Technical Report
Hotel Feedback Management System

Ethan Boyar, Bhavpreet Singh, Minglok Lin, and Jaan Malik

1285774, 1291491, 1278749, 1279023

Software Failures

CSCI-380

Introduction to Software Engineering

Fall 2023

Maherukh Akhtar

12/8/23

https://github.com/fangedShadow/SE_Project

Table of Contents

Abstract

The Node.js-based Hotel Feedback Portal is a feature-rich web application employing Express.js and MongoDB, and has user authentication, ML complaint management, and reporting. Users, both regular and managerial, can register, log in, and submit complaints categorized into specific types. A distinctive feature is the ability for managers to generate detailed PDF reports summarizing complaints based on types and date ranges. The modular code structure enhances maintainability, while error handling ensures a seamless user experience. Notably, the system has incorporated machine learning for automated complaint categorization, providing a novel label to user-submitted complaints.

Keywords: Express.js, Machine Learning, MongoDB, Complaint management, and User Authentication

I) Introduction

The Hotel Feedback Portal is a comprehensive web application catering to the needs of guests, users, and managers. Guests can submit complaints anonymously, providing valuable feedback without the need for user registration. Users, whether named or anonymous, have the flexibility to log in, and submit complaints.. Managers enjoy privileged access, efficiently managing complaints through sorting, a dedicated dashboard, and generating insightful reports based on complaint types and dates. The tech stack includes Node.js, Express, MongoDB, Passport, EJS, PDFKit, Multer, Cloudinary, and various middleware components for enhanced functionality and user experience.

The incorporation of machine learning, powered by Python, Flask, spaCy, and Classy Classification, sets the project apart. This external ML server handles text classification using a pre-trained model (PMMLM v2) to automatically categorize and label user-submitted complaints. The model's ability to interpret paraphrases and its multilingual capabilities ensure robust complaint classification. This innovative approach streamlines the complaint management process, enhancing the overall efficiency and user satisfaction of the Hotel Feedback Portal.

Existing Systems

General feedback software already exists, but there is very little dedicated software for hotels. Just about any business requires specific feedback management, and hotels are no different. Nonetheless, some of the benefits of mainstream feedback management systems are:

Savio streamlines the feedback process by centralizing product feedback from various sources, providing businesses with a unified view. This versatile add-on seamlessly integrates with platforms like email, Google, Zendesk, and more, facilitating a comprehensive and accessible feedback management solution.

Parative Voc takes feedback management to the next level by combining customer feedback with CRM and contract data. By doing so, it identifies and prioritizes critical customer needs based on factors such as pervasiveness, affected customer segments, revenue impact (both existing and potential), urgency to the customer, and renewal risk.

Proposed System

We want to design and develop a simple, straightforward and robust feedback management system. Users should be able to easily write and submit feedback into a portal. The feedback should be sorted into complaints and compliments, and then categorized using sentiment and category analysis, which ML will handle. Once the feedback has been correctly sorted and stored, it should be easily accessible by employees and management. However, user access control needs to be carefully implemented to ensure feedback is only visible to those with adequate credentials.

Advantage of the proposed system

The proposed system stands out in the market by addressing the unique needs of the hospitality industry. While general feedback software exists, there is a notable gap in dedicated solutions for hotels. Unlike mainstream feedback management systems, our specialized platform focuses exclusively on the intricacies of hotel feedback. This targeted approach ensures a centralized space where customers can conveniently submit feedback, and hotels can efficiently manage, analyze, and prioritize these inputs. By tailoring our solution to the specific requirements of the hotel industry, we provide a comprehensive and dedicated tool that goes beyond generic feedback systems. This specialization enhances the user experience for both customers and hotel management, making our platform a valuable and essential asset for the hospitality sector.

Software Engineering Model

The selection of the Feature-Driven Agile (FDA) model for the development of the Node.js-based Hotel Feedback Portal was a strategic decision, driven by the project's unique requirements and a shift from the initially proposed Waterfall model. Feature-Driven Agile seamlessly blends the customer-centric approach of Agile development with the structured feature delivery system of Feature-Driven Development (FDD). The Feature-Driven Agile model's emphasis on tangible feature delivery aligns perfectly with our goal of creating a specialized and responsive feedback management system tailored specifically for the hotel industry, ensuring the end product remains finely tuned to the unique needs of our users.

Purpose

The purpose of the Hotel Feedback Portal is to provide a specialized and efficient feedback management system for the hospitality industry, offering guests a convenient platform to submit feedback and complaints while empowering hotel management with tools for streamlined complaint handling, analysis, and reporting.

Project Objective/Goals and Scope

The project aims to develop a feature-rich Hotel Feedback Portal, utilizing Node.js, MongoDB, and Machine Learning for automated complaint categorization. The goals include providing guests with an anonymous feedback option, enabling users to submit categorized complaints, and offering managers tools for efficient complaint management and insightful reporting. The scope encompasses user authentication, modular code structure, and integration of external ML servers, ensuring a comprehensive solution for the hospitality sector. (we go in detail about these in sections 3 and 4)

Technologies and Tools

All the Tools and technologies are discussed in detail in sections 3 & 4

- ➔ Frontend: HTML, CSS, Javascript
- ➔ Backend: Javascript, nodeJS, Express
- ➔ ML: Flask, spaCy, Classy Classification, PMMLM v2
- ➔ DB: MongoDB
- ➔ Auth: Passport
- ➔ Deployment: Render

Users

Guests: can anonymously submit complaints, ensuring feedback without identity disclosure.

Users: have the flexibility to submit named or anonymous complaints via login/sign-up, with potential future features.

Managers: have privileged users, access and manage complaints efficiently through sorting, a dedicated dashboard, and insightful reporting.

All the Privilege levels are discussed in detail in section 4.

Motivation

The motivation behind us creating this feedback management system is making sure customers know they are being heard, and are satisfied throughout their stay at a hotel. A hotel can always learn from customer feedback, even if they don't think they can. We don't want good customer feedback to go unnoticed, and we want to make sure whatever the feedback mentions is corrected. We all liked the idea of making a feedback management system, since we could see it actually helping a small hotel business out.

Timeline

This is the Timeline before the project Started, and we will compare in section 5(Earned Value Analysis)

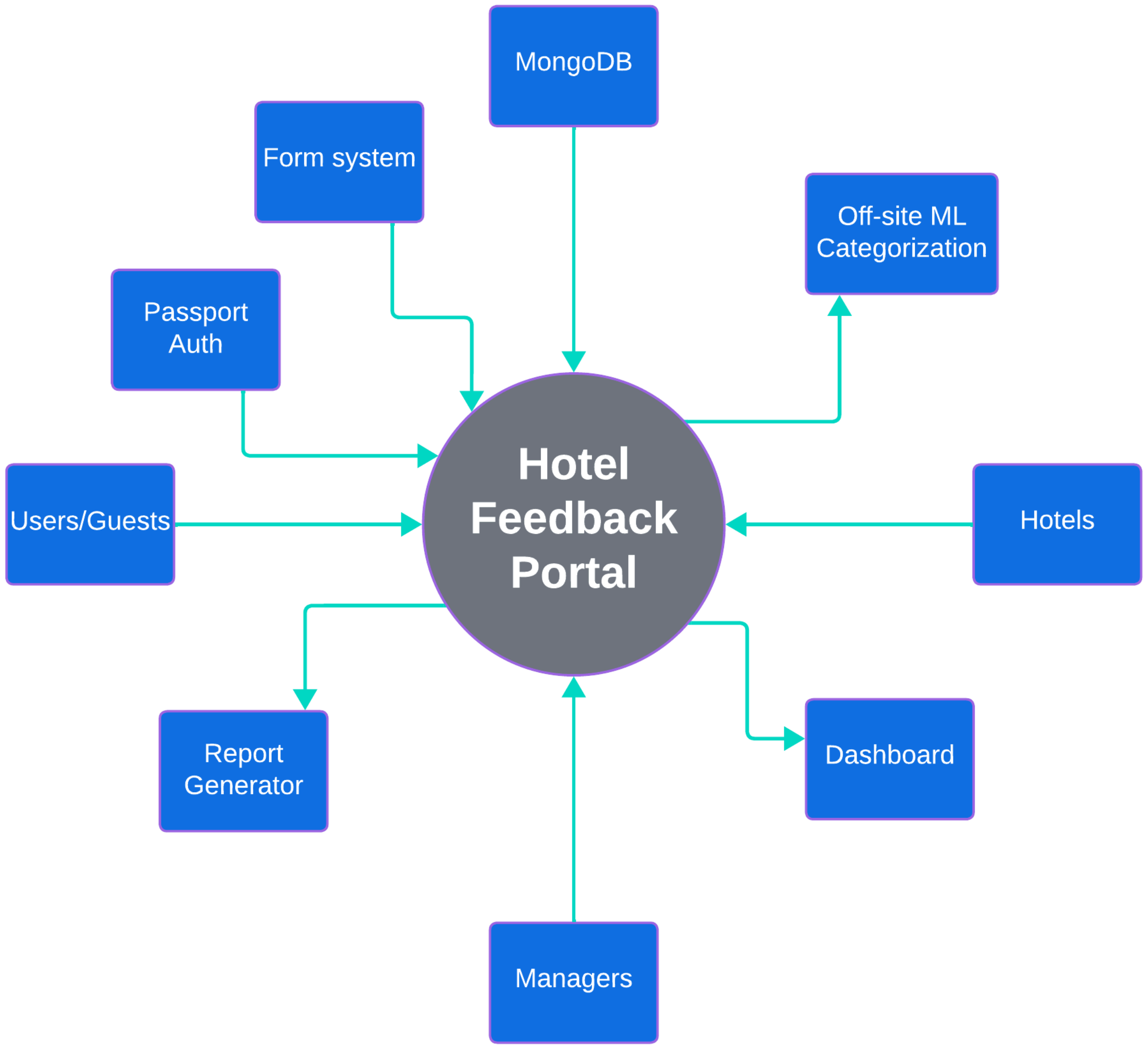
Nov 1: - submit proposals
Nov 9: - all the project models and designs
Nov 18: - basic working website
Nov 25: - integration of the Ml into the code
Nov 30: - completion of the whole App
Dec 5: deployment and testing
Dec 8: technical report and presentation submission

II) Analysis of the Project

Activity List

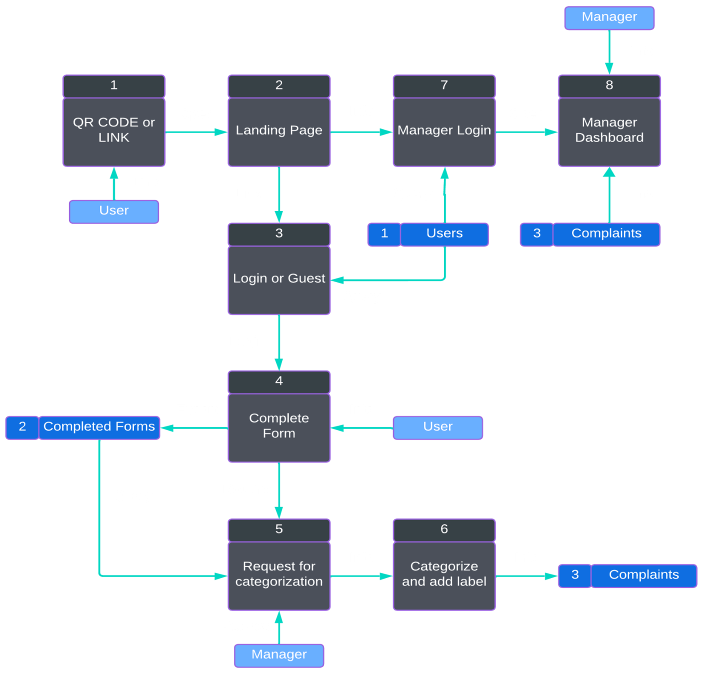
- ➔ **i) Complaint Submission:**Create a form for users to submit complaints, allowing for anonymous submissions.
- ➔ **ii) User Authentication:** Develop user registration and login functionality for user and Management. Implement secure authentication mechanisms using Passport middleware.
- ➔ **iii) Database Interaction:** Establish connections to MongoDB for efficient data storage and retrieval. Implement Mongoose for streamlined interactions with MongoDB.
- ➔ **iv) Frontend Development:** Develop responsive and visually appealing frontend using Bootstrap. Use EJS for simplified integration of dynamic content into views.
- ➔ **v) Machine Learning Integration:** Set up a separate server for machine learning tasks using Flask. Implement spaCy and Classy Classification for automated complaint categorization.
- ➔ **vi) Complaint Management Dashboard:** Design a dedicated dashboard for managers to view and sort all submitted complaints. Include different tabs for sorting complaints based on various criteria.
- ➔ **vii) Reporting System:** Integrate a reporting feature for managers to generate detailed reports. Apply filters, such as complaint type and date, to refine the generated reports. Use PDFKit to generate dynamic PDF reports summarizing complaints.
- ➔ **viii) Deployment:** Deploy the application using the Render platform for hosting.

Context Diagram



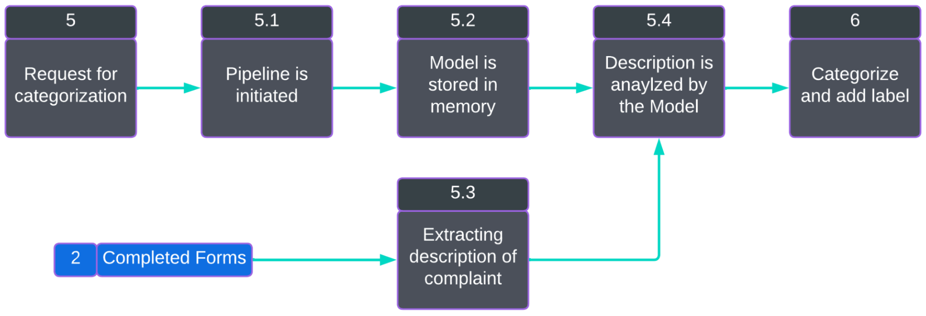
Data Flow Diagrams

i) Main DFD



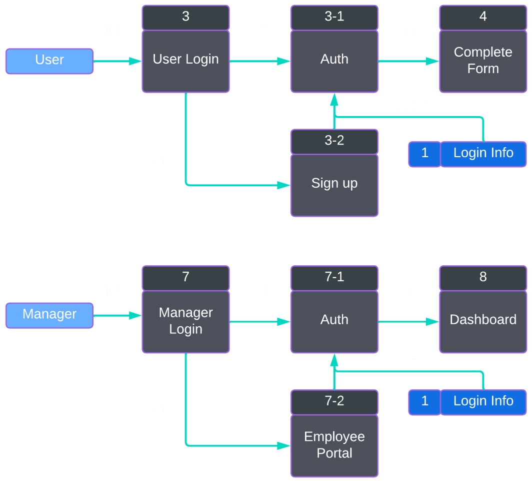
The process begins with a user accessing the website through a link or QR code, directing them to the landing page. From there, they have two main options: submitting a complaint or proceeding to the management login. If the user opts for the complaint submission, they can either submit it as a guest without authentication or log in as a registered user, invoking the authentication child diagram. Once the complaint form is filled, it is submitted, and based on the manager's request, the complaint undergoes categorization. This categorized data then flows to the Machine Learning (ML) child diagram, where the ML system applies labels to the complaints. The labeled complaints are stored for future reference. On the other hand, if the user chooses the management login, they undergo authentication through the child diagram, gaining access to a dashboard with sorted complaints. Managers can further request reports based on the categorized complaints, providing a comprehensive and streamlined feedback management system.

ii) Ml DFD - Child Diagram


5
Request for categorization
5.1
Pipeline is initiated
5.2
Model is stored in memory
5.4
Description is anaylzed by the Model
6
Categorize and add label
5.3
Extracting description of complaint
2
Completed Forms

In the ML Child Diagram, you can observe the synchronous process that occurs once a request for categorization is made. Completed forms are ingested from MongoDB, and at the same time, the pipeline is initiated, and the Model loaded into its memory. Both of these are handled as subprocesses on a Flask web app, and proceed to

iii) Authentication DFD - Child Diagram

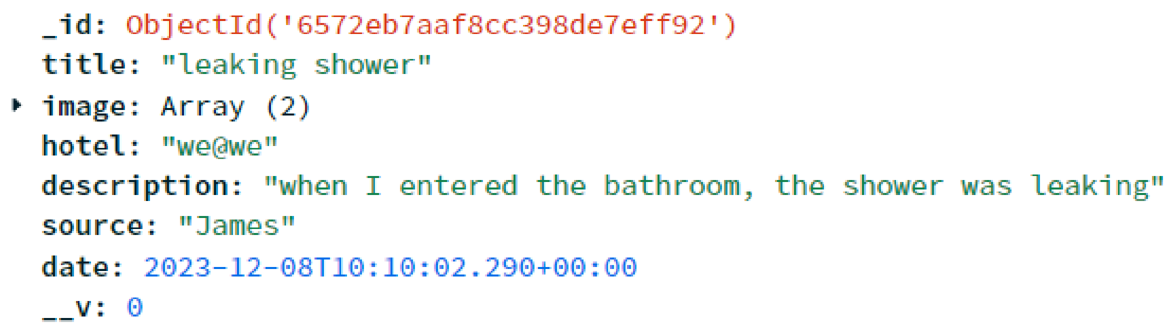


In the Authentication Child Diagram, the process diverges based on the user's intent. If the user wishes to submit a complaint as a registered user, they navigate to the login page. If they already possess login credentials, they proceed to log in. If not, they have the option to redirect to the registration page, where they can register and then log in. On the other hand, if the user aims to access the management dashboard, they are directed to the login page. If they possess login credentials, they can log in; otherwise, they need to be registered. To register, they contact the system administrator(us) for a registration form link. Upon receiving the link, they can register themselves or their hotel if it does not already exist, and then proceed with the login.

III) Database Design

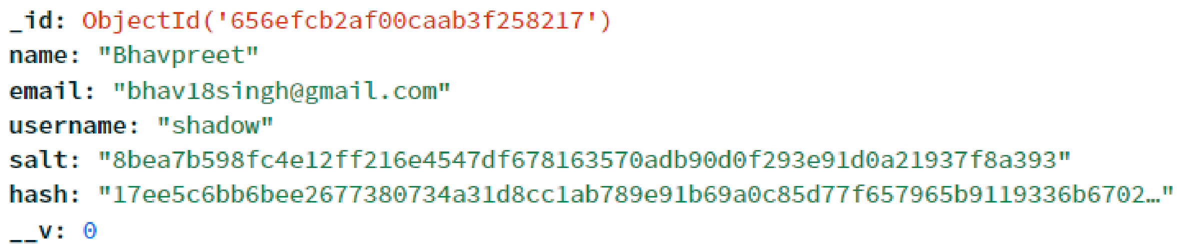
Principle

We developed a web-based application using MongoDB to manage user complaints. Users can submit **complaints**, including details such as title, hotel, description, image, date, and source (linked to the user if logged in; otherwise, left empty).



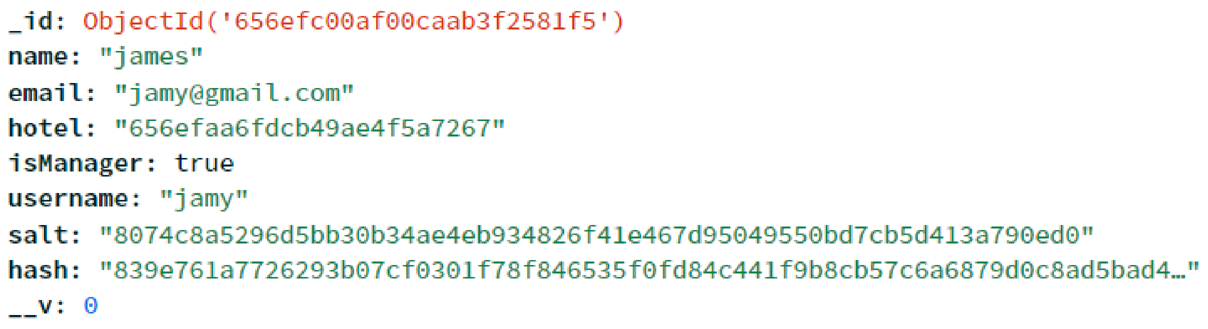
```
  _id: ObjectId('6572eb7aaf8cc398de7eff92')
  title: "leaking shower"
▸ image: Array (2)
  hotel: "we@we"
  description: "when I entered the bathroom, the shower was leaking"
  source: "James"
  date: 2023-12-08T10:10:02.290+00:00
  __v: 0
```

User information is collected, including name, password, email, and username.



```
_id: ObjectId('656efcb2af00caab3f258217')
name: "Bhavpreet"
email: "bhav18singh@gmail.com"
username: "shadow"
salt: "8bea7b598fc4e12ff216e4547df678163570adb90d0f293e91d0a21937f8a393"
hash: "17ee5c6bb6bee2677380734a31d8cc1ab789e91b69a0c85d77f657965b9119336b6702…"
__v: 0
```

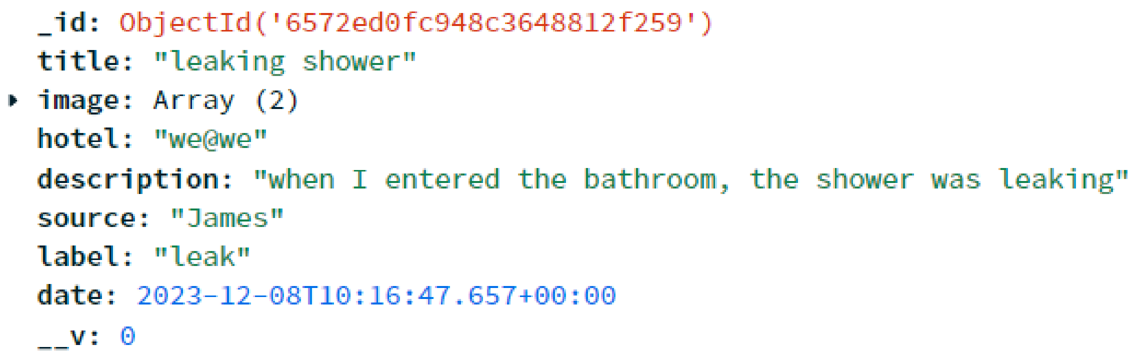
The complaints are stored and can be accessed by a **manager**, who has details like name, email, hotel, username, and password.



```
_id: ObjectId('656efc00af00caab3f2581f5')
name: "james"
email: "jamy@gmail.com"
hotel: "656efaa6fdcb49ae4f5a7267"
isManager: true
username: "jamy"
salt: "8074c8a5296d5bb30b34ae4eb934826f41e467d95049550bd7cb5d413a790ed0"
hash: "839e761a7726293b07cf0301f78f846535f0fd84c441f9b8cb57c6a6879d0c8ad5bad4…"
__v: 0
```

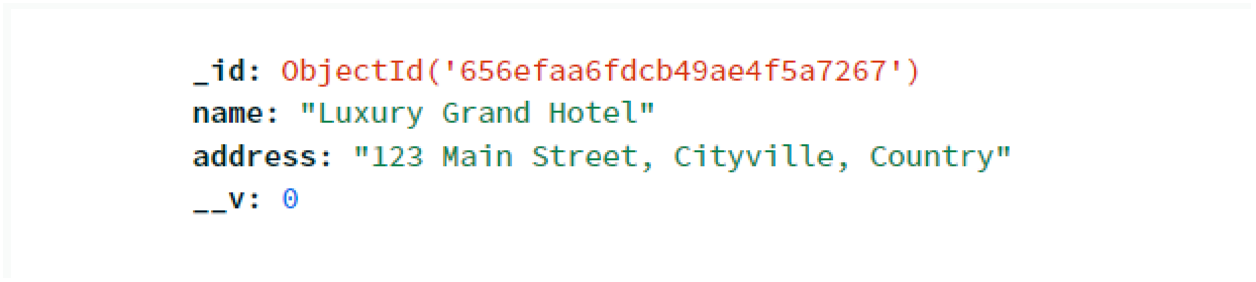
The manager reviews the complaints, and can categorize them by processing them through a Python machine-learning script. This script analyzes the data, assigns a label to each complaint, and stores the sorted complaints in a new table called

sortedComplaint. This table contains the same fields as complaints but with a newly assigned label.



```
  _id: ObjectId('6572ed0fc948c3648812f259')
  title: "leaking shower"
▸ image: Array (2)
  hotel: "we@we"
  description: "when I entered the bathroom, the shower was leaking"
  source: "James"
  label: "leak"
  date: 2023-12-08T10:16:47.657+00:00
  __v: 0
```

The original complaint data is then removed from the initial table. The manager can view and manage the sorted complaints, enhancing the efficiency of the complaint resolution process.
The complaints are connected to **hotels**, and hotels are connected to the Manager

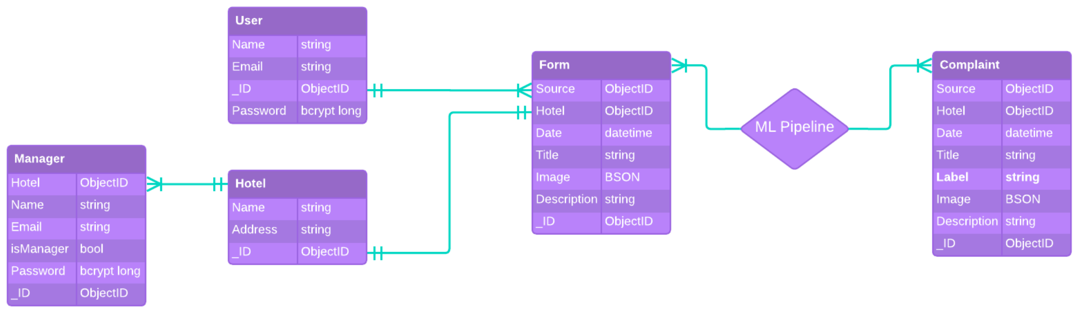


```
_id: ObjectId('656efaa6fdcb49ae4f5a7267')
name: "Luxury Grand Hotel"
address: "123 Main Street, Cityville, Country"
__v: 0
```

We went with MongoDB because it aligned well with the requirements of the project. MongoDB is a NoSQL database that offers flexibility and scalability, making it suitable for handling diverse data types, such as user and complaint information. Its document-oriented structure allows you to store complex data in a format that mirrors your application's objects, simplifying data retrieval and manipulation.

Additionally, MongoDB's ability to handle large volumes of unstructured data, like images, makes it well-suited for a complaint management system that involves multimedia elements. The ease of integration with Python, used for our machine learning script, further streamlines the development process.

Entity Relationships



IV) Implementation

Website (front-end & back-end)

We used **Render** platform for hosting our website application without incurring hosting expenses as the base pack was free. As discussed in the previous section we used **MongoDB** because it is a NoSQL database that offers flexibility in handling diverse data types.

We used **Node.js** which is a JavaScript runtime that allows for server-side scripting. **Express** is a minimal and flexible Node.js web application framework and simplifies routing and middleware usage, streamlining the development of your web application. These technologies enable us to build a fast and scalable server-side application.

Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js, and facilitated the interactions with MongoDB by providing a schema-based solution.

We used **Passport** which is a widely used authentication middleware for Node.js. It provided us a straightforward way to implement authentication strategies, enhancing the security of your application by ensuring that only authorized users can access certain features.

We used **Flash, Express Error, Sessions** which are Express Middleware for handling flash messages, error handling, and session management(cookies). These middleware components enhance the user experience by providing feedback through flash messages, managing errors gracefully, and maintaining user sessions securely.

EJS (Embedded JavaScript) is a simple templating language that lets you generate HTML markup with plain JavaScript. It simplified the integration of dynamic content into your views, making it easier to display, and store data from and in MongoDB.

PDFKit helped in creating dynamic PDF reports or documents. **Multer** simplified the process of handling file uploads, which is essential for managing multiple images in user complaints.

We utilized **Cloudinary**, a cloud service for image and video management, to store complaint images. We saved the Cloudinary image link in the database for efficient retrieval and display.

JavaScript was the main language, but was also used for client-side interactivity as it enhances the user interface with client-side functionalities. **Bootstrap** was used for responsive and visually appealing design and provided a mobile-first design approach, ensuring a seamless experience across different devices.

Machine Learning

Machine Learning was handled separately from the website itself, primarily to alleviate security issues, but also to reduce server load on the host. Everything related to Machine Learning was written in **Python**.

The foundation is an external server which is running **Flask**. Flask is a super lightweight framework that lets us build a web application using Python, meaning we could off-load our ML tasks to a dedicated server.

The implementation of ML starts with **spaCy**, a powerful library specialized in handling Natural Language Processing, aka NLP. There are some key factors that make spaCy extremely useful for our specific application, which is a niche subset of NLP, called Text Classification. First of all, spaCy provides a framework for representing a Machine Learning Model as a Python package, which means we could train *and* export our own model. The same works in reverse, we can *import* a pre-trained model, give it additional training to narrow its scope to our specific problem, and export the new one for our own use.

Second, and perhaps most importantly, spaCy uses a **Pipeline** to handle ML requests. Imagine it as a simple function, data goes into one side of the pipe, and comes out “categorized” from the other. But because it’s a pipeline, data could flow into it indefinitely, synchronously, and in huge batches. This is perfect for our purposes, because we wanted to make the ML as automatic as possible.

Next, we had to pick a Classifier to apply to our pipeline - essentially, data that goes through the pipeline is “transformed”; the transformation is based on a Classifier, and the Classifier compares data against a Model. The Classifier of choice for us is **Classy Classification**, a really powerful library built on top of spaCy. It’s highly specialized for specifically performing text classification and categorization tasks… which is exactly what we want to do.

Finally, we need to pick an actual Model. The idea is that the Classy Classification will compare our data, using a set of rules, against the data within a trained model. By comparing it, similarities between the data can be revealed, and similarities let the Classifier appropriately categorize the data. For this project, the model of choices was:

PMMLM v2 `(paraphrase-multilingual-miniLM-L12-v2)`

Despite the otherwise confusing name, this model is basically perfect for our needs. It’s a pre-trained **sentence-transformer** model, with the goal of mapping sentences to a 3D vector space. A classifier can compare the mapped sentences against our provided data, and use similarities between mapped and unmapped sentences to classify our sentences.

- Paraphrase means that it can interpret sentences that mean roughly the same thing, despite being syntactically different. Such as “the room is cold” and “my room was freezing” both basically being the same thing.
- Multilingual is unused for our purposes.
- miniLM (LM = Language Model) refers to the cardinality of the transformer used, basically this Model is designed for hyper-specialized tasks, and the dimensions of the transformer were shrunk to accommodate that. Since our task is pretty specific, we can save a lot of compute resources by sticking to a miniML.
- L12 is the number of layers within the vector space; it’s mostly irrelevant to us.

The classifier needs to be provided with criteria to use when categorizing the data. In our case, we want to categorize by label, as in determine the subject of the sentence, and label it with that. We created and provided the Model training for 27 unique labels, all of which are related to different possible complaints about a hotel. The classifier is then given the 27 labels to use, and the whole process is complete.

Within Flask, a connection is established to MongoDB and our website. The web app will listen for a POST request from our main website, which tells it to ingest all uncategorized data in the DB, and run it through the pipeline.

Once a POST is received, it will create a new “subprocess”, basically just reserve a thread for use. On that thread, the Model will be loaded into memory, and the pipeline will initiate as a stream. Uncategorized data from the DB will be then run through the pipeline, transformed by our classifier, given a label, and then pushed to a different table containing sorted data in the DB.

Features

- ➔ Login as a guest or user
- ➔ Submit a complaint via form
- ➔ Form is tokenized and stored for later use
    - ◆ Image into BSON format
- ➔ Forms are categorized and labeled by ML
    - ◆ This should occur on by hourly routine to maximize batching
- ➔ Manager can login to their dashboard
- ➔ Manager can view organized complaints for their hotel
    - ◆ Hotel specificity must be one-to-one
    - ◆ Dashboard should also separate by hotel
- ➔ Manager can request all data be categorized right away
    - ◆ Forces the POST request to be sent, resets the timer
- ➔ Manager can generate a report containing stats about their hotel
    - ◆ Works based on date range, and keeps track of manager that requested

Security

Auth handling: Passport is our authentication middleware, used for authenticating everything from login to POST requests. In our case, we implemented it using local-auth, basically meaning we handle checking the password and username locally. The standard for authentication is to check the hash against a generated/bcrypted hash.

Off-loading ML scripts: ML required us to execute Python scripts, including loading an entire Model into memory. Doing this through JS would have exposed the script to a huge number of client-side security problems. Common ones like injection, phishing, and cross-origin breaches could have all deeply compromised our site security. The best way around this is to completely remove the scripts from the website, off-load them to a remote server, and only call for them by POST request as needed.

Privilege Levels

Guest: Guests using the application have the privilege to submit complaints anonymously. This feature ensures that individuals who may not want to disclose their identity can still provide valuable feedback and report issues related to their experiences without the need for user registration.

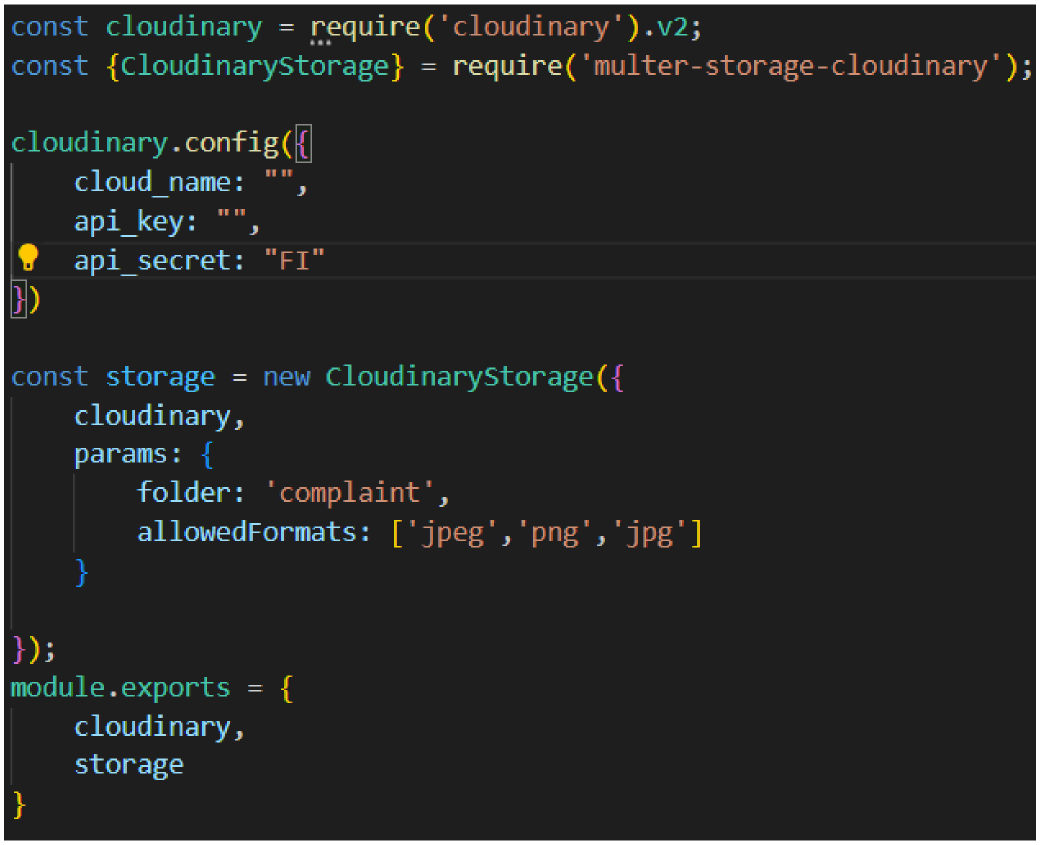
User: Users can choose to submit complaints with their name or anonymously, allowing for flexibility based on individual preferences. Users have to login and sign up. As the application evolves, additional features can be integrated, such as a user dashboard that provides a consolidated view of the complaints they have submitted. While users play a crucial role in submitting complaints, the primary focus is on complaints management for the hotel management.

Manager: Managers, as privileged users, have comprehensive access to the application's functionalities. They can view all submitted complaints, sort them, and utilize a dedicated dashboard for efficient management with different sort complaint tabs. Additionally, managers have the capability to generate reports on the complaints, applying filters such as the date of the complaint and the type of complaint. This feature equips managers with valuable insights, enabling them to address issues promptly and enhance the overall quality of service.

File Structure

The project follows a structured file organization.

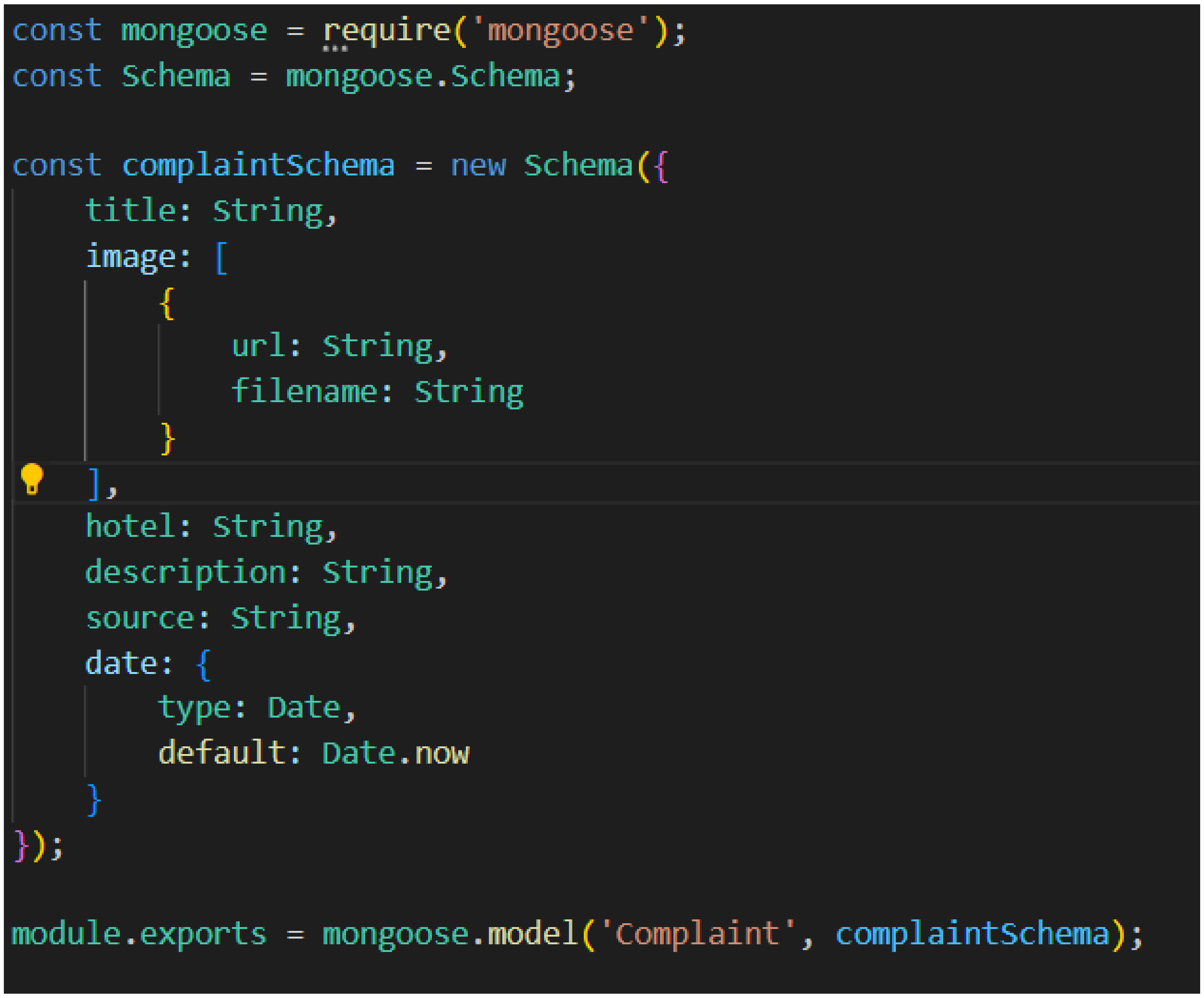
The 'cloudinary' directory contains 'index.js' for Cloudinary configuration.



```
const cloudinary = require('cloudinary').v2;
const {CloudinaryStorage} = require('multer-storage-cloudinary');

cloudinary.config({
    cloud_name: "",
    api_key: "",
    api_secret: "FI"
})

const storage = new CloudinaryStorage({
    cloudinary,
    params: {
        folder: 'complaint',
        allowedFormats: ['jpeg','png','jpg']
    }

});
module.exports = {
    cloudinary,
    storage
}
```

This code configures and exports the Cloudinary setup for image storage in a Node.js application. It uses the 'cloudinary' library to interact with Cloudinary services, providing the cloud name, API key, and API secret for authentication. The 'CloudinaryStorage' module from 'multer-storage-cloudinary' is employed to define storage settings, such as the folder ('complaint') and allowed image formats ('jpeg', 'png', 'jpg'). The 'storage' object is then exported for use in other parts of the application, facilitating seamless integration of Cloudinary for efficient image management within the complaint system.

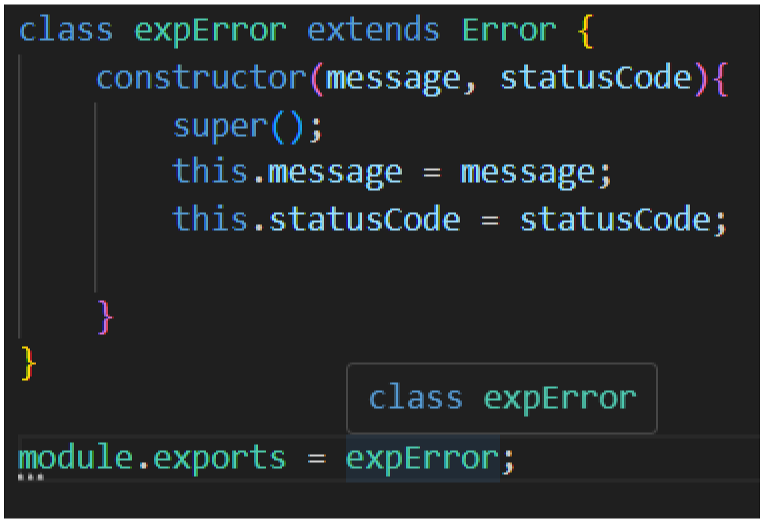
The 'models' directory includes 'complaint.js' and other MongoDB schemas.



```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const complaintSchema = new Schema({
    title: String,
    image: [
        {
            url: String,
            filename: String
        }
    ],
    hotel: String,
    description: String,
    source: String,
    date: {
        type: Date,
        default: Date.now
    }
});

module.exports = mongoose.model('Complaint', complaintSchema);
```

This code defines a MongoDB schema for the 'Complaint' model using Mongoose, a MongoDB object modeling tool for Node.js. The 'complaintSchema' specifies the structure of a complaint document, including fields like 'title,' 'image' (an array containing URL and filename), 'hotel,' 'description,' 'source,' and 'date' (with a default value of the current date). The schema is then exported as a Mongoose model named 'Complaint,' allowing the application to interact with MongoDB and perform CRUD operations on complaint data within the defined schema.

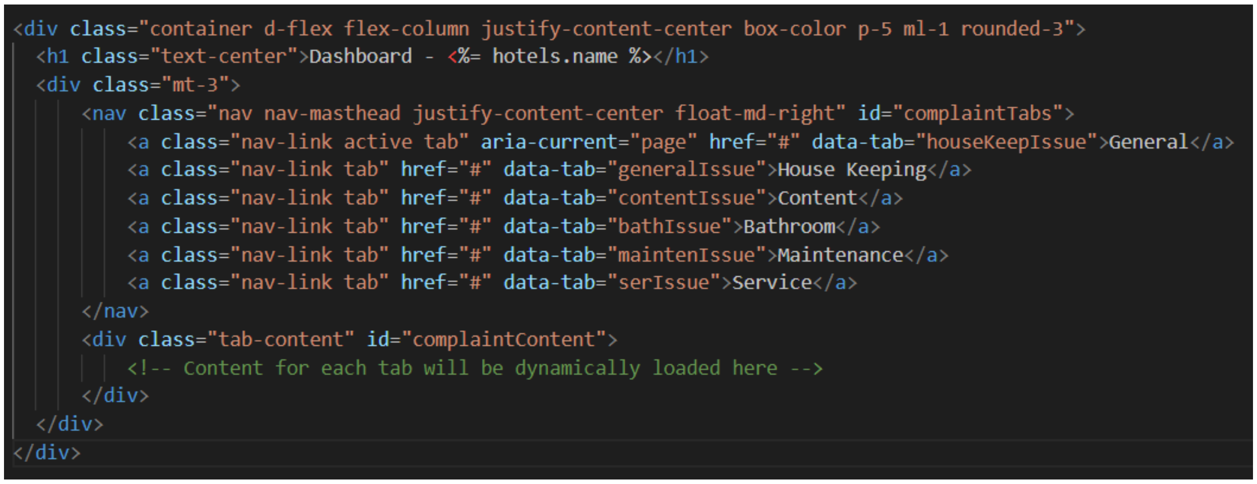
'Utils' contains 'catchasync' for async function error handling and 'expError' for Express error middleware.



```
class expError extends Error {
    constructor(message, statusCode){
        super();
        this.message = message;
        this.statusCode = statusCode;

    }
}
                    class expError
module.exports = expError;
```

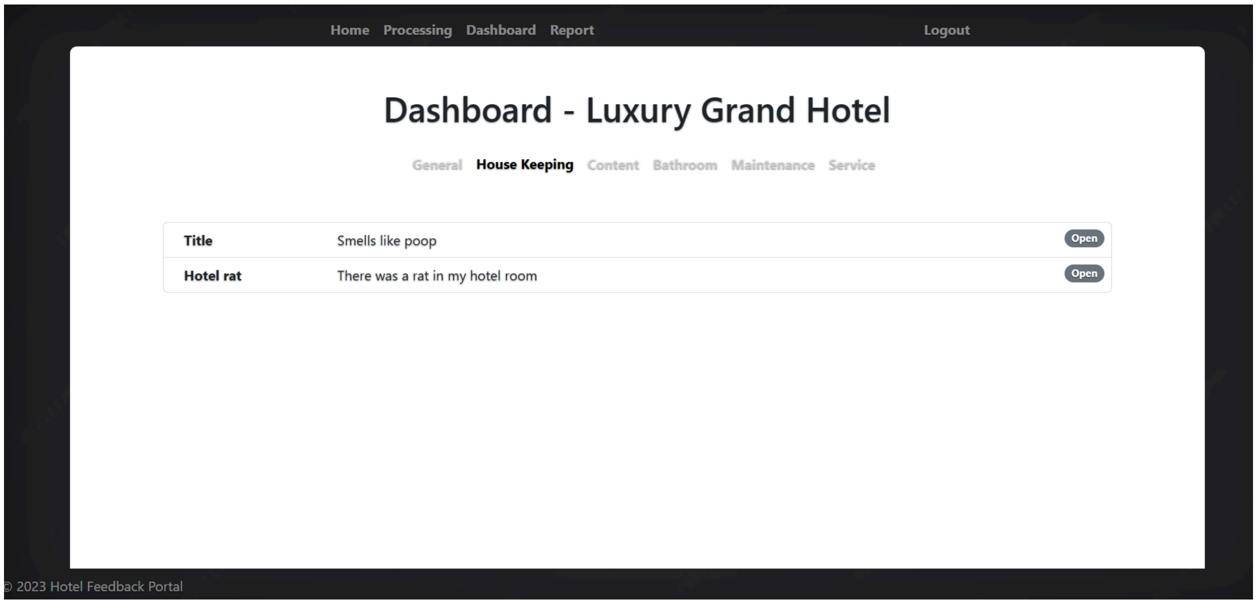
This code defines an 'expError' class that extends the built-in JavaScript 'Error' class. It is designed to create custom error objects for handling Express-related errors in a Node.js application. The constructor takes parameters for the error message and HTTP status code. By encapsulating errors in this custom class, it allows for consistent and organized error handling within the Express middleware.

The **'views'** directory which means everything we see has subdirectories:-
i)'complaint' has the EJS files related to complaints.



```
<div class="container d-flex flex-column justify-content-center box-color p-5 ml-1 rounded-3">
  <h1 class="text-center">Dashboard - <%= hotels.name %></h1>
  <div class="mt-3">
    <nav class="nav nav-masthead justify-content-center float-md-right" id="complaintTabs">
      <a class="nav-link active tab" aria-current="page" href="#" data-tab="houseKeepIssue">General</a>
      <a class="nav-link tab" href="#" data-tab="generalIssue">House Keeping</a>
      <a class="nav-link tab" href="#" data-tab="contentIssue">Content</a>
      <a class="nav-link tab" href="#" data-tab="bathIssue">Bathroom</a>
      <a class="nav-link tab" href="#" data-tab="maintenIssue">Maintenance</a>
      <a class="nav-link tab" href="#" data-tab="serIssue">Service</a>
    </nav>
    <div class="tab-content" id="complaintContent">
      <!-- Content for each tab will be dynamically loaded here -->
    </div>
  </div>
</div>
```

This EJS Snippet shows a dynamic dashboard interface for handling and categorizing user complaints. The HTML structure defines a container with a title, navigation tabs, and a content area. The tabs are associated with different types of complaints, such as housekeeping, content, bathroom, maintenance, and service. The JavaScript script activates the first tab by default, loads content for the default tab, and allows users to switch between tabs dynamically. The event listener on the navigation tabs triggers the loading of specific content for the selected tab, providing a seamless and responsive user interface. The script also includes a function, loadTabContent(tabId) which fetches the content dynamically using AJAX.


Home Processing Dashboard Report
Logout

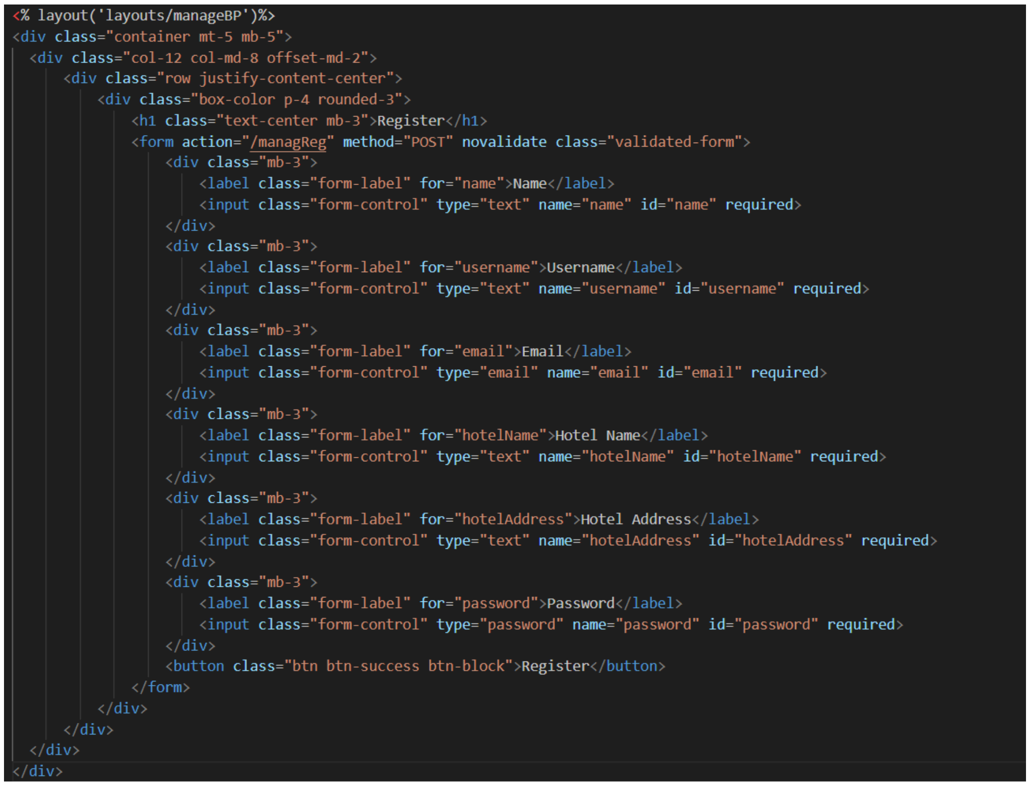
Dashboard - Luxury Grand Hotel

General House Keeping Content Bathroom Maintenance Service

Title	Smells like poop	Open
Hotel rat	There was a rat in my hotel room	Open

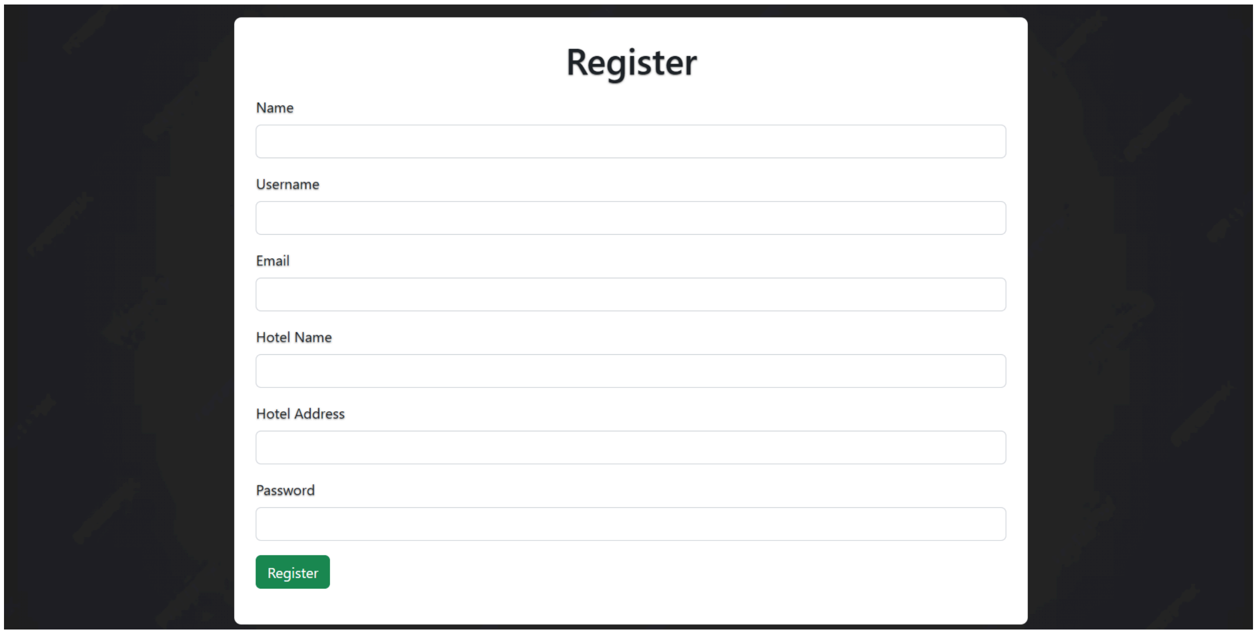
© 2023 Hotel Feedback Portal

'hotel' has EJS files related to manager login and register.

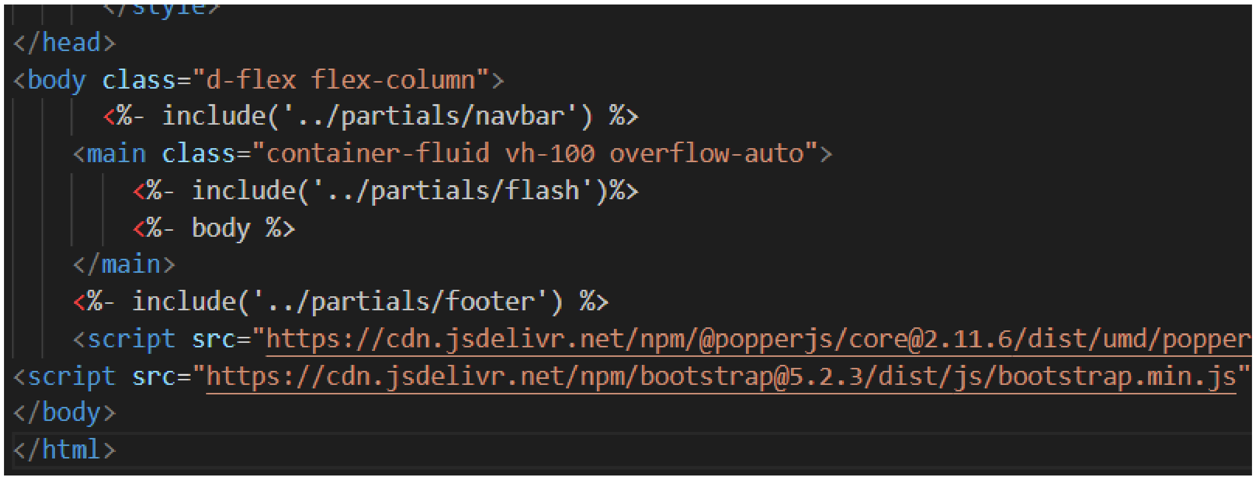


```
<% layout('layouts/manageBP')%>
<div class="container mt-5 mb-5">
  <div class="col-12 col-md-8 offset-md-2">
      <div class="row justify-content-center">
          <div class="box-color p-4 rounded-3">
              <h1 class="text-center mb-3">Register</h1>
              <form action="/managReg" method="POST" novalidate class="validated-form">
                  <div class="mb-3">
                      <label class="form-label" for="name">Name</label>
                      <input class="form-control" type="text" name="name" id="name" required>
                  </div>
                  <div class="mb-3">
                      <label class="form-label" for="username">Username</label>
                      <input class="form-control" type="text" name="username" id="username" required>
                  </div>
                  <div class="mb-3">
                      <label class="form-label" for="email">Email</label>
                      <input class="form-control" type="email" name="email" id="email" required>
                  </div>
                  <div class="mb-3">
                      <label class="form-label" for="hotelName">Hotel Name</label>
                      <input class="form-control" type="text" name="hotelName" id="hotelName" required>
                  </div>
                  <div class="mb-3">
                      <label class="form-label" for="hotelAddress">Hotel Address</label>
                      <input class="form-control" type="text" name="hotelAddress" id="hotelAddress" required>
                  </div>
                  <div class="mb-3">
                      <label class="form-label" for="password">Password</label>
                      <input class="form-control" type="password" name="password" id="password" required>
                  </div>
                  <button class="btn btn-success btn-block">Register</button>
              </form>
          </div>
      </div>
  </div>
</div>
```

This code creates a registration form for a manager in a web application. The form is styled using Bootstrap classes for a clean and responsive design. The registration form collects essential information such as name, username, email, hotel name, hotel address, and password. The <% layout('layouts/manageBP')%> statement specifies the layout template used for this particular view, indicating the overall structure of the page. The accompanying JavaScript script enhances form validation by preventing default form submission and checking form validity before submission. It utilizes Bootstrap's custom validation styles to provide visual feedback to users, highlighting any validation errors. The script employs the Immediately Invoked Function Expression (IIFE) to encapsulate its functionality, ensuring that variables and functions do not leak into the global scope(so that no one can submit the form using "postman" on the backend).



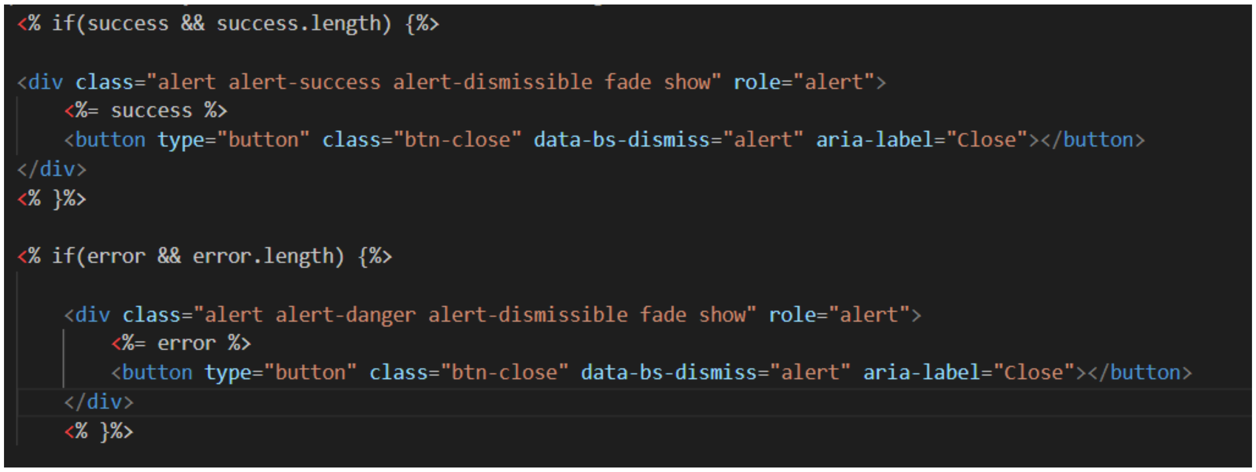
'**Layouts**' hold webpage boilerplates.



```
        </style>
</head>
<body class="d-flex flex-column">
        <%- include('../partials/navbar') %>
    <main class="container-fluid vh-100 overflow-auto">
        <%- include('../partials/flash')%>
        <%- body %>
    </main>
    <%- include('../partials/footer') %>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.min.js"
</body>
</html>
```

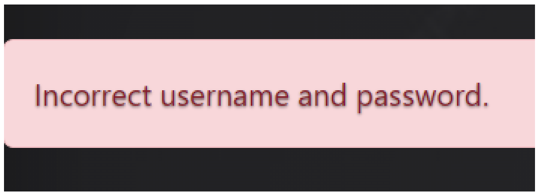
This HTML code defines the structure and styling for the management dashboard's main layout. It includes Bootstrap for styling and responsiveness. The background is set with a gradient and an image. The layout features a navigation bar, a main content area,

and includes partials for the navbar, flash messages, and footer. It ensures a visually appealing and functional design for the management interface, promoting a seamless user experience.
'Partials' encompass components like flash messages, footer, and navbar.

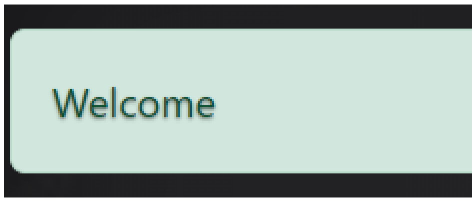


```
<% if(success && success.length) {%>

<div class="alert alert-success alert-dismissible fade show" role="alert">
    <%= success %>
    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
</div>
<% }%>

<% if(error && error.length) {%>

    <div class="alert alert-danger alert-dismissible fade show" role="alert">
        <%= error %>
        <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
    </div>
    <% }%>
```

These snippets are commonly used in web applications to provide feedback to users after certain actions (e.g., form submissions, requests). If there is a success message (success variable exists and has a length greater than 0). If a success message is present, it renders an alert with a green background, displaying the success message. The alert includes a close button for the user to dismiss it. If there is an error message (error variable exists and has a length greater than 0). If an error message is present, it renders an alert with a red background, displaying the error message. Similarly, it includes a close button for the user to dismiss the alert.

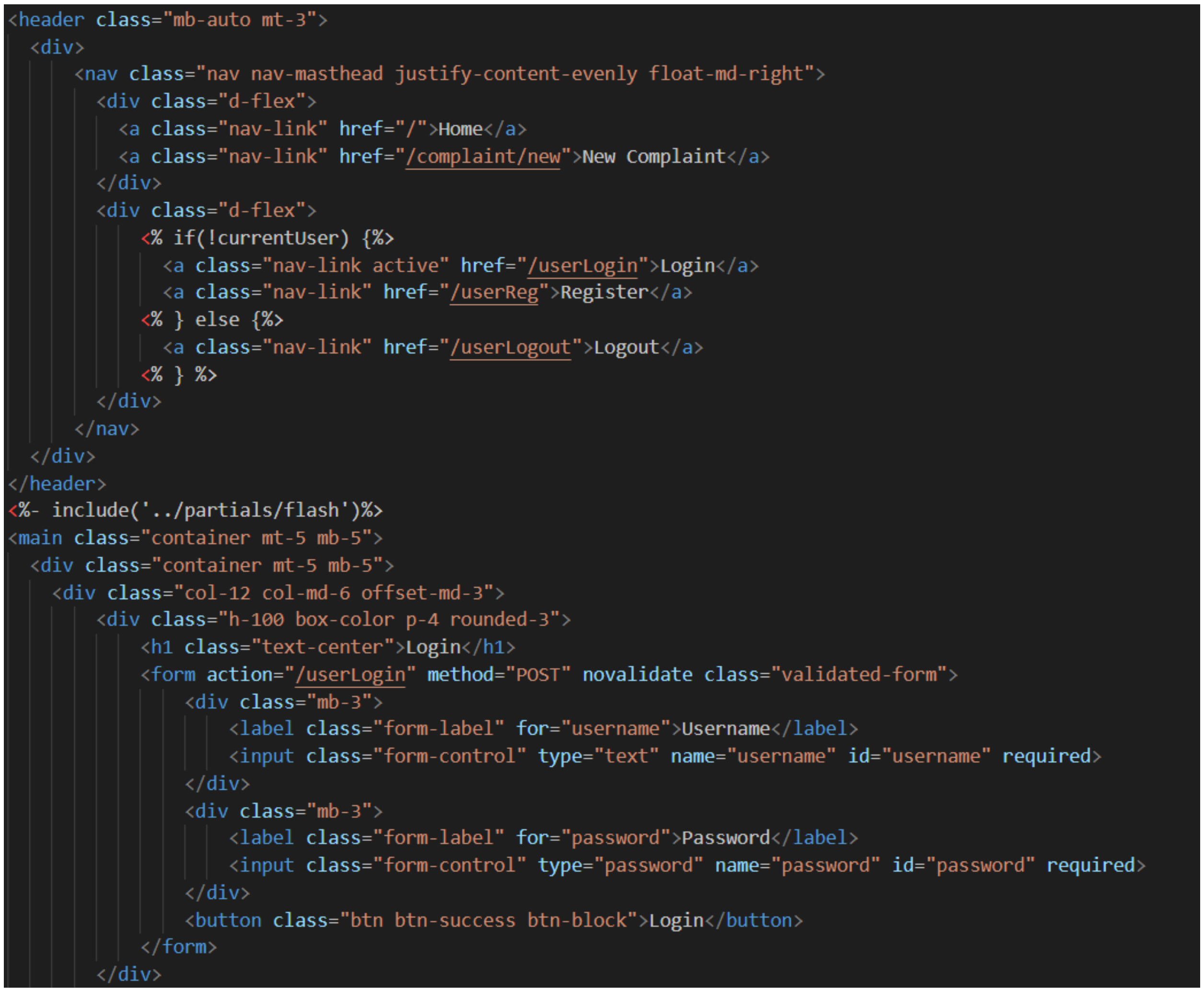


Incorrect username and password.



Welcome

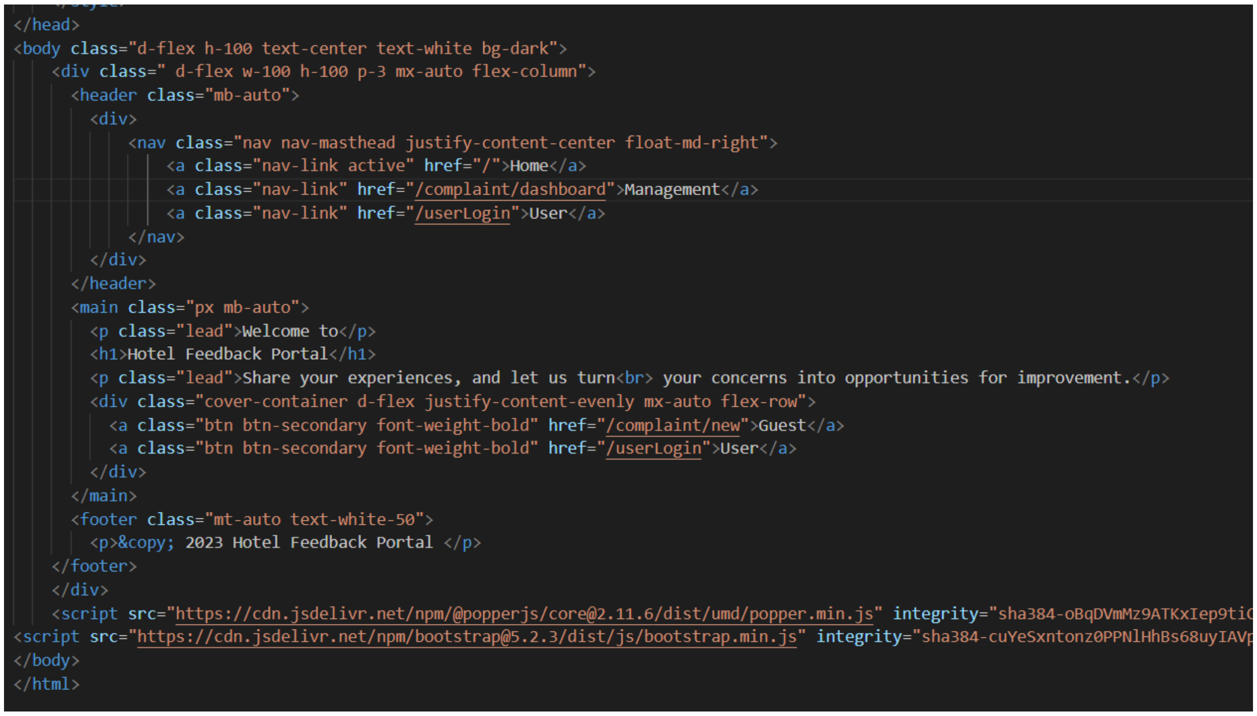
The **'user'** directory contains 'login.ejs' and 'register.ejs.'



```
<header class="mb-auto mt-3">
  <div>
      <nav class="nav nav-masthead justify-content-evenly float-md-right">
        <div class="d-flex">
          <a class="nav-link" href="/">Home</a>
          <a class="nav-link" href="/complaint/new">New Complaint</a>
        </div>
        <div class="d-flex">
            <% if(!currentUser) {%>
              <a class="nav-link active" href="/userLogin">Login</a>
              <a class="nav-link" href="/userReg">Register</a>
            <% } else {%>
              <a class="nav-link" href="/userLogout">Logout</a>
            <% } %>
        </div>
      </nav>
  </div>
</header>
<%- include('../partials/flash')%>
<main class="container mt-5 mb-5">
  <div class="container mt-5 mb-5">
    <div class="col-12 col-md-6 offset-md-3">
        <div class="h-100 box-color p-4 rounded-3">
            <h1 class="text-center">Login</h1>
            <form action="/userLogin" method="POST" novalidate class="validated-form">
                <div class="mb-3">
                    <label class="form-label" for="username">Username</label>
                    <input class="form-control" type="text" name="username" id="username" required>
                </div>
                <div class="mb-3">
                    <label class="form-label" for="password">Password</label>
                    <input class="form-control" type="password" name="password" id="password" required>
                </div>
                <button class="btn btn-success btn-block">Login</button>
            </form>
        </div>
```

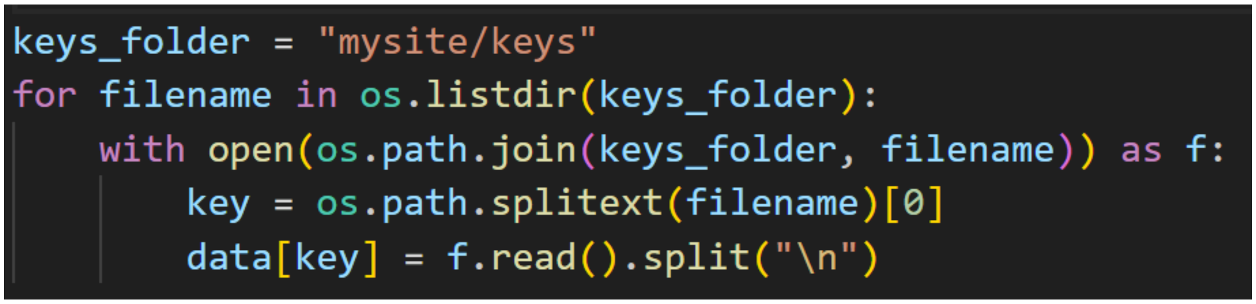
This code defines a webpage for user authentication in a web application. The header includes navigation links for home, submitting a new complaint, and user login/register/logout based on the user's authentication status. The main section contains a login form with Bootstrap styling, and client-side form validation is added to enhance user experience by preventing invalid form submissions. Flash messages, which provide feedback on user actions, are also included in the layout for a comprehensive and user-friendly interface. The embedded script uses an Immediately Invoked Function Expression (IIFE) to apply custom Bootstrap validation styles to the forms. It prevents form submission if the form is not valid, providing a client-side validation mechanism for better user experience.

The root '**Error.ejs' and 'home.ejs'** are present within the views folder,



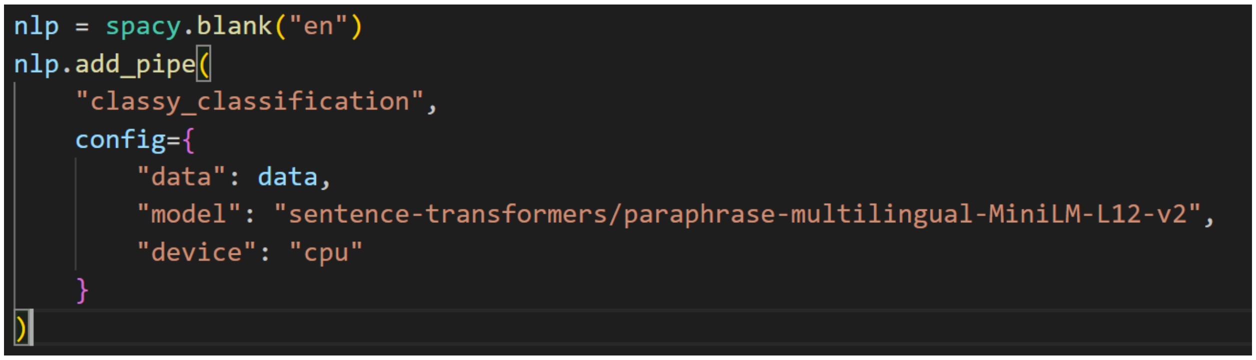
```html
</head>
<body class="d-flex h-100 text-center text-white bg-dark">
    <div class=" d-flex w-100 h-100 p-3 mx-auto flex-column">
      <header class="mb-auto">
        <div>
            <nav class="nav nav-masthead justify-content-center float-md-right">
                <a class="nav-link active" href="/">Home</a>
                <a class="nav-link" href="/complaint/dashboard">Management</a>
                <a class="nav-link" href="/userLogin">User</a>
            </nav>
        </div>
      </header>
      <main class="px mb-auto">
        <p class="lead">Welcome to</p>
        <h1>Hotel Feedback Portal</h1>
        <p class="lead">Share your experiences, and let us turn<br> your concerns into opportunities for improvement.</p>
        <div class="cover-container d-flex justify-content-evenly mx-auto flex-row">
          <a class="btn btn-secondary font-weight-bold" href="/complaint/new">Guest</a>
          <a class="btn btn-secondary font-weight-bold" href="/userLogin">User</a>
        </div>
      </main>
      <footer class="mt-auto text-white-50">
        <p>&copy; 2023 Hotel Feedback Portal </p>
    </footer>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js" integrity="sha384-oBqDVmMz9ATKxIep9tiC
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.min.js" integrity="sha384-cuYeSxntonz0PPNlHhBs68uyIAVp
</body>
</html>
```

This HTML code establishes the front-end structure for the Hotel Feedback Portal's landing page. Utilizing Bootstrap for styling, it presents a visually appealing interface with navigation links, a welcome message, and options for guests and users to provide feedback. The background image, styling, and layout contribute to an inviting and user-friendly portal for sharing experiences and improving hotel services.



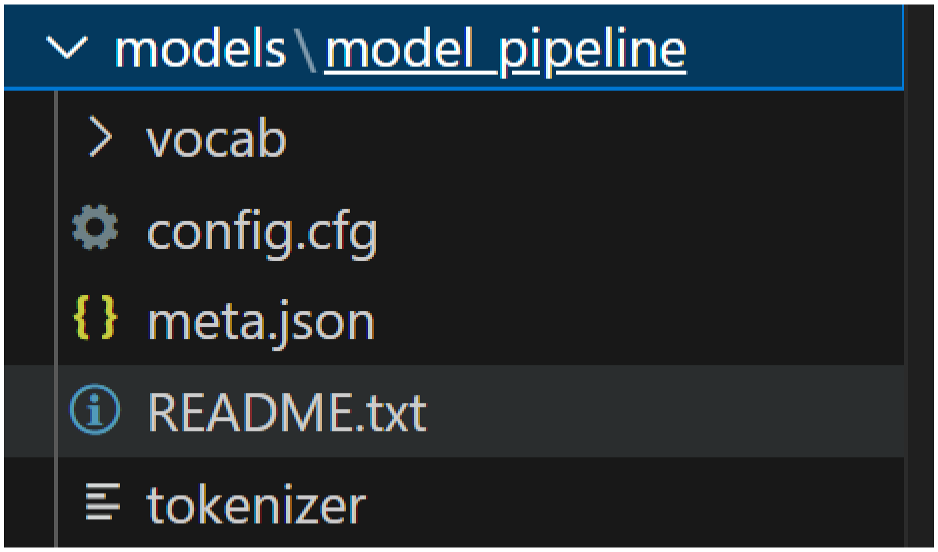
```python
keys_folder = "mysite/keys"
for filename in os.listdir(keys_folder):
    with open(os.path.join(keys_folder, filename)) as f:
        key = os.path.splitext(filename)[0]
        data[key] = f.read().split("\n")
```

Keys contains pre-built dictionary entries, with each file being labeled using the name of the key, and containing the values for the pair. Each key[value(s)] pair represents the *paraphrase reduced* labeling criteria. Basically, it contains the barebones sentence structure for a given complaint, and the label to assign to it. The model compares similarity against the keys, and assigns a “weight” to each, which is how we classify them.



```
nlp = spacy.blank("en")
nlp.add_pipe(
    "classy_classification",
    config={
        "data": data,
        "model": "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2",
        "device": "cpu"
    }
)
```

Models contains our specialized variant of PMMLM v2, which is loaded through Classy Classification into the pipeline. This is the high level operation that loads the entire model into memory from the exported model data.



```
models\model_pipeline
    vocab
    config.cfg
    meta.json
    README.txt
    tokenizer
```

Config and vocab rules, which are used by the sentence-transformer to correctly map sentences, are stored here as well for ease of loading.

'app.js' serves as the main file orchestrating all components and functionality.
https://github.com/fangedShadow/SE_Project/blob/main/app.js (the code was too big to have a Screen shot.
The provided Node.js code defines a web application using the Express.js framework for a Hotel Feedback Portal, implementing various features such as user authentication, complaint management, and reporting. It integrates with MongoDB using Mongoose for data storage and retrieval. Let's break down the key functionalities:

Initialization and Database Connection: The code begins by importing necessary libraries, including Express, Mongoose, and Passport for authentication. It establishes a connection to the MongoDB database, and the seedDB function initializes the database with sample data, specifically a sorted complaint with predefined details.

Middleware Setup: Express middleware, such as express-session and connect-flash, is configured for session management and displaying flash messages. Passport middleware is set up for user authentication, supporting both regular users and managers. The passport-local strategy is employed for local username and password authentication.

Routes and Views: The application defines various routes to handle user registration, login, and logout for both regular users and managers. Different views are rendered based on the user type. Complaint-related routes facilitate CRUD (Create, Read, Update, Delete) operations. Complaints are associated with hotels, and managers can view complaints based on predefined labels, such as bathroom issues or general complaints.

V) Earned Value Analysis

Today's date: - Dec 7 2023

Job No.	Job description	Est. Days	Actual Days	Due Date	Comp. Date
1	Proposal	7	7	Nov 1	Nov 1
2	Model and Design	8	8	Nov 9	Nov 9
3	Working website	9	17	Nov 18	Nov 26
4	Ml integration	7	6	Nov 25	Dec 3
5	Completion of APP	5	1	Nov 30	Dec 5
6	Deployment	5	0	Dec 5	Dec 5
7	Submission	3	3	Dec 8	

1) BAC = job 1 to job 7 = 44 days
2) BCWS = job 1 to job 6 = 41 days
3) BCWP = job 1 to job 6 = 41 days
4) EV = BCWP/BAC = 41/44 = 93%
5) SPI = BCWP/BCWS = 41/41 = 100%

6) SV = BCWP/BCWS = 41-41 = 0 (0 Productive Days ahead)
7) ACWP = job 1 to job 6 = 41 days
8) CV = BCWP - ACWP = 41 -41 = 0 (0 Days ahead)
9) CPI = BCWP/ACWP = 41/41 = 100%
10) Both the schedule and cost performance indices are 100%. CV and SC we are neither ahead or Behind our Project ,indicating that the project is on track according to the original plan. However, the EV percentage of 93% suggests that there is a slight delay in completing the planned work.

VI) Testing

All testing done for this project was **white-box**.

Principally, there were 3 distinct tests run throughout the project's life cycle. First, we had to test the UI and features of the website. We were entirely interested in the inner workings of the website, and ensuring all features were working as intended (rather than *appearing* to work as intended), so white-box testing was required. Functionality was fairly low in mechanics, so we were certain that if the underlying features worked, the functionality would be as intended.

Second, we had to test that the Model (for ML) was learning as we desired. This is often referred to as supervised learning, and is a form of white-box testing. At the end of the day, we want to see and keep track of the desired results from ML, and more importantly, keep an eye on how it reached those results. So all testing related to ML was done while keeping an eye on inner workings, and by extension white-box.

Third and finally, we had to test the dashboard's ability to correctly sort the data, which the manager would see. Since the functionality of the dashboard is super lightweight, and all of the real operation occurs from the back-end, we needed to carefully observe (using logging and verbose labeling) how the categorized information was labeled and then sorted into super-groups. This would, like the others, be white-box testing, since it focused exclusively on the inner mechanics of the software.

VII) Conclusion

In conclusion, the Node.js-based Hotel Feedback Portal is a simple and easy to use product for the hospitality industry, offering a user-friendly interface for anonymous guest feedback and empowering managers with robust tools for efficient complaint handling. The integration of machine learning, driven by spaCy and Classy

Classification, sets this portal apart, providing automated and accurate complaint categorization. The MongoDB database, Cloudinary for image management, and Passport for user authentication collectively contribute to a secure and scalable system. The chosen Feature-Driven Agile model, coupled with the Earned Value Analysis, showcases the project's strategic planning and successful execution. Both schedule and cost performance indices remain at 100%, indicating the project's overall adherence to the established plan.

Limitations & Challenges

Our biggest challenge (and limitation!) is a severe lack of public datasets on specific complaints. Despite huge volumes of “general feedback” being available for public use, our application handles more specific feedback, and data for it is either expensive (paid datasets) or outright unavailable.

Our solution for this was to hand-write and generate all of our testing data ourselves. We used approximately ~600 unique complaints, and trained the model to paraphrase each of them into similar variants. Some of the original testing data was hand-written, and some was generated with GPT-4.

Ultimately, our Model will never be as accurate as we want it to be without a huge supply of testing data (millions of documents), but it will get the job done with a fair degree of accuracy.

Getting the model to load into memory within a timely manner was also extremely tricky. Despite being a fairly specialized task, paraphrase models are by definition large, as they require many layers of pre-mapped data. Many small optimizations lead to it being loaded quickly, but overall it was a challenge to setup.

We also experienced some challenges with porting the complaint form to mobile, which required some clever uses of Bootstrap to adapt.

Finally, the security concerns associated with running Python scripts locally (within the web server) were a major hurdle to overcome. We had to think about different solutions to avoid putting our users at risk, and eventually settled on moving all python to an off-site server.

Future Enhancements

We want to Address the major limitation of the current version of the application. As AI improves, we can leverage it to generate far larger sets of data, and further refine, test and tune our Model. Google's Gemini was *just* released, and looks extremely promising as a way to further bolster our Models accuracy.

Enhancements to the user interface are next on the list, with the general goal of making submitting forms even easier and faster. We also want to improve on the managers dashboard and report generation, to include more statistics and more options for filtering.

Finally, we want to add functionality to split larger complaints into more granular components, which can be stored as separate complaints. This way we can more accurately keep track of *exactly* what went wrong in a person's stay at the hotel.

Team members

Ethan: Lead, Machine Learning, Training
Bhavpreet: Front-end, Back-end, Database
Jaan: UEX, Training Data
Minglok: Testing, Training Data

Learning outcomes

Ethan: This was a tough but really rewarding experience. The part of the project I primarily focused on was ML, which for our somewhat specialized task, was fairly tricky to learn and implement correctly. Leading in general isn't an easy task, as balancing other's scheduling and working out times to meet and discuss is often a really tedious task. Personally, the biggest challenge I faced was getting the pipeline to efficiently ingest batched complaints, so that the demo would work in a timely manner.

Bhavpreet: Engaging in this project was both enjoyable and challenging, as it involved learning new concepts within a tight schedule, often colliding with other projects. The experience was a mix of fun and stress, particularly when dealing with system errors or script issues. One notable challenge was spending over two hours troubleshooting a login issue caused by a simple typo. I tackled these challenges by utilizing console.log to identify errors or null values. Despite the hurdles, the outcome was rewarding— a consolidated review of web development skills, with added proficiency in API use, CSS, and JavaScript concepts.

Jaan: I really enjoyed working on this project. I primarily focused on user experience and training data. I learned MongoDB and Mongoose to make the user experience more enjoyable and efficient on the website. MongoDB makes the website have high performance and we needed the website to perform like this because we use machine learning. For the training data I just thought about all the things someone could complain about and typed them out. After that I organized them into different categories. The most challenging part of this project for me was learning MongoDB and Mongoose in a reasonable time so we were able to use for the project.

Minglok:

References

1. Bootstrap: https://getbootstrap.com/docs/5.3/getting-started/introduction/
Bootstrap. (n.d.). Introduction. Retrieved from https://getbootstrap.com/docs/5.3/getting-started/introduction/
2. Cloudinary: https://cloudinary.com
Cloudinary. (n.d.). Home. Retrieved from https://cloudinary.com
3. EJS: https://ejs.co/#docs
EJS. (n.d.). Documentation. Retrieved from https://ejs.co/#docs
4. Express.js: https://expressjs.com/en/api.html
Express.js. (n.d.). API reference. Retrieved from https://expressjs.com/en/api.html
5. Express Session: https://www.npmjs.com/package/express-session
npm. (n.d.). express-session package. Retrieved from https://www.npmjs.com/package/express-session
6. Flask: https://flask.palletsprojects.com/en/2.3.x/views/
Flask. (n.d.). Views and Templates. Retrieved from https://flask.palletsprojects.com/en/2.3.x/views/
7. Mongoose: https://mongoosejs.com/docs/
Mongoose. (n.d.). Documentation. Retrieved from https://mongoosejs.com/docs/
8. Multer: https://github.com/expressjs/multer
Multer. (n.d.). GitHub repository. Retrieved from https://github.com/expressjs/multer
9. Node.js Guides: https://nodejs.org/en/guides
Node.js. (n.d.). Guides. Retrieved from https://nodejs.org/en/guides
10. Passport.js: https://www.passportjs.org/docs/
Passport.js. (n.d.). Documentation. Retrieved from https://www.passportjs.org/docs/
11. PDFKit: https://pdfkit.org
PDFKit. (n.d.). Home. Retrieved from https://pdfkit.org

12. Render: https://render.com/docs
    Render. (n.d.). Documentation. Retrieved from https://render.com/docs
13. spaCy: https://spacy.io/usage
    spaCy. (n.d.). Usage Documentation. Retrieved from https://spacy.io/usage