AnomXplorer

PROJECT SUPERVISOR

DR. Atif Tahir

PROJECT CO-SUPERVISOR

DR. Nouman Durrani

PROJECT TEAM

Hafsa Baig	K20-1683
Samia Azeem	K20-1685
Maryam Raheem	K20-1700

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

FAST SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES KARACHI CAMPUS

December 2023

Project Supervisor	Mr. Atif Tahir
Project Team	Hafsa Baig K20-1683 Samia Azeem K20-1685 Maryam Raheem K20-1700
Submission Date	May 11, 2024

Mr. Supervisor Name _

Supervisor

Mr. Co-Supervisor Name ____________________

Co-Supervisor

Dr. Zulfiqar Ali Memon

____

Head of Department

FAST SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

KARACHI CAMPUS

Acknowledgement

We would like to extend our sincere gratitude to Dr. Atif Tahir for invaluable guidance and mentorship throughout the entire project.

Abstract

This project presents an anomaly detection system tailored for hospital environments utilizing machine learning techniques. Leveraging a decision tree algorithm, our system offers real-time analysis of hospital data to identify anomalous patterns that indicate potential issues or irregularities. The implementation integrates a React.js front end and a Node.js backend with Express framework as well as Mongodb database to provide a user-friendly interface for administrators to input hospital data at database endpoint. Preprocessing techniques are employed to prepare the data for analysis, and the resulting anomalies are visually presented on the frontend through a dynamic table. Additionally, a dashboard provides insights into the frequency of anomalies detected and overall patient statistics. Through rigorous testing and evaluation, the effectiveness of our system in detecting anomalies is demonstrated, paving the way for enhanced monitoring and management within hospital settings.To alert the admin of anomaly we have a notification alert that also sends notification on Gmail related to the corresponding anomaly. This guarantees a better check on all anomalies and to rectify the anomaly as soon as it hits our system.By looking at the model of decision tree we can easily find the root cause of our anomaly and trigger the alert to the user at Gmail.

Contents

List of Figures

List of Tables

Introduction

In today's rapidly evolving healthcare landscape, the efficient functioning of hospital systems is essential for providing quality care to patients. However, ensuring the reliability and integrity of these systems poses significant challenges, particularly in detecting anomalies and irregularities that may compromise patient safety or operational efficiency. Traditional methods of testing and monitoring hospital systems often fall short in detecting subtle anomalies indicating potential issues and manually testing and detecting anomalies require a lot of effort and time.
Against this backdrop, the integration of ML techniques into hospital systems testing has emerged as an advanced approach to address these challenges. By leveraging the power of ML algorithms, healthcare administrators can gain valuable insights into the performance and integrity of hospital systems, enabling fast identification of risks.

Need For The Product:

The need for an anomaly detection system tailored for hospital systems stems from several critical factors:

- Manual Testing: Hospitals are testing their system manually or automated but still require resources and a lot of time so our project aims to detect anomalies using machine learning algorithms to be more effective and fast.

- Patient Safety and Care Quality: Anomalies within hospital systems have potential to threaten patient safety and care quality. Detecting and addressing these anomalies in a timely manner is paramount to ensuring optimal patient outcomes.

- Operational Efficiency: By identifying and addressing these anomalies, hospitals can streamline operations and optimize resource allocation.

Benefits of the Product:

The proposed AnomXplorer offers several benefits to healthcare organizations and stakeholders:

- Prevention of Extra charges and Unauthorized fees: Hospital bills often include various charges and fees for services. However, anomalies such as unauthorized fees or excessive charges can result in financial burdens for patients and undermine trust in healthcare institutions.

- Early Anomaly Detection: By leveraging ML algorithms, the system can detect anomalies in hospital systems data in real-time, enabling early intervention of potential risks.

- Data-driven insights: By providing actionable insights into the performance and integrity of hospital systems, the healthcare administrators can make informed decisions and implement targeted interventions to improve system reliability and performance.

RelATED WORK

Features	Weka Data Mining	Shogun	RapidMiner	**AnomXplorer**
Real-time Detection	✘	✘	✘	✔
Anomaly Detection	✔	✔	✔	✔
Smart Alerts	✘	✘	✘	✔
Bring Data from Source like Database	✔	✘	✘	✔
Dashboard	✘	✘	✘	✔
Machine Learning Algorithm	✔	✔	✔	✔
Data Preprocessing	✔	✔	✔	✔
Data Classification	✔	✔	✔	✔
Implementation of ML Pipeline	✘	✘	✘	✔

Table 1. Competitive Analysis

The competitive analysis table highlights the varying capabilities of four anomaly detection systems: Weka Data Mining, Shogun, RapidMiner, and AnomXplorer. While all systems excel in fundamental anomaly detection and employ machine learning algorithms, Our system emerges as a standout with its real-time detection feature, providing immediate identification of anomalies as they occur. Additionally, AnomXplorer offers smart alerts to notify users of detected anomalies and a dashboard for visual insights into anomaly detection results and system performance. Moreover, AnomXplorer implements an end-to-end machine learning pipeline, streamlining the process from data preprocessing to anomaly detection, making it well-suited for applications requiring timely detection and response to anomalies in dynamic datasets, such as those found in hospital environments.

In contrast, Weka Data Mining, Shogun, and RapidMiner lack real-time detection, smart alerts, and dashboard visualization features, limiting their effectiveness in scenarios requiring immediate anomaly detection and proactive decision-making. However, they still provide essential capabilities such as data preprocessing and classification, making them viable options for less time-sensitive anomaly detection tasks. AnomXplorer's comprehensive feature set positions it as a robust solution for organizations seeking advanced anomaly detection capabilities coupled with real-time monitoring and actionable insights, particularly in healthcare where timely detection and response to anomalies are critical.

Requirements

1. Functional Requirements

1.1. Functional Hierarchy

1.1.1 Data Ingestion and preprocessing

Sub-Function 1: Retrieve and preprocess hospital data from database.

Sub-Function 2: Validate and clean data for consistency.

1.1.2 Anomaly Detection Module

Sub-Function 1: Apply machine learning algorithms to identify anomalies.

Sub-Function 2: Evaluate patterns and deviations in the data.

1.1.3 Alert Generation

Sub-Function 1: Generate alerts for detected anomalies.

1.1.4 Notification System

Sub-Function : Notification alert.

1.1.5 Dashboard

Sub-Function 1: Display anomaly statistics.

Sub-Function 2: Visualize anomaly trends.

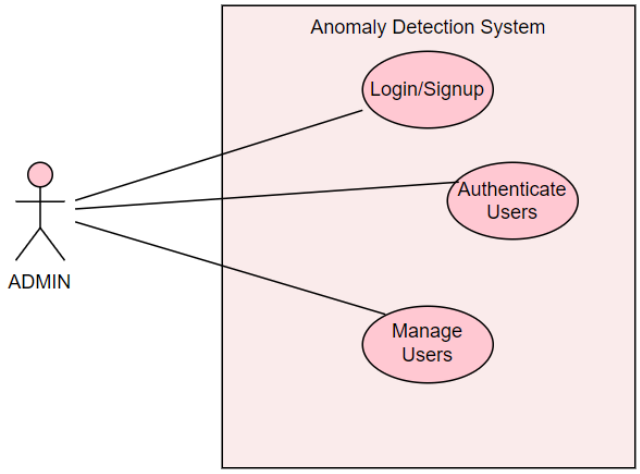
1.1.6 User Authentication

Sub-Function 1: Authenticate users securely.

2. USE CASES

2.1 Login and Authentication

Use Case Diagram:



***Actor**:Admin*

***Description**: The admin will login into the system and also manage authentication settings of other users so that no invalid user will be accessing the system.*

UC001: Login and Authentication			
Use case Id:	UC001		
Actors: Admin			
Feature: login and authentication			
Pre-condition:	TheAdmin has access rights to configure authentication settings		
Scenarios ***1. The admin authenticates users' access.*** ***2. User roles and permissions are managed.***			
Step#	***Action***	***Software Reaction***	

1.	The admin will login to the system.	The system authenticates admin details and moves to home page.
2.	The admin will manage users.	The system will allow admin to manage users.

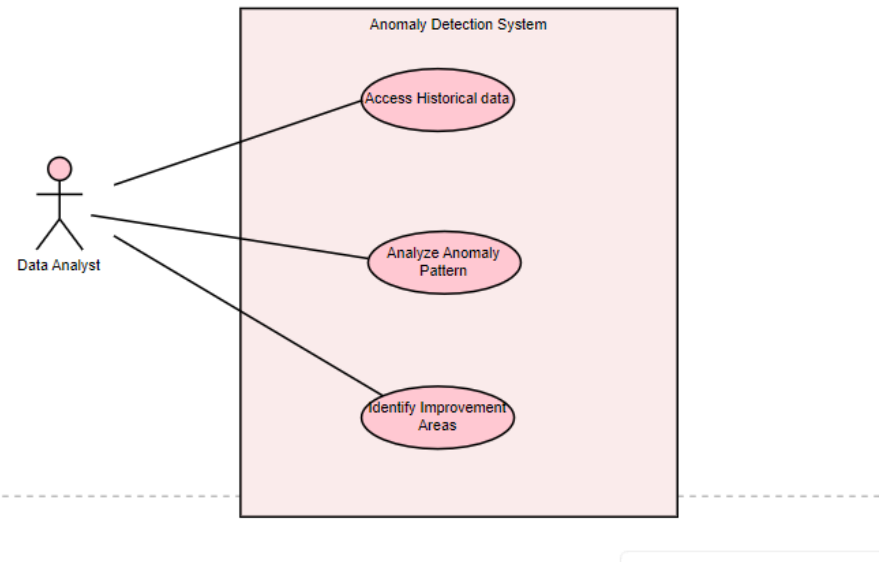
Alternate Scenarios: Following are some alternatives
1a: If a user enters invalid credentials, the system will only give 3 chances to enter valid credentials.

Post Conditions	
Step#	***Description***
1.	Login and authentication settings successfully configured by the administrator.

Use Case Cross referenced	Authenticate User

2.2 Reviewing Anomaly Trends

Use Case Diagram:



Use Case Description:

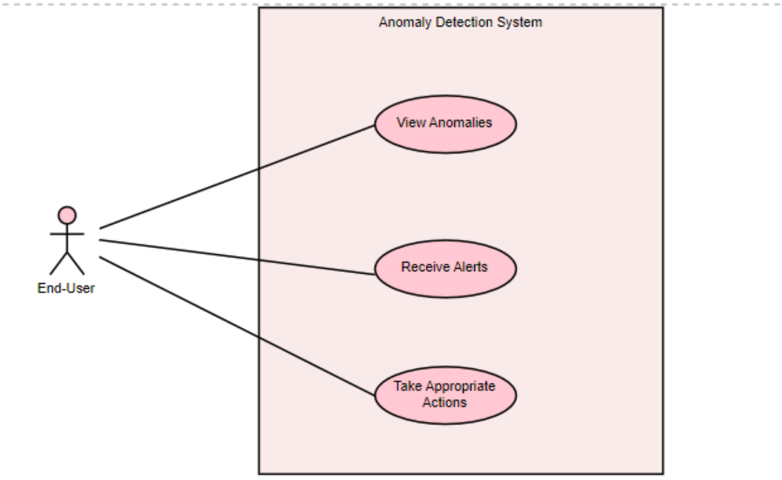
Actor: Data Analyst

Description: The data analyst interacts with the system to review historical anomaly trends. This use case includes analyzing anomaly patterns, and identifying what improvements can be done to overcome these.

UC002: Reviewing Anomaly Trends		
Use case Id:	UC002	
Actors: Data Analyst		
Feature: Anomaly Analysis		
Pre-condition:	The system has historical data available for analysis.	
Scenarios 1. ***The Data analyst access data through system dashboard*** 2. ***Anomaly patterns are analyzed.*** 3. ***Identify areas of improvement.***		
Step#	***Action***	***Software Reaction***
1.	The data analyst navigates to the historical data section.	The system loads historical data on the dashboard.
2.	The data analyst identifies patterns.	The system may suggest areas for improvement..
Alternate Scenarios: N/A		
Post Conditions		
Step#	***Description***	
1.	The data analyst has gained insights into historical anomaly trends and found areas for improvement.	
Use Case Cross referenced	Authenticate User, logged in	

2.3 Analyzing Anomalies

Use Case Diagram:



Use Case Description:

***Actor**: End-User*

***Description**: The system will allow end users to view real-time anomalies, receive alerts, and take appropriate actions in response to detected anomalies.*

<table>
<tr><th colspan="3">UC003: Analyzing Anomalies</th></tr>
<tr><td colspan="2">Use case Id:</td><td>UC003</td></tr>
<tr><td colspan="3">Actors: End-User, Analyst</td></tr>
<tr><td colspan="3">Feature: Anomaly Analysis</td></tr>
<tr><td colspan="2">Pre-condition:</td><td>The system is operational and has access to real-time log data</td></tr>
<tr><td colspan="3">Scenarios
1. The hospital staff views anomalies on the system dashboard
2. End-User takes appropriate actions in response to alerts
3. The system generates alerts for detected anomalies.</td></tr>
<tr><td>Step#</td><td>Action</td><td>Software Reaction</td></tr>
<tr><td>1.</td><td>The end user navigates to the system dashboard</td><td>The system displays real-time anomalies.</td></tr>
<tr><td>2.</td><td>The end user reviews the list of anomalies</td><td>The system visualizes real-time anomalies data on the dashboard.</td></tr>
<tr><td>3.</td><td>Anomaly detection module identifies a critical anomaly.</td><td>The system generates an alert.</td></tr>
<tr><td>4.</td><td>Hospital staff receives the alert notification</td><td>The system prioritizes the alert based on severity..</td></tr>
<tr><td colspan="3">Alternate Scenarios: Following are some alternatives</td></tr>
</table>

1a: If no anomalies, the system will display 'no anomalies' ***2a: The system prioritizes alerts.***		
Post Conditions		
Step#	***Description***	
1.	The end user has successfully analyzed real-time anomalies and taken appropriate actions..	
Use Case Cross referenced	Authenticate User, logged in	

3. Non-Functional Requirements

3.1 Performance Requirements

3.1.1 Speed

The system must achieve real-time anomaly detection with a response time not exceeding 2.5 seconds. The speed of anomaly detection is crucial for timely decision making and intervention.

3.1.2 Precision

The system is required to achieve a minimum accuracy rate of 95% in detecting anomalies.

3.1.3 Reliability

For continuous monitoring and timely anomaly detection the system is expected to maintain an uptime of at least 99%.

3.2 Safety Requirements

The system must implement robust measures to ensure the confidentiality of hospital data. Access to sensitive information must be restricted to authorized personnel only. To ensure continuous operation, the system should have redundancy and failover mechanisms in place.

3.3 Security Requirements

3.3.1 User authentication and authorization

Access to system functionalities must be role-based, with different user roles having specific permissions. This ensures that users only have access to the functionalities necessary for their roles.

3.3.2 Data security

All the hospital data transmitted and stored by the system must be encrypted.

3.4 User Documentation

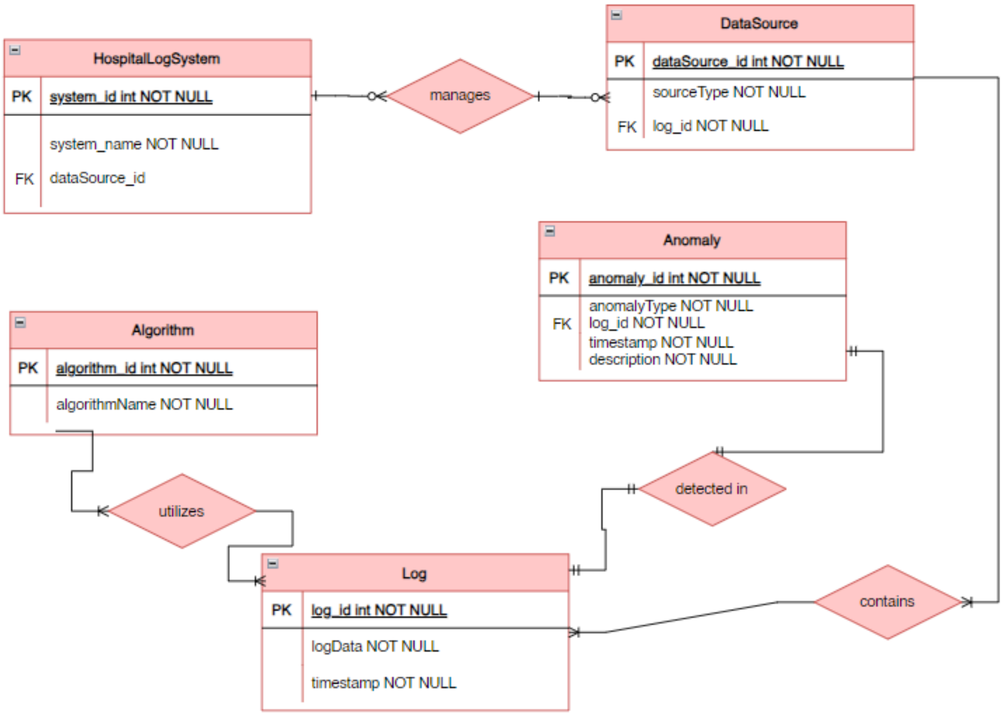
Following is the list of the user documentation components that will be delivered along with the software:

- *Online help*
- *Tutorials*

Design

1. DataBase Design:

1.1 ERD Diagram:



1.2 Data Dictionary:

Data Dictionary provides a detailed description of the entities along with their attributes and their characteristics. Following are the key entities of our anomaly detection system for hospital logs.

1.2.1 HospitalLogSystem:

HospitalLogSystem	
Name	HospitalLogSystem
Alias	system
Where-used/how- used	Initialized whenever a anomaly detection system is activated or started
Content descripti	The content includes hospital log data, which comprises various parameters such as system_id,

on	system_name etc. Utilizing a standardized format (csv or json).

Column Name	**Description**	**Type**	**Length**	**Nullable**	**Default Value**	**Key Type**
system_id	*provides system id*	*int*	*10*	*not nullable*	*0*	*Primary key*
system_name	*provides system name*	*String*	*char[50]*	*not nullable*	*system*	

1.2.2 DataSource:

DataSource	
Name	DataSource
Alias	
Where-used/how- used	Initialized as an attribute of HospitalLogSystem. Gets added or removed in the HospitalLogSystem. Contains logs.
Content description	

Column Name	**Description**	**Type**	**Length**	**Nullable**	**Default Value**	**Key Type**
dataSource_id	*provides data source id*	*int*	*10*	*not nullable*	*0*	*Primary key*

source_ type	*provides the name of the type of the source*	*String*	*char[50]*	*not nullable*	*type*	
system _id	*provides id of the system it is accessed by*	*int*	*10*	*not nullable*	*0*	*Foreign key*

1.2.3 Log:

Log	
Name	Log
Alias	hospital logs, system logs, data
Where-used/how- used	Data Sources contains logs, anomaly detection algorithm utilizes logs. They are processed by hospitalLogSstem functions.

Content description						
Column Name	**Description**	**Type**	**Length**	**Nullable**	**Default Value**	**Key Type**
log_id	*provides log id*	*int*	*10*	*not nullable*	*0*	*Primary key*
log_data	*provides the events that has generated the log*	*String*	*char[70]*	*not nullable*	*data*	
timestamp	*provides the time at which log was generated*	*DateTime*	*20*	*not nullable*	*00:00:00*	
data_source_id	*provides the id of data source that contains the log*	*int*	*10*	*not nullable*	*0*	*Foreign key*
algorithm_id	*provides the id of the algorithm that uses it*	*int*	*10*	*nullable*		*Foreign key*
anomaly_id	*id of anomaly that is created when anomaly is detected in lo*	*int*	*10*	*nullable*		*Foreign key*

1.2.4 Anomaly:

Anomaly

Name	Anomaly
Alias	abnormality
Where-used/how- used	Created when algorithm finds an anomaly in the log. It is detected in the log.
Content description	

Column Name	**Description**	**Type**	**Length**	**Nullable**	**Default Value**	**Key Type**
anomaly_id	*provides id for the anomaly*	*int*	*10*	*not nullable*	*0*	*Primary key*
anomaly_type	*provides type of the anomaly that is created*	*String*	*char[50]*	*not nullable*	*type*	
timestamp	*provides time when anomaly is found*	*DateTime*	*20*	*not nullable*	*00:00:00*	
description	*textual description of the anomaly*	*String*	*char[70]*	*not nullable*	*description*	
log_id	*id of the log where anomaly is detected*	*int*	*10*	*not nullable*	*0*	*Foreign key*

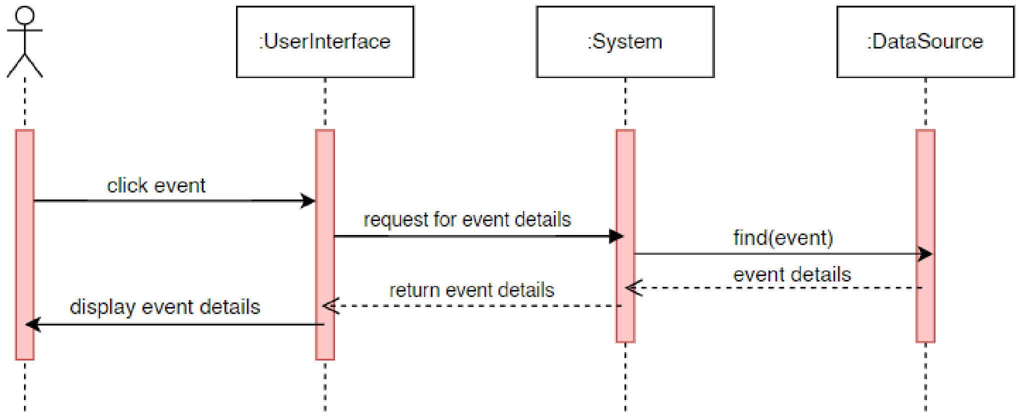
2. Application Design:

   This section describes the working of the system. It represents the chronological interactions of the objects in a Hospital Logs Anomaly Detection System at various events.

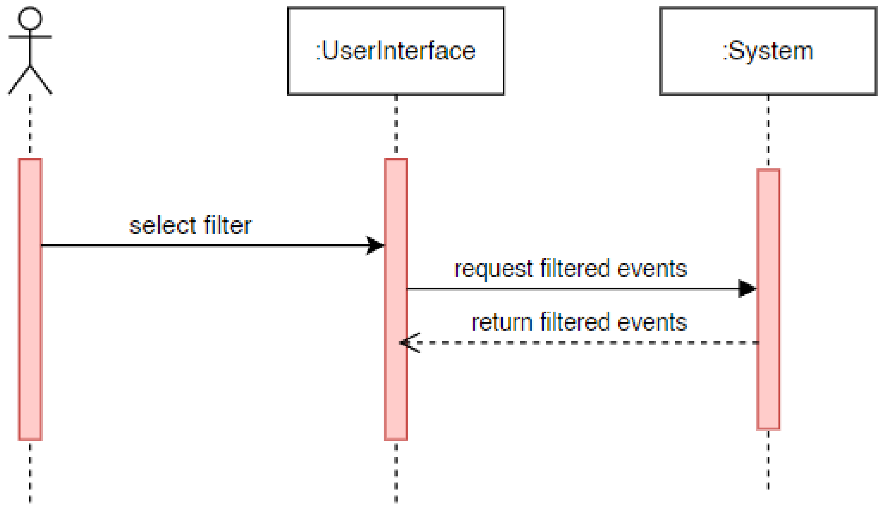
2.1 Sequence Diagram:

2.1.1 Sequence Diagram 1:

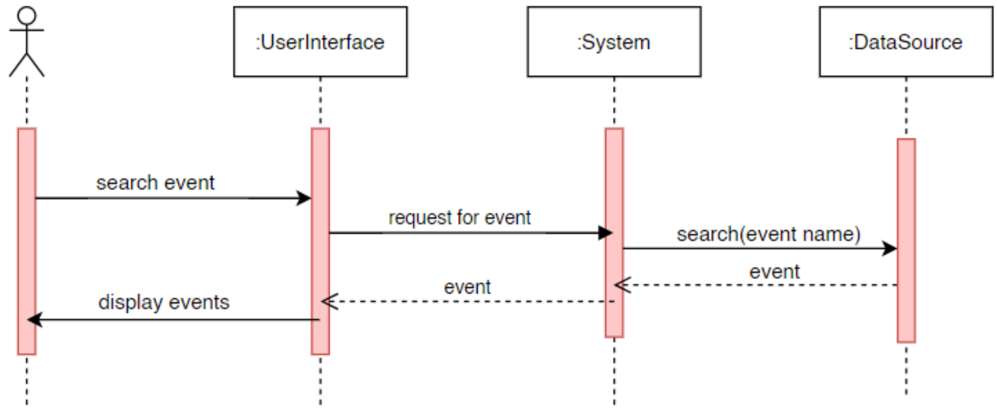
Diagram shows the sequence of interactions between user interface, system and data source when the user clicks at an event and the system sends back the event details.



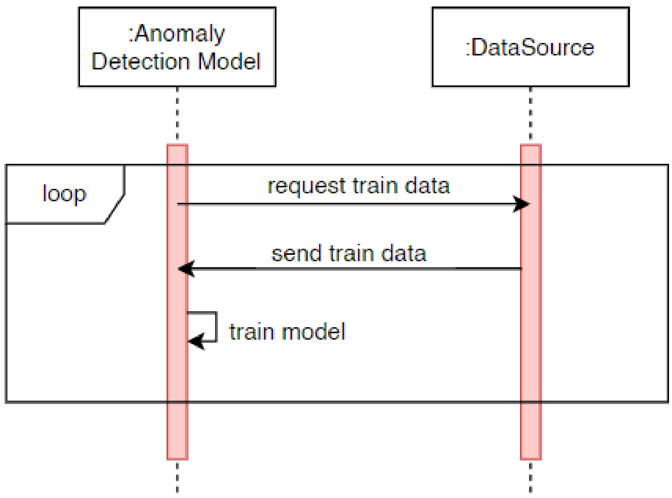
2.1.2 Sequence Diagram 2:
Diagram shows the sequence of interactions between user interface and system when the user chooses a filter from the drop down menu for the events and system returns filtered events.



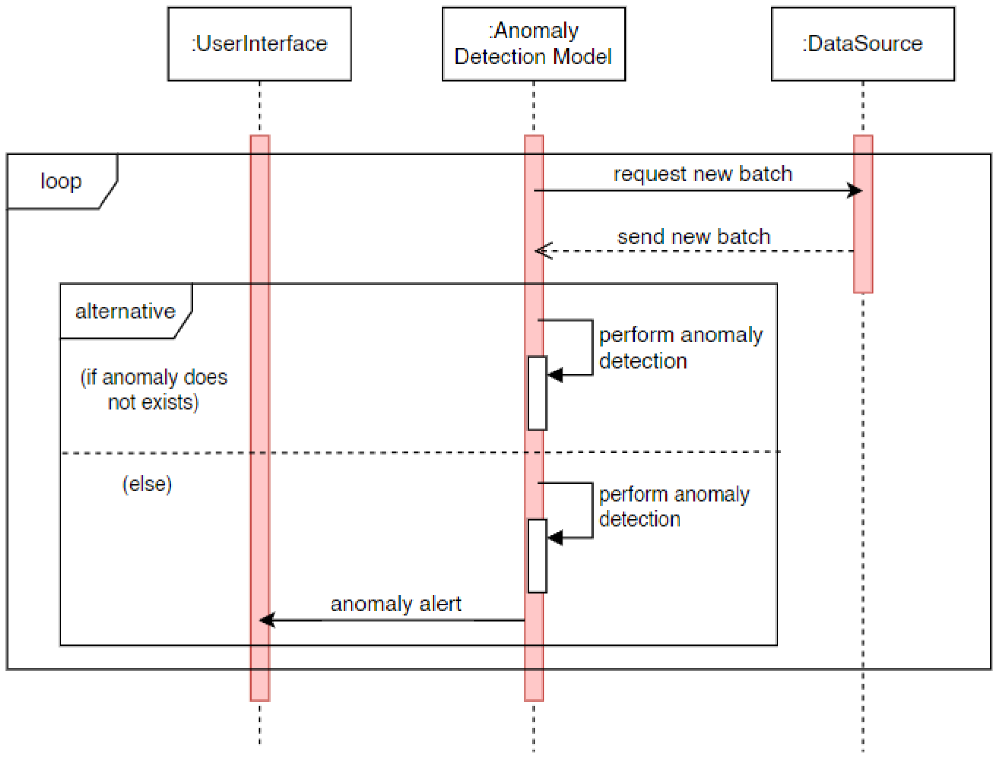
2.1.3 Sequence Diagram 3:
Diagram shows the sequence of interactions between user interface, system and data source when the user searches for an event by its name. System searches it in the data source and returns back the events that are named equal to the name searched.



2.1.4 Sequence Diagram 4:
Diagram shows the sequence of interactions between anomaly detection model and data source during model training phase. Data source is requested for train data for the model and it is sent back to the model. This interaction occurs in loop.



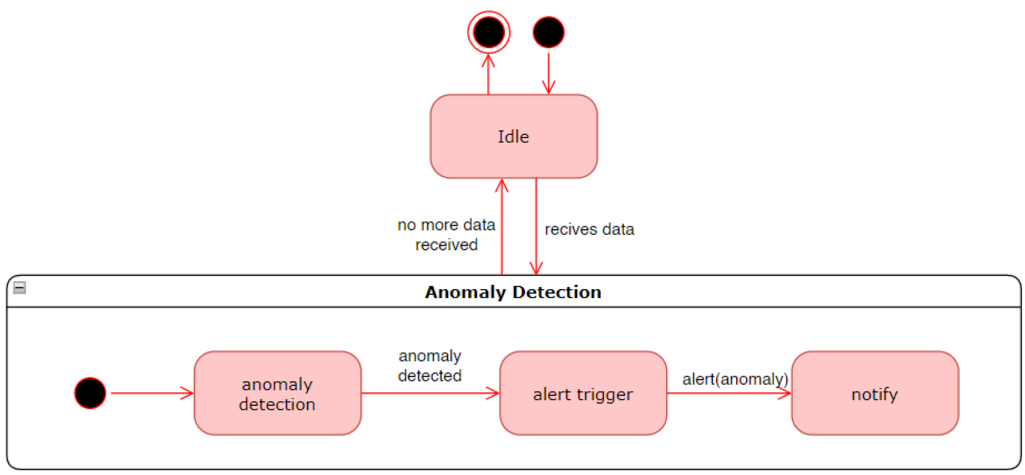
2.1.5 Sequence Diagram 5:
Diagram shows the sequence of interactions between the user interface, anomaly detection model and data
source during the anomaly detection phase. Anomaly Detection model requests a batch of logs from data source to detect anomalies. Performs anomaly detection on the logs from the data source. This process occurs in a loop. If anomaly is detected, then an alert is sent to user interface



2.2 State Diagram:

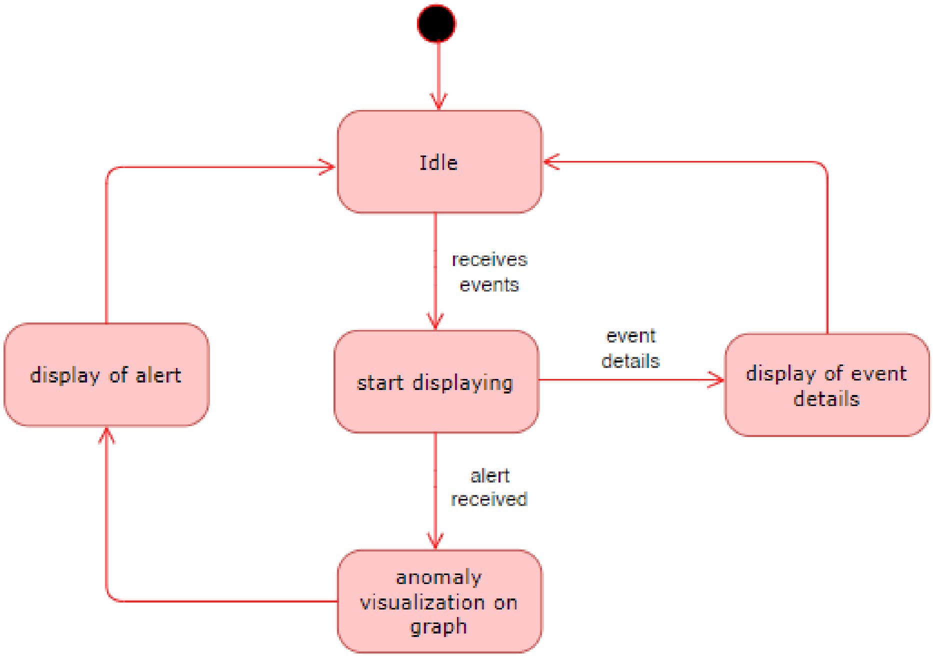
2.1 State Diagram 1:

Diagram shows different states of the system during the anomaly detection process. System is idle when there is no data to analyze. Once the data is received, it goes under anomaly detection state. When an anomaly is detected, an alert trigger state is started.



2.2 State Diagram 2:

Diagram below shows the states of the visualization tool during anomaly detection.


Idle
receives
events
display of alert
start displaying
event
details
display of event
details
alert
received
anomaly
visualization on
graph

Implementation

Anomaly Detection Machine Learning:

Preprocessing:

jupyter FYP_Preprocessing Last Checkpoint: yesterday

```
[1]: import numpy as np
     import pandas as pd
```

```
[2]: def preprocess_data(data):

       columns_to_drop = ['PT_NAME','PT_AGE', 'PT_GENDER', 'PT_ADMITBY']
       data.drop(columns=columns_to_drop, inplace=True)

       data.dropna(subset = ['PT_ADMISSION', 'PT_DISCHARGE', 'PT_BILLDATE', 'PT_ID', 'PT_CHARGESAMOUNT', 'PT_ADVANCEAMOUNT'], inplace = True, axis = 0)

       data['PT_ADMISSION'] = pd.to_datetime(data['PT_ADMISSION'], errors='coerce')
       data['PT_ADMISSION'] = (data['PT_ADMISSION'] - pd.Timestamp("1970-01-01")) // pd.Timedelta(seconds=1)

       data['PT_BILLDATE'] = pd.to_datetime(data['PT_BILLDATE'], errors='coerce')
       data['PT_BILLDATE'] = (data['PT_BILLDATE'] - pd.Timestamp("1970-01-01")) // pd.Timedelta(seconds=1)

       data['PT_DISCHARGE'] = pd.to_datetime(data['PT_DISCHARGE'], errors='coerce')
       data['PT_DISCHARGE'] = (data['PT_DISCHARGE'] - pd.Timestamp("1970-01-01")) // pd.Timedelta(seconds=1)

       data.reset_index(drop = True, inplace = True)
       data = data.dropna(how = 'any')
       return data
```

Model:

jupyter FYP_Model Last Checkpoint: yesterday

```
[11]: import pandas as pd
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import classification_report
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sns
      from imblearn.over_sampling import SMOTE
```

```
[12]: def load_model():
        df = pd.read_csv('labeled_data.csv')

        df['Anomaly'] = df['Anomaly'].astype(float)

        X = df.drop(['Anomaly'], axis=1)
        y =df['Anomaly']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
        smote = SMOTE(random_state=42) #for imbalanced data
        X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

        model = DecisionTreeClassifier (max_depth=18, random_state=42) #applied pruning
        model.fit(X_train_balanced, y_train_balanced)
        return model
```

```
[13]: def predict_anomlay(model, preprocessed_data):
        prediction = model.predict(preprocessed_data)
        preprocessed_data['Anomaly'] = prediction
        return preprocessed_data
```

Pipeline:

jupyter FYP_Pipeline Last Checkpoint: yesterday

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```python
import pandas as pd
```

[15]:
```python
%run "FYP_Preprocessing.ipynb"

#taking input log entry
pt_id, pt_name, pt_age, pt_gender, pt_admission, pt_admitby, pt_discharge, pt_chargesamount, pt_advanceamount, pt_billdate = input().split(',')

input_data_entry = {
    'PT_ID': pt_id,
    'PT_NAME': pt_name,
    'PT_AGE': pt_age,
    'PT_GENDER': pt_gender,
    'PT_ADMISSION': pt_admission,
    'PT_ADMITBY': pt_admitby,
    'PT_DISCHARGE': pt_discharge,
    'PT_CHARGESAMOUNT': pt_chargesamount,
    'PT_ADVANCEAMOUNT': pt_advanceamount,
    'PT_BILLDATE': pt_billdate
}

input_data = pd.DataFrame(input_data_entry, index=[0])

#preprocessing log entry
preprocessed_data = preprocess_data(input_data)

%run "FYP_Model.ipynb"

# Loading the trained model
model = load_model()

# Predicting anomalies for the input data entry
predicted_data_entry = predict_anomlay(model, preprocessed_data)
print(predicted_data_entry)
```

```python
from pymodm import connect, MongoModel, fields

# Connect to MongoDB
connect('mongodb://localhost:27017/mydb')

# Define a PyMODM model
class PredictedDataEntry(MongoModel):
    PT_ID = fields.IntegerField()
    PT_ADMISSION = fields.IntegerField()
    PT_DISCHARGE = fields.IntegerField()
    PT_CHARGESAMOUNT = fields.IntegerField()
    PT_ADVANCEAMOUNT = fields.IntegerField()
    PT_BILLDATE = fields.IntegerField()
    Anomaly = fields.IntegerField()

    class Meta:
        collection_name = 'PredictedData'  # Specify the collection name

# Create an instance of the model with the predicted data entry
predicted_entry = PredictedDataEntry(**predicted_data_entry)

# Save the instance to MongoDB
predicted_entry.save()

# Print the ID of the saved document
print("Data entry saved with ID:", predicted_entry._id)
```

Backend:

User.js:

User.js

server > models > User.js > userSchema.pre('save') callback

```javascript
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');

// Define the user schema
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

// Middleware to hash the password before saving
userSchema.pre('save', async function(next) {
  const user = this;
  if (!user.isModified('password')) {
    return next();
  }

  try {
    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(user.password, salt);
    user.password = hash;
    next();
  } catch (error) {
    return next(error);
  }
});

// Method to compare password during login
userSchema.methods.comparePassword = async function(candidatePassword) {
  try {
    return await bcrypt.compare(candidatePassword, this.password);
  } catch (error) {
    throw new Error(error);
  }
};

// Create the User model
const User = mongoose.model('users', userSchema);

module.exports = User;
```

Predictions.js:

DataTable.js | DataTable.css client\... | Predictions.js | Signup.js | Main.css | App.css | App.js client\... | Dashboard.js | Home.js | Log.js client\... | data.json client\...

server > models > Predictions.js > ...

```javascript
const mongoose = require('mongoose');

const predictionsSchema = new mongoose.Schema({
  id: { type: Number, required: true, unique: true },
  PT_ADMISSION: { type: Number, required: true },
  PT_DISCHARGE: { type: Number, required: true },
  PT_CHARGESAMOUNT: { type: Number, required: true },
  PT_ADVANCEAMOUNT: { type: Number, required: true },
  PT_BILLDATE: { type: Number, required: true },
  Anomaly: { type: String, required: true }
});

const Predictions = mongoose.model('predictions', predictionsSchema);

module.exports = Predictions;
```

Server.js:

server.js

server > server.js > ...

```javascript
const express = require('express');
const app = express();
const cors = require('cors');
const User = require('./models/User');
const Predictions = require('./models/Predictions');
const nodemailer = require('nodemailer');

const http = require('http');
const WebSocket = require('ws');
const mongoose = require('mongoose');

const PORT = 5000;

mongoose.connect('mongodb://localhost:27017/my_database', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});

app.use(express.json()); // Parse JSON-encoded bodies
app.use(cors());

// Define a Mongoose model for users

// Define a route to fetch all users
app.get('/api/users', async (req, res) => {
  try {
    // Fetch all users from the database
    const users = await User.find({});
    res.json(users); // Send users as a JSON response
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});


app.get('/api/predictions', async (req, res) => {
  try {
    // Fetch all predictions from the database
    const predictions = await Predictions.find({});
    res.json(predictions); // Send predictions as a JSON response
  } catch (error) {
    res.status(500).json({ message: error.message });
```

```javascript
wss.on('connection', (ws) => {
  changeStream.on('change', (change) => {
  });


});



app.post('/api/login', async (req, res) => {
  try {
    // Extract username and password from request body
    const { username, password } = req.body;


    // Check if the username exists
    const user = await User.findOne({ username });
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }

    // Check if the password is correct
    const isPasswordValid = await user.comparePassword(password);
    if (!isPasswordValid) {
      return res.status(401).json({ error: 'Invalid password' });
    }

    // If username and password are correct, send a success response
    res.status(200).json({ message: 'User logged in successfully' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Internal server error' });
  }
});


app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

```javascript
app.post('/api/Email', async (req, res) => {

    // Setup email data
    const mailOptions = {
      from: 'anomexplorer@outlook.com', // Your email address
      to: email,
      subject: 'Anomaly Detected',
      text: message,
    };

    // Send email
    const info = await transporter.sendMail(mailOptions);
    console.log('Email sent:', info.response);
    res.send('Email sent successfully');
  } catch (error) {
    console.error('Error sending email:', error);
    res.status(500).send('Failed to send email');
  }
});



const wss = new WebSocket.Server({ port: 5001 }); // WebSocket server port

// WebSocket connection handler
wss.on('connection', (ws) => {
  console.log('WebSocket client connected');

  // MongoDB change stream setup (assuming you're using Mongoose)
  const changeStream = Predictions.watch();

  // Listen for changes in the MongoDB collection
  changeStream.on('change', (change) => {
    // Send the updated record to the frontend
    ws.send(JSON.stringify(change));
    console.log('Change detected:', change);

  });


});



app.post('/api/login', async (req, res) => {
  try {
    // Extract username and password from request body
```

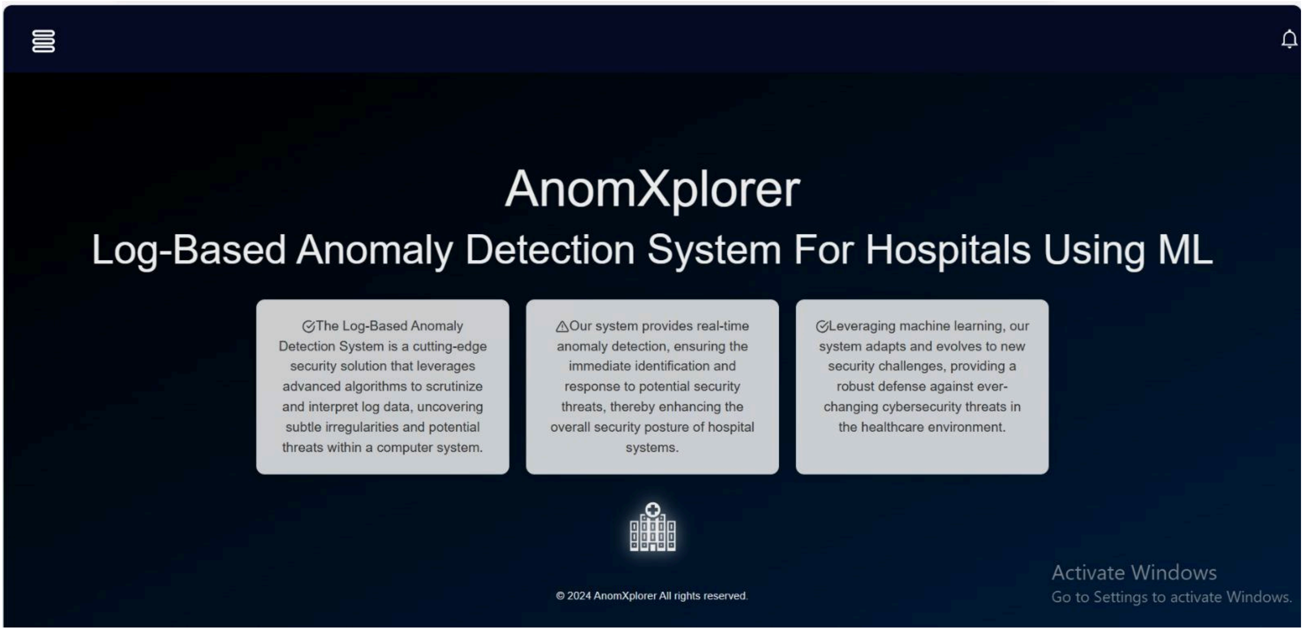
```
(node:18348) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Server is running on port 5000
Connected to MongoDB
```

```javascript


app.post('/api/signup', async (req, res) => {
  try {
    const { username, password } = req.body;

    const existingUser = await User.findOne({ username });
    if (existingUser) {
      return res.status(400).json({ error: 'Username already exists' });
    }


    const newUser = new User({ username, password });
    await newUser.save();

    res.status(201).json({ message: 'User signed up successfully' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Internal server error' });
  }
});

app.post('/api/Email', async (req, res) => {
  const { email, message } = req.body;

  try {
    // Create a transporter using SMTP
    const transporter = nodemailer.createTransport({
      host:'smtp-mail.outlook.com', // Your SMTP host
      //secureConnection: false,
      port: 587, // Your SMTP port
      secure: false, // Use TLS
      auth: {
        user: 'anomexplorer@outlook.com', // Your SMTP username
        pass: 'Fastnuces:)', // Your SMTP password
      },
      tls: {
        // Specify the minimum and maximum SSL/TLS protocol versions
        // Use 'TLSv1.2' as a secure protocol version
        minVersion: 'TLSv1.2',
        maxVersion: 'TLSv1.3', // You can adjust this as needed
      },
    });

    // Setup email data
    const mailOptions = {
      from: 'anomexplorer@outlook.com', // Your email address
      to: email,
```

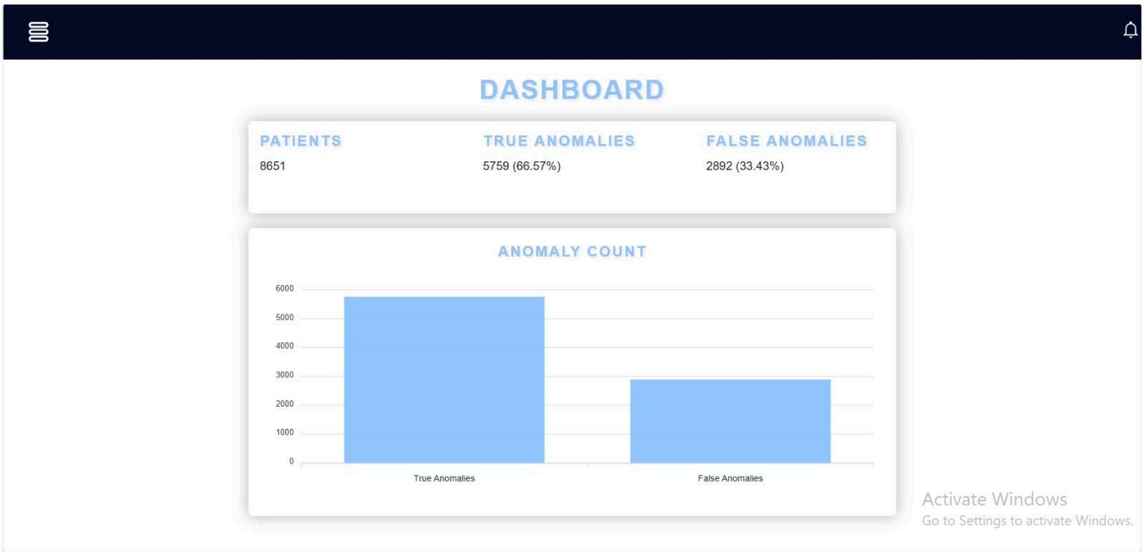
Frontend:

LoginPage:

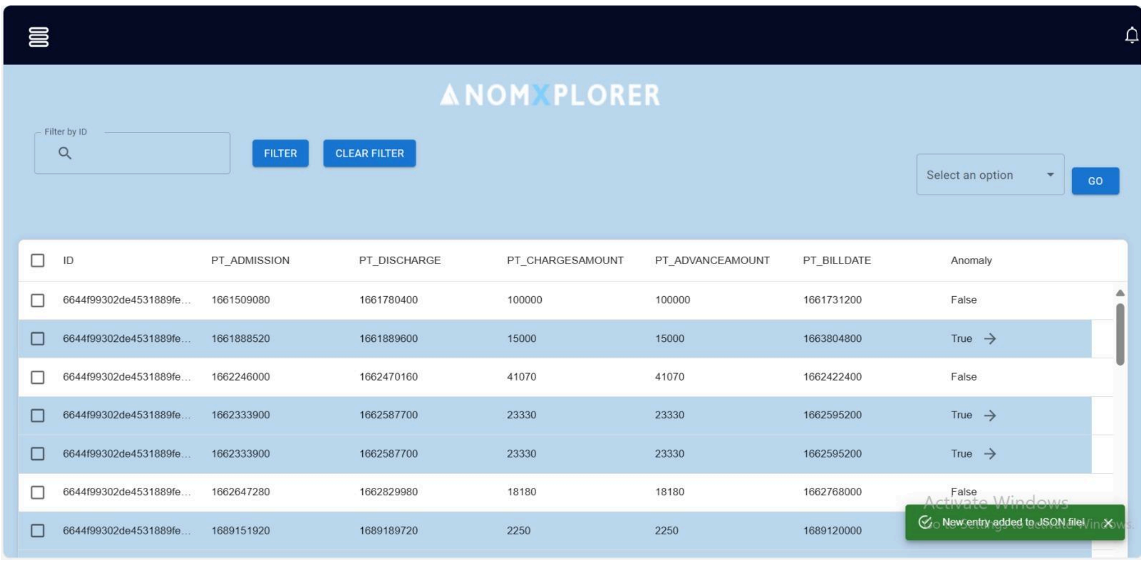
HomePage:



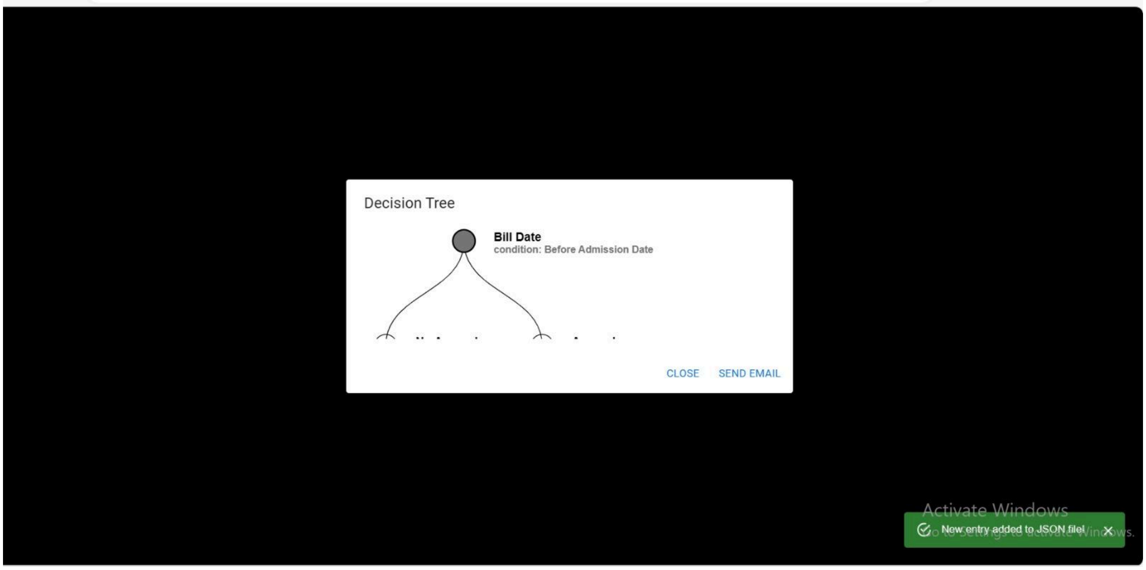
Dashboard:



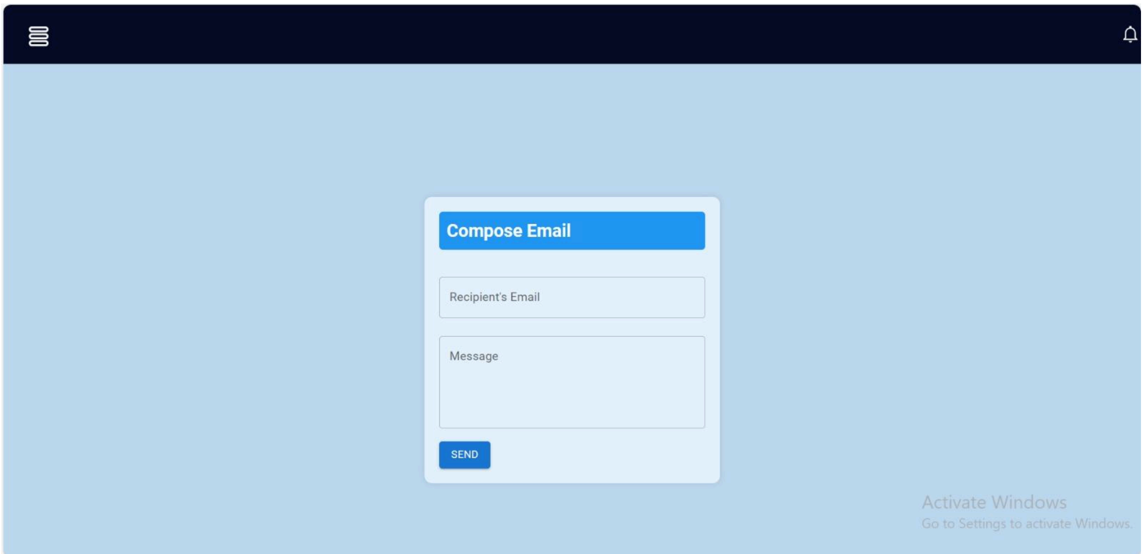
Logs:



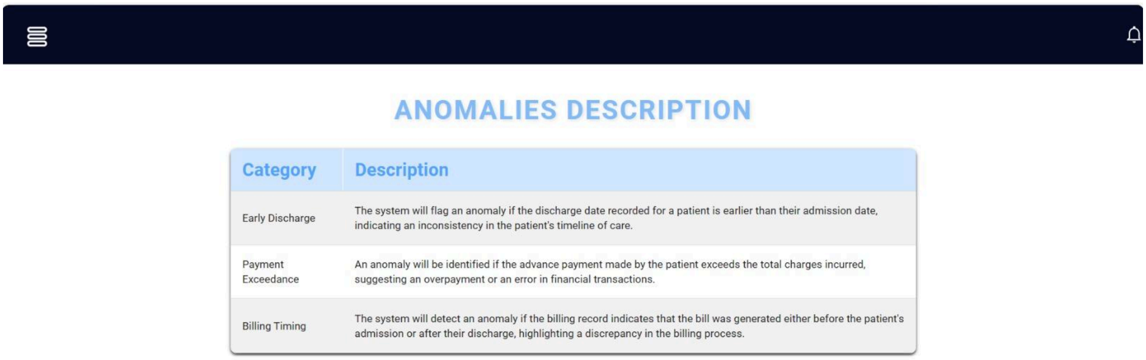
Anomaly graph visualization:



Send Alert:



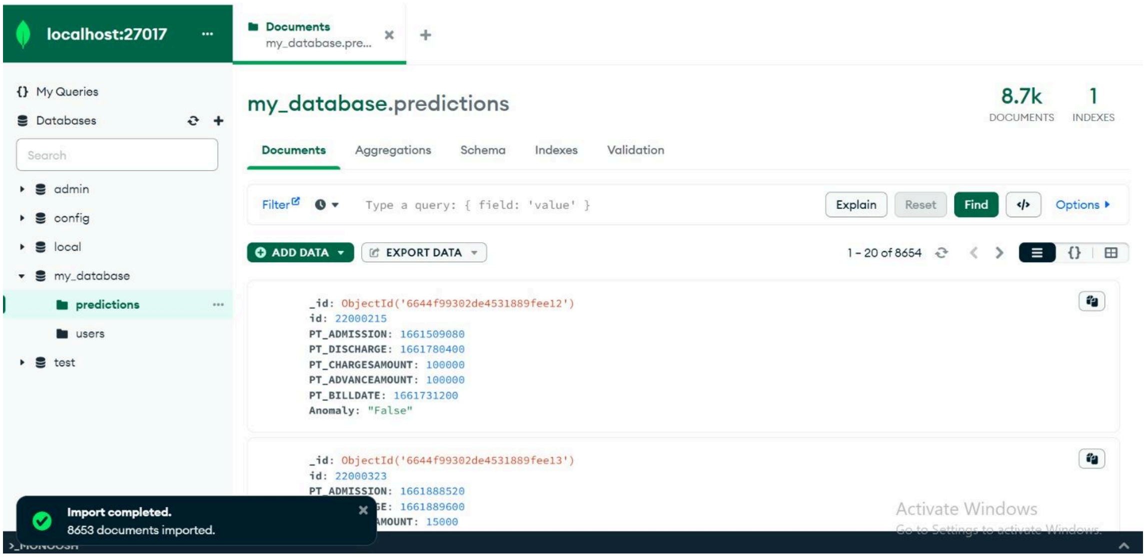
Anomaly descriptions:

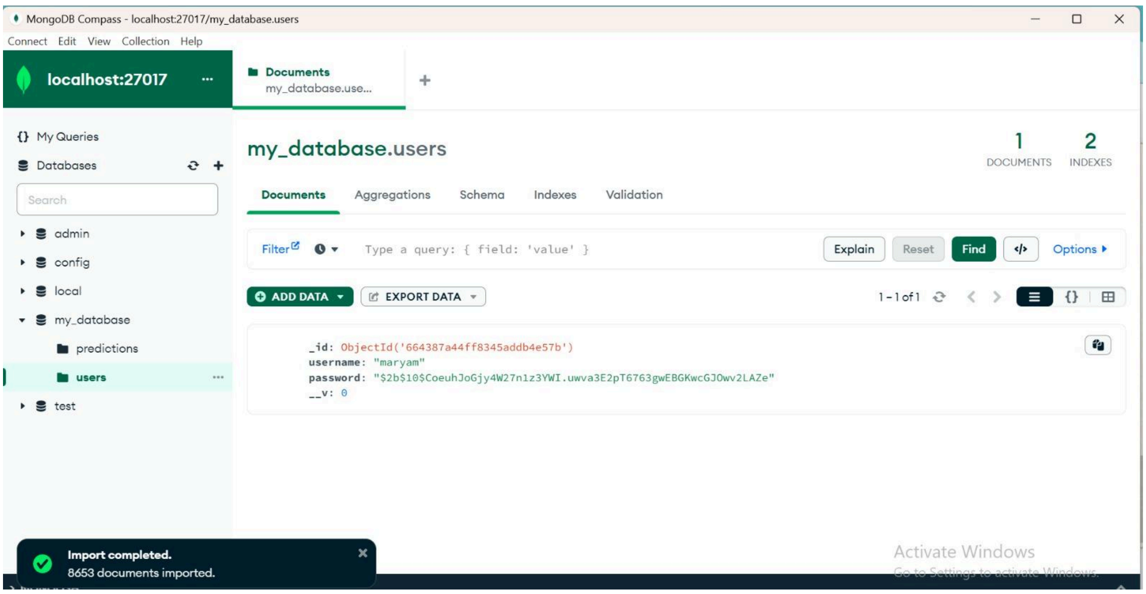


ANOMALIES DESCRIPTION

Category	Description
Early Discharge	The system will flag an anomaly if the discharge date recorded for a patient is earlier than their admission date, indicating an inconsistency in the patient's timeline of care.
Payment Exceedance	An anomaly will be identified if the advance payment made by the patient exceeds the total charges incurred, suggesting an overpayment or an error in financial transactions.
Billing Timing	The system will detect an anomaly if the billing record indicates that the bill was generated either before the patient's admission or after their discharge, highlighting a discrepancy in the billing process.

DataBase:





Testing and Evaluation

Purpose:

The testing and evaluation phase of our anomaly detection system serves several critical purposes:

- Validate the effectiveness and accuracy of the anomaly detection algorithm in identifying anomalies within hospital data.
- Assess the performance and reliability of the system under varying conditions and data inputs.
- Ensure the system meets the specified requirements and objectives outlined during the design and development phase.
- Identify any potential issues, limitations, or areas for improvement in the system's functionality or performance.

Environmental Needs:

- The system will be designed to operate on Windows operating systems commonly used in healthcare IT environments.
- A stable internet connection is required.
- Only the English language is supported.

Validation Testing:

Validation testing involves assessing the accuracy, reliability, and performance of the anomaly detection system under various scenarios and input conditions.

This include:

- Input Data Validation: Ensuring systems can process and analyze different types of hospital data, including patient records, bill information, discharge and admission information, without errors or data loss.

- Anomaly detection Accuracy: Assessing the model's ability to accurately identify anomalies within the input data, including billing discrepancies, admission and discharge date inconsistencies.
- Robustness Testing: Evaluating the system's resilience to noise and outliers.

Test Cases:

TEST CASE ID: TC1 TEST CASE NAME: SIGN IN				
No.	STEPS	EXPECTED RESULTS	ACTUAL RESULTS	PASS/FAIL
1.	Input valid login credentials	User should be able to login	User Logged In	Pass
2.	Input invalid login credentials	User should not be able to login	User login again	Pass
3.	Input Password	Password encrypted to the user	password is encrypted	Pass
4.	Input email with invalid format	Invalid email format message should appear	Invalid email format message appears	Pass

TEST CASE ID: TC2 TEST CASE NAME: Anomaly Detection Accuracy				
No.	STEPS	EXPECTED RESULTS	ACTUAL RESULTS	PASS/FAIL
1.	Provide dataset with known anomalies and normal data points	Anomalies should be correctly identified and classified	Anomalies correctly identified and classified.	Pass

TEST CASE ID: TC3 TEST CASE NAME: Real Time Detection				
No.	STEPS	EXPECTED RESULTS	ACTUAL RESULTS	PASS/FAIL
1.	Input real time data containing anomalies	Anomalies should correctly identified	Anomalies correctly identified	Pass

TEST CASE ID: TC4 TEST CASE NAME: Notification System				
No.	STEPS	EXPECTED RESULTS	ACTUAL RESULTS	PASS/FAIL

1.	Input new data	Notification should pop up if enter new data	Notification popped up	Pass

TEST CASE ID: TC5 TEST CASE NAME: Send Email				
No.	STEPS	EXPECTED RESULTS	ACTUAL RESULTS	PASS/FAIL
1.	Input new data with anomalies .	To resolve anomalies, users should be able to send email.	Email sent	Pass

TEST CASE ID: TC6 TEST CASE NAME: Dashboard Functionality				
No.	STEPS	EXPECTED RESULTS	ACTUAL RESULTS	PASS/FAIL
1.	Input new entry.	Dashboard' s graphs should be updated	Dashboard updated	Pass

Conclusion

In conclusion, we have developed a comprehensive anomaly detection system tailored for hospital environments, leveraging machine learning techniques and user-friendly web application interface. By providing real-time analysis and visualization of hospital data, our system empowers administrators to identify and address potential issues proactively, ultimately enhancing patient care and operational efficiency. Moving forward, further refinements and enhancements will be pursued to ensure the continued effectiveness and relevance of our system in addressing the evolving needs of healthcare settings.

References

[1] Chen, M., Zheng, A. X., Lloyd, J., Jordan, M. I., & Brewer, E. (2021). Failure diagnosis using decision trees. In International Conference on Autonomic Computing, 2021. Proceedings (pp.36-43). IEEE.

[2] He, S., Zhu, J., He, P., & Lyu, M. R. (2020). Experience Report: System Log Analysis for Anomaly Detection. 2020 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). doi:10.1109/issre.2016.21

[3] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M. R. (2019). Tools and Benchmarks for Automated Log Parsing. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)