crucial phase of the software development life cycle ensure that the quality goals have been met.
- There are various testing methods available for software testing like unit testing, automation techniques, etc.
- This system is tested in unit testing. Here, each and every module tested separately against various test cases.
- In the end, integration testing applies to UI and server.

### 5.1 Unit Testing

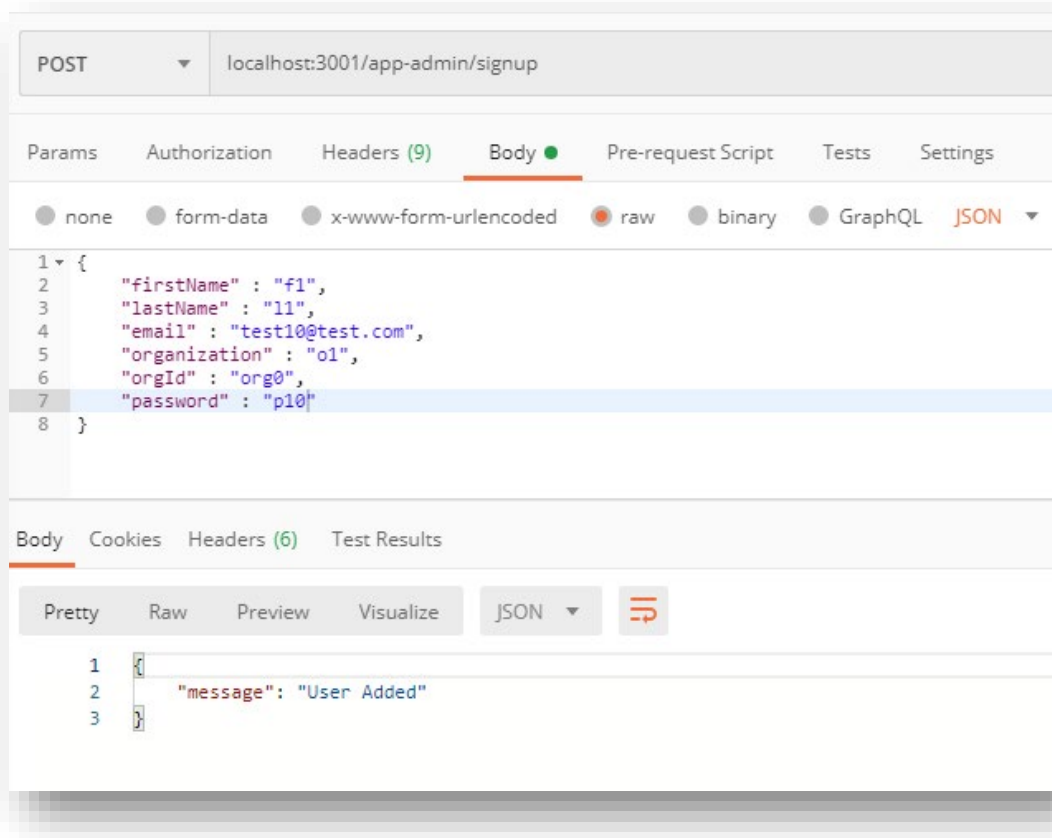
- In the unit testing of this system, I have to check all the APIs defined in both the microservices.
- I used the postman tool for testing all APIs. Postman is an API client that makes it easy for developers to test APIs. It is done by allowing users to create and save HTTP requests, as well as responses.

#### 5.1.1. User Microservice Testing

- Test case 1: Check 'localhost:3001/app-admin/signUp' API
  - Input Data: New user data with unique email and password.
  - Expected Result: Get Response form server that User Added.
  - Actual Result: Diagram 5.1
- Test case 2: Check 'localhost:3001/app-admin/signUp' API
  - Input Data: Enter user data which already registered.

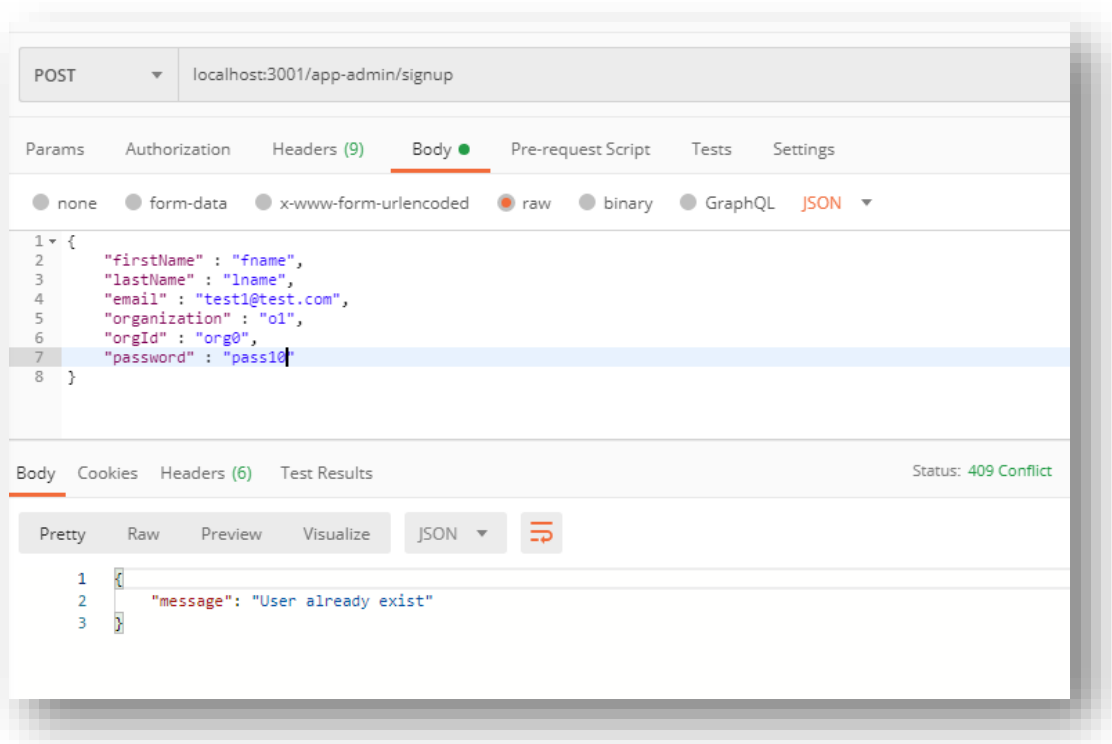
- Expected Result: Get Response from the server that User already exist
- Actual Result: Diagram 5.2

**Diagram 5.1 Result of Test Case 1 in the postman**

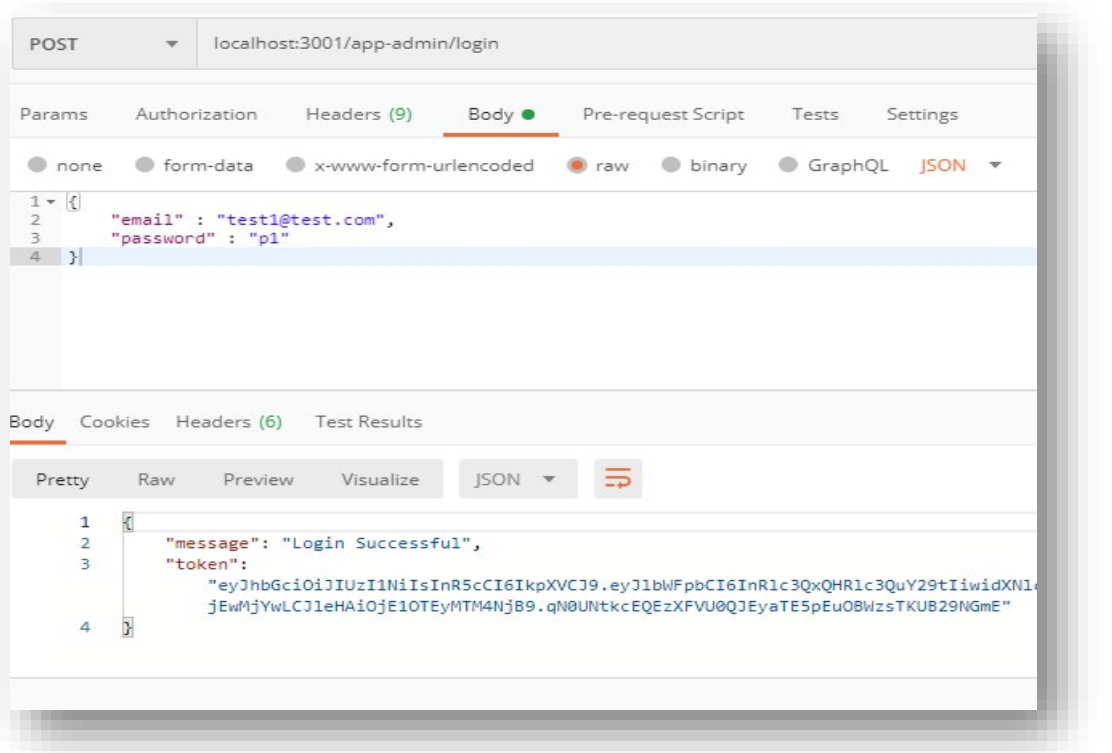


- Test Case 3: Check 'localhost:3001/app-admin/login' API
  - Input Data: Enter email and password of existing user.
  - Expected Output: Successful Login Response.
  - Actual Output: Diagram 5.3 contains a successful login message along with system generated token. This token recognizes the user for that session.

**Diagram 5.2 Result of Test case 2 in Postman showing error: 409 conflict error**

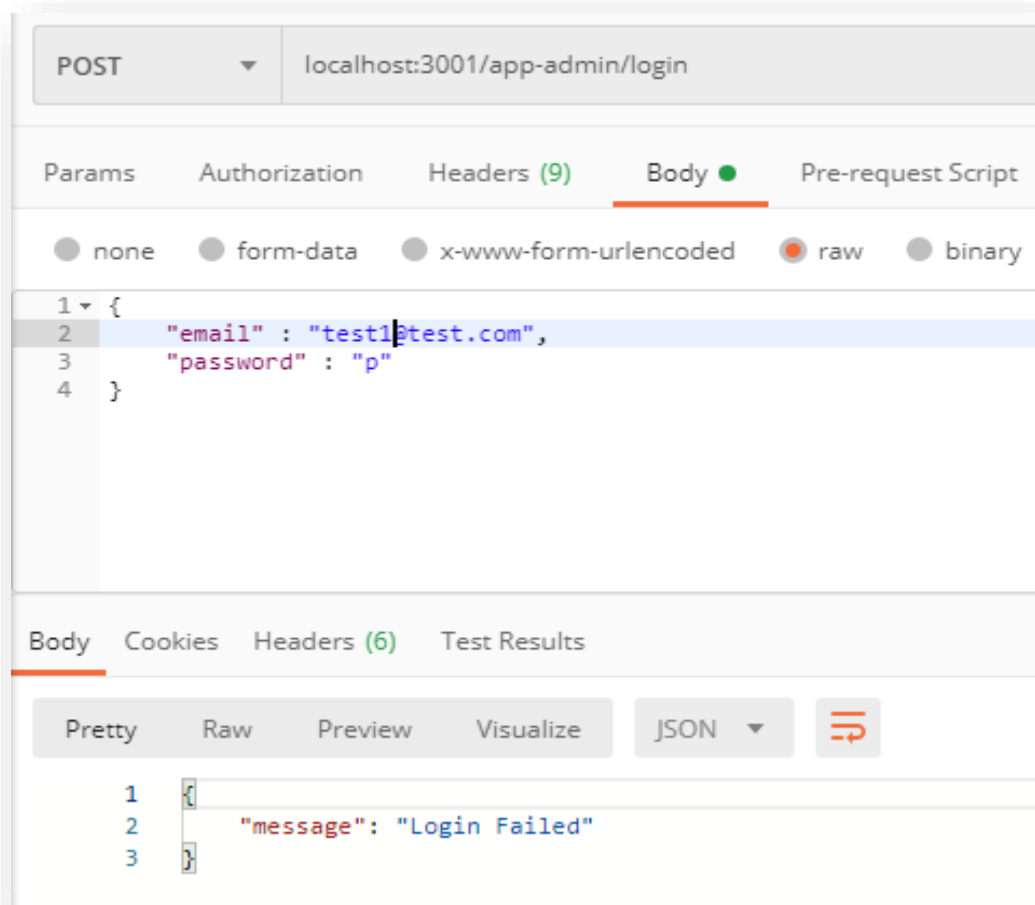


### Diagram 5.3 Result of Test Case 3 in the postman



- Test case 4: Check 'localhost:3001/app-admin/login' API
  - Input data: Enter email and wrong password of existing user.
  - Expected Output: Error message Incorrect Password.
  - Actual Output: Diagram 5.4

**Diagram 5.4 Output of Test Case 4**



- Test case 5: Check 'localhost:3001/app-admin/userList' API
  - It is a GET request to fetch the organization's employee list.
  - Here, user is not logged in.
  - Output: Error message "Authentication Failed", error code: 401 unauthorized.

- Here, request header need token to access this API
- Test case 6: Check 'localhost:3001/app-admin/userList' API
  - Here, user is logged in. Now user have token.
  - Output: Organization's user list.

### 5.1.2. Invoice and PO service Testing:

- Test case 1: Check 'localhost:3002/invRoute/addInv' API
  - Input data: Enter required data in json format in postman.
  - Output: Here, I get a response message that Invoice created, but we have to manually check the database to ensure response message.
  - Database Query: This is a MongoDB query.

Switch to our database: use POIDB.

Find invoice query: db.invoices.find()
  - It shows all invoices, so this API working properly.
- Test case 2: Check 'localhost:3002/invRoute/updateState' API.
  - This API is a part of validation process.
  - Input Data: There are three required field inv\_id, status, status\_chndBy. Enter data in these three fields and send a request.
  - Output: A response message 'Invoice Status Updated'.
- Test case 3: Check 'localhost:3002/invRoute/addPo' API.
  - Input Data: Send post request body in json format to the server.

- Request body contains all required field.
- Output: Server send response message 'P.O created'.
- So, far I described all-important test cases. There are more than 20 APIs.

## 5.2 Integrated Testing:

- Integrated testing takes place with UI and Server. Here, testing tool is Google Chrome browser.
- It is a cross-platform testing, where UI and APIs running on two different terminals.
- User Interface deployed by a react app on localhost:3000 and APIs of User Service running on localhost:3001.
- This testing is faster than unit testing because we have proper UI.
- Tester has a user registration form, fill that form regardless of data format.
- React component associated with that form convert all input data into JSON format and send a request to target API.
- In unit testing, the tester has to create the whole request body with a header, in this testing, it is done by the browser.
- On a successful user registration, the user will be redirected to another component. But tester has to ensure that the data stored in the database.



- There is no particular test case in integrated testing, because all APIs working properly in unit testing. We have to ensure UI and server communication is reliable.
- Tester has to make sure that all the requests from localhost:3000 received by localhost:3001 and localhost:3002.
- I checked every request form localhost:3000 was reached to target APIs. And also, receive a response from the server.

## Chapter 6 Summary

- The Invoice and P.O system is efficient, reliable, and cost-efficient for small scale organizations. It connects traditional financial operations of an organization to the digital world.
- This project builds using modern tools and technologies. All the tools and technologies are used in this project is open-source. In future any software problem arises due to technology will be solved using the open-source community.
- This project helps me to learn new technologies and development tools.
- I learned the microservice concept as new development techniques and its principles. And implement these concepts in this project.
- I would like to thank Dr. Rutvij Jhaveri. He is my academic mentor and help me in this project as an advisor and also, provide vision toward the software development life cycle.

**Project Group Personal Details**

1. Name of the Student: Vishalkumar Parekh  
Photograph

Permanent Address: 15, Suryarath Society, Naroda, Ahmedabad-382330

Email: vishalparekh130@gmail.com

Mobile no: 9904353730