**Project Report: Insurance Management System**

**1. Project Overview**

**Project Title:** Insurance Management System

**Objective:**  
To develop a comprehensive Insurance Management System that allows users to manage policies and claims efficiently. This system includes functionalities for adding new policies, filing claims, and viewing both policies and claims. The system also includes a user-friendly web interface for interaction.

**2. Features**

1. **Dashboard:**
   * Displays navigation options for managing policies and claims.
2. **Add Policy:**
   * Form to input new policy details, including policy name, type, and premium amount.
3. **View Policies:**
   * Table displaying all the policies added by the user.
4. **Add Claim:**
   * Form to file new claims against existing policies, including policy ID, claim date, and claim amount.
5. **View Claims:**
   * Table displaying all claims made, including policy name, claim date, claim amount, and status.
6. **Logout:**
   * Option to log out from the system.

**3. E.R Diagram**

Creating an ER (Entity-Relationship) diagram for your project involves defining the main entities, their attributes, and relationships. Based on the functionality described in your project (insurance management system with policies and claims), here's a conceptual ER diagram outline:

**Entities and Attributes**

1. **Policy**
   * **PolicyID** (Primary Key)
   * **PolicyName**
   * **PolicyType**
   * **PremiumAmount**
2. **Claim**
   * **ClaimID** (Primary Key)
   * **PolicyID** (Foreign Key, references Policy.PolicyID)
   * **ClaimDate**
   * **ClaimAmount**
   * **Status**

**Relationships**

* **Policy to Claim**: One-to-Many
  + **One Policy** can have **Many Claims**.
  + Each **Claim** is associated with exactly **One Policy**.

**ER Diagram**

Here’s a textual representation of how you might structure your ER diagram:

+----------------+ +-----------------+

| Policy | | Claim |

+----------------+ +-----------------+

| PolicyID (PK) | 1 \* | ClaimID (PK) |

| PolicyName |-------| PolicyID (FK) |

| PolicyType | | ClaimDate |

| PremiumAmount | | ClaimAmount |

+----------------+ | Status |

+-----------------+

**Detailed Steps to Create the Diagram**

1. **Identify Entities**: List out all major entities involved (Policy and Claim).
2. **Define Attributes**: For each entity, list the attributes. Mark the primary key (PK) and foreign key (FK).
3. **Determine Relationships**: Define the type of relationships between entities. For example, a Policy can have multiple Claims, but each Claim is related to only one Policy.
4. **Use a Tool**: To visually create the ER diagram, you can use tools like:
   * **Draw.io** (now diagrams.net)
   * **Lucidchart**
   * **Microsoft Visio**
   * **MySQL Workbench** (if you’re working with MySQL databases)
   * **ERDPlus** (an online tool)

**Example Using Draw.io**

1. Go to Draw.io.
2. Start a new diagram.
3. Use the shapes for entities (rectangles) and relationships (diamonds or lines).
4. Label the entities, attributes, and relationships according to your project's structure.
5. Draw lines to represent relationships and use crow's foot notation to indicate cardinality (one-to-many).

**Sample ER Diagram**

Here’s a simple example created using basic shapes:

Policy (PolicyID, PolicyName, PolicyType, PremiumAmount)

|

| 1

|

| \*

|

Claim (ClaimID, PolicyID, ClaimDate, ClaimAmount, Status)

In this ER diagram:

* **PolicyID** is the primary key of the Policy table and is used as a foreign key in the Claim table.
* The relationship between Policy and Claim is a one-to-many relationship (one Policy can have many Claims).

**4. DFD**

A Data Flow Diagram (DFD) is a graphical representation of the flow of data through a system, showing how input is processed to produce output. It is used to visualize the data processing steps and the interactions between different components of a system.

**Key Components of a DFD**

1. **Processes**: Represent the operations that transform data inputs into outputs (e.g., Add Policy, View Policies, File a Claim).
2. **Data Flows**: Arrows that show the direction of data movement between processes, data stores, and external entities.
3. **Data Stores**: Repositories where data is stored (e.g., Policy Database, Claim Database).
4. **External Entities**: Sources or destinations of data outside the system (e.g., Users, Administrators).

**DFD Levels**

1. **Context Diagram (Level 0)**: Shows the system as a single process and its interactions with external entities.
2. **Level 1 DFD**: Breaks down the single process from the context diagram into major sub-processes and data flows.
3. **Level 2 DFD (and beyond)**: Further decomposes Level 1 processes into more detailed processes and data flows.

**DFD for Insurance Management System**

**Context Diagram (Level 0)**

**External Entities:**

* **User**: Interacts with the system to add policies, file claims, view policies, and view claims.
* **Administrator**: May also manage policies and claims (if applicable).

**Process:**

* **Insurance Management System**: Handles all operations related to policies and claims.

**Data Flows:**

* **User Inputs**: Add Policy, File Claim
* **System Outputs**: Policy Details, Claim Status
* **Data Exchange**: Policies, Claims

**Diagram Example:**

+-------------------------+ +------------------------------+

| User | | Insurance Management System |

|-------------------------| |------------------------------|

| - Add Policy |<----->| - Manage Policies |

| - File Claim | | - Manage Claims |

| - View Policies | | |

| - View Claims | | |

+-------------------------+ +------------------------------+

^

|

v

+-------------------+

| Policy Database |

+-------------------+

+-------------------+

| Claim Database |

+-------------------+

**Level 1 DFD**

**Processes:**

1. **Manage Policies**: Handles adding, updating, and retrieving policies.
2. **Manage Claims**: Handles filing, updating, and retrieving claims.

**Data Stores:**

* **Policy Database**: Stores policy information.
* **Claim Database**: Stores claim information.

**Data Flows:**

* **User** to **Manage Policies**: Policy Details
* **Manage Policies** to **Policy Database**: Store Policy
* **Policy Database** to **Manage Policies**: Retrieve Policy
* **User** to **Manage Claims**: Claim Details
* **Manage Claims** to **Claim Database**: Store Claim
* **Claim Database** to **Manage Claims**: Retrieve Claim

**Diagram Example:**

+------------------+ +------------------+ +------------------+

| User | | Manage Policies | | Policy Database |

|------------------| |------------------| |------------------|

| - Add Policy |-------->| - Add Policy |-------->| - Store Policy |

| - View Policies |<--------| - View Policies |<-------->| - Retrieve Policy|

+------------------+ +------------------+ +------------------+

|

|

v

+------------------+ +------------------+

| Manage Claims | | Claim Database |

|------------------| |------------------|

| - File Claim |-------->| - Store Claim |

| - View Claims |<-------->| - Retrieve Claim |

+------------------+ +------------------+

**Tools for Creating DFDs**

* **Draw.io (diagrams.net)**: A free, web-based tool for creating various types of diagrams, including DFDs.
* **Lucidchart**: A cloud-based diagram tool with DFD templates.
* **Microsoft Visio**: A diagramming tool with support for creating DFDs.
* **Creately**: An online diagramming tool with DFD capabilities.

To visualize and create these diagrams, you can use one of the tools mentioned above or draw them manually using standard shapes and symbols for processes, data stores, and data flows.

**5.Technical Design**

**Technologies Used:**

* **Python:** For backend logic and handling HTTP requests.
* **Flask:** Lightweight web framework to develop the web application.
* **Bootstrap:** CSS framework for responsive and modern front-end design.
* **Font Awesome:** For icons used in the interface.

**Project Structure:**

* **app.py**: Main Python file that contains Flask routes and logic.
* **templates/**: Directory containing HTML templates for different pages.
  + **layout.html**: Base template with header, sidebar, and footer.
  + **add\_policy.html**: Page for adding new policies.
  + **view\_policies.html**: Page for viewing all policies.
  + **add\_claim.html**: Page for adding new claims.
  + **view\_claims.html**: Page for viewing all claims.
* **static/**: Directory containing static files like CSS, JavaScript, and images.

**4. Implementation**

**4.1. Flask Application (app.py)**

from flask import Flask, render\_template, request, redirect, url\_for, session

import sqlite3

app = Flask(\_\_name\_\_)

app.secret\_key = 'your\_secret\_key'

# Database setup

def init\_db():

with sqlite3.connect('insurance.db') as conn:

cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS policies (

id INTEGER PRIMARY KEY AUTOINCREMENT,

policy\_name TEXT,

policy\_type TEXT,

premium\_amount REAL)''')

cursor.execute('''CREATE TABLE IF NOT EXISTS claims (

id INTEGER PRIMARY KEY AUTOINCREMENT,

policy\_id INTEGER,

claim\_date TEXT,

claim\_amount REAL,

status TEXT,

FOREIGN KEY (policy\_id) REFERENCES policies (id))''')

@app.route('/')

def index():

return render\_template('dashboard.html')

@app.route('/add\_policy', methods=['GET', 'POST'])

def add\_policy():

if request.method == 'POST':

policy\_name = request.form['policy\_name']

policy\_type = request.form['policy\_type']

premium\_amount = request.form['premium\_amount']

with sqlite3.connect('insurance.db') as conn:

cursor = conn.cursor()

cursor.execute('INSERT INTO policies (policy\_name, policy\_type, premium\_amount) VALUES (?, ?, ?)',

(policy\_name, policy\_type, premium\_amount))

conn.commit()

return redirect(url\_for('view\_policies'))

return render\_template('add\_policy.html')

@app.route('/view\_policies')

def view\_policies():

with sqlite3.connect('insurance.db') as conn:

cursor = conn.cursor()

cursor.execute('SELECT \* FROM policies')

policies = cursor.fetchall()

return render\_template('view\_policies.html', policies=policies)

@app.route('/add\_claim', methods=['GET', 'POST'])

def add\_claim():

if request.method == 'POST':

policy\_id = request.form['policy\_id']

claim\_date = request.form['claim\_date']

claim\_amount = request.form['claim\_amount']

with sqlite3.connect('insurance.db') as conn:

cursor = conn.cursor()

cursor.execute('INSERT INTO claims (policy\_id, claim\_date, claim\_amount, status) VALUES (?, ?, ?, ?)',

(policy\_id, claim\_date, claim\_amount, 'Pending'))

conn.commit()

return redirect(url\_for('view\_claims'))

return render\_template('add\_claim.html')

@app.route('/view\_claims')

def view\_claims():

with sqlite3.connect('insurance.db') as conn:

cursor = conn.cursor()

cursor.execute('''SELECT p.policy\_name, c.claim\_date, c.claim\_amount, c.status

FROM claims c

JOIN policies p ON c.policy\_id = p.id''')

claims = cursor.fetchall()

return render\_template('view\_claims.html', claims=claims)

@app.route('/logout')

def logout():

session.pop('user', None)

return redirect(url\_for('index'))

if \_\_name\_\_ == '\_\_main\_\_':

init\_db()

app.run(debug=True)

**4.2. HTML Templates**

**Base Template (layout.html):**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css">

<style>

body {

margin-top: 70px;

}

.header {

background-color: #007bff;

color: white;

padding: 15px 20px;

position: fixed;

width: 100%;

top: 0;

z-index: 1000;

}

.footer {

background-color: #343a40;

color: white;

text-align: center;

padding: 10px 20px;

position: fixed;

width: 100%;

bottom: 0;

}

.sidebar {

position: fixed;

top: 70px;

left: 0;

width: 200px;

height: calc(100vh - 140px);

background-color: #f8f9fa;

padding: 15px;

border-right: 1px solid #ddd;

}

.content {

margin-left: 220px;

padding: 20px;

}

</style>

<title>{% block title %}Insurance Management System{% endblock %}</title>

</head>

<body>

<div class="header">

<h1>Insurance Dashboard</h1>

<a href="{{ url\_for('logout') }}" class="btn btn-danger float-right">Logout</a>

</div>

<div class="sidebar">

<h4>Dashboard</h4>

<a href="/add\_policy" class="btn btn-light btn-block"><i class="fas fa-plus-circle"></i> Add Policy</a>

<a href="/view\_policies" class="btn btn-light btn-block"><i class="fas fa-eye"></i> View Policies</a>

<a href="/add\_claim" class="btn btn-light btn-block"><i class="fas fa-clipboard-list"></i> File a Claim</a>

<a href="/view\_claims" class="btn btn-light btn-block"><i class="fas fa-list"></i> View Claims</a>

</div>

<div class="content">

{% block content %}{% endblock %}

</div>

<div class="footer">

&copy; 2024 Insurance Dashboard. All Rights Reserved.

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.0.6/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**Add Policy Page (add\_policy.html):**

{% extends "layout.html" %}

{% block title %}Add Policy{% endblock %}

{% block content %}

<div class="container">

<h2>Add New Policy</h2>

<form method="POST" action="/add\_policy">

<div class="form-group">

<label for="policy\_name">Policy Name</label>

<input type="text" class="form-control" id="policy\_name" name="policy\_name" required>

</div>

<div class="form-group">

<label for="policy\_type">Policy Type</label>

<input type="text" class="form-control" id="policy\_type" name="policy\_type" required>

</div>

<div class="form-group">

<label for="premium\_amount">Premium Amount</label>

<input type="number" class="form-control" id="premium\_amount" name="premium\_amount" required>

</div>

<button type="submit" class="btn btn-primary">Add Policy</button>

<a href="/view\_policies" class="btn btn-secondary">Go Back</a>

</form>

</div>

{% endblock

**Add Claim Page (add\_claim.html):**

{% extends "layout.html" %}

{% block title %}Add Claim{% endblock %}

{% block content %}

<div class="container">

<h2>Add New Claim</h2>

<form method="POST" action="/add\_claim">

<div class="form-group">

<label for="policy\_id">Policy ID</label>

<input type="text" class="form-control" id="policy\_id" name="policy\_id" required>

</div>

<div class="form-group">

<label for="claim\_date">Claim Date</label>

<input type="date" class="form-control" id="claim\_date" name="claim\_date" required>

</div>

<div class="form-group">

<label for="claim\_amount">Claim Amount</label>

<input type="number" class="form-control" id="claim\_amount" name="claim\_amount" required>

</div>

<button type="submit" class="btn btn-primary">Submit Claim</button>

<a href="/view\_claims" class="btn btn-secondary">Go Back</a>

</form>

</div>

{% endblock %}

**View Policies Page (view\_policies.html):**

{% extends "layout.html" %}

{% block title %}View Policies{% endblock %}

{% block content %}

<div class="container">

<h2>Your Policies</h2>

<table class="table table-striped">

<thead>

<tr>

<th>Policy Name</th>

<th>Policy Type</th>

<th>Premium Amount</th>

</tr>

</thead>

<tbody>

{% for policy in policies %}

<tr>

<td>{{ policy[1] }}</td> <!-- policy\_name -->

<td>{{ policy[2] }}</td> <!-- policy\_type -->

<td>{{ policy[3] }}</td> <!-- premium\_amount -->

</tr>

{% endfor %}

</tbody>

</table>

<a href="/add\_policy" class="btn btn-primary">Add New Policy</a>

<a href="/" class="btn btn-secondary">Go Back</a>

</div>

{% endblock %}

**View Claims Page (view\_claims.html):**

{% extends "layout.html" %}

{% block title %}View Claims{% endblock %}

{% block content %}

<div class="container">

<h2>Your Claims</h2>

<table class="table table-striped">

<thead>

<tr>

<th>Policy Name</th>

<th>Claim Date</th>

<th>Claim Amount</th>

<th>Status</th>

</tr>

</thead>

<tbody>

{% for claim in claims %}

<tr>

<td>{{ claim[0] }}</td> <!-- policy\_name -->

<td>{{ claim[1] }}</td> <!-- claim\_date -->

<td>{{ claim[2] }}</td> <!-- claim\_amount -->

<td>{{ claim[3] }}</td> <!-- status -->

</tr>

{% endfor %}

</tbody>

</table>

<a href="/add\_claim" class="btn btn-primary">File New Claim</a>

<a href="/" class="btn btn-secondary">Go Back</a>

</div>

{% endblock %}

**5. Deployment**

1. **Local Setup:**
   * Install the necessary packages using pip:

pip install flask

* + Ensure SQLite is available, or use another preferred database.

1. **Running the Application:**
   * Start the Flask server by running:

python app.py

* + Access the application via http://127.0.0.1:5000/ in a web browser.

1. **Production Deployment:**
   * Consider using a WSGI server like Gunicorn for deploying Flask applications in production.
   * Host the application on a platform like Heroku, AWS, or any other cloud service provider.

**6. Conclusion**

The Insurance Management System is designed to streamline the management of insurance policies and claims with a user-friendly web interface. By utilizing Python and Flask, the application provides a robust backend solution with a responsive front-end layout powered by Bootstrap. This report outlines the primary features and technical details of the project, providing a clear guide for both development and deployment.