**ABSTRACT**

The **Data Visualization Web Application** is a **Flask-based interactive platform** that enables users to upload, store, and visualize datasets dynamically. The system integrates SQLite as a lightweight database, allowing users to manage datasets efficiently.

Users can select different visualization types (bar, line, pie charts) and configure parameters such as X-axis, Y-axis, colors, and grid settings. The generated charts can be exported in PNG or PDF formats.

**Key Features:**

* Dynamic Data Upload – Users can import Excel (.xlsx) files.
* Structured Storage – Data is stored in SQLite tables.
* Interactive Dashboard – Allows chart customization.
* Visualization Options – Supports bar, line, and pie charts.
* Export Capabilities – Charts can be saved as PNG/PDF.

**1. INTRODUCTION**

**1.1 About the Project**

The Data Visualization Web Application is designed to help users analyze and represent data in a structured format. The application is built using Python (Flask) for backend processing, SQLite for data management, and HTML/CSS/JavaScript for frontend interactions.

Users can:

* Upload Excel datasets to be stored in an SQLite database.
* Dynamically select table columns for chart generation.
* Customize charts (colors, grid settings, axis selection).
* Generate and view charts directly on the web dashboard.
* Export charts for further analysis or presentations.

**1.2 Objective**

The primary goal of this project is to provide a user-friendly data visualization platform that simplifies data processing and representation**.**

**Specific Objectives:**

* Enable automated data import from Excel files.
* Provide an interactive chart-building experience.
* Ensure structured storage for data in an SQLite database.
* Support multiple visualization types.
* Offer flexible export options for professional usage.

**2. SYSTEM STUDY**

**2.1 Existing System**

* Users manually generate charts in Excel.
* No centralized database for data storage.
* Time-consuming for large datasets.

**2.2 Disadvantages of Existing System**

* No automation in chart creation.
* Data redundancy due to manual operations.
* Lack of interactive web interface for dynamic visualization.

**2.3 Proposed System**

* Automated chart generation from SQLite database.
* Web-based UI with interactive dashboards.
* Seamless Excel data import with structured storage.

**2.4 Advantages of Proposed System**

* Faster data processing with SQLite.
* User-friendly dashboard with dynamic inputs.
* Flexible export options for charts.

.

**3. SYSTEM ANALYSIS**

**3.1 Technologies Used**

* **Front-end**: HTML, CSS, JavaScript (AJAX)
* **Back-end**: Python (Flask, Pandas, Matplotlib)
* **Database**: SQLite
* **Libraries Used**: Pandas, Matplotlib, Flask, Openpyxl

**3.2 Hardware Requirements**

* **Processor**: Intel Core i3 or higher
* **RAM**: 4GB minimum
* **Storage**: 500MB free space

**Software Requirements**

* Python 3.x
* Flask, SQLite3
* Pandas, Matplotlib

**3.3 Feasibility Study**

* **Technical Feasibility**: Uses lightweight frameworks and open-source technologies.
* **Operational Feasibility**: Easy to use with a simple web-based interface.

**3.4 Use Case Diagram**

Shows user interactions, including file upload, table selection, visualization, and export.

**3.5 Data Flow Diagram**

**4. SYSTEM DESIGN**

**4.1 Architectural Design**

The system follows a three-tier architecture, dividing functionalities into Presentation, Logic, and Data Layers.

**System Architecture Overview**

1. **Presentation Layer (Frontend)**
   * Developed using HTML, CSS, JavaScript.
   * Provides an interactive dashboard.
   * Handles user interactions (data upload, chart selection, export).
2. **Business Logic Layer (Backend)**
   * + Flask framework processes requests.
     + Handles data processing and API requests.
     + Manages chart generation and export functions.
3. **Data Layer (Database)**
   * + SQLite database stores uploaded datasets.
     + Enables structured data retrieval for visualization.

**System Architecture Diagram**

**4.3 Database Design**

The SQLite database stores the data extracted from uploaded Excel files. The system dynamically maps Excel columns to SQLite tables.

**Database Schema**

Each uploaded file corresponds to a table in SQLite (e.g., sheet1, sheet2, sheet3). The database schema is as follows:

**4.4 Tables**

**Table Name: sheet1**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| Year | INT | The academic year in which the course is offered. |
| Title\_of\_the\_Course | VARCHAR | The official name of the course. |
| Course\_Code | VARCHAR | A unique identifier or code assigned to the course. |
| Title\_of\_the\_Content | VARCHAR | The title of the specific content or module within the course. |
| Nature\_of\_e\_content | TEXT | A detailed description of the type and nature of the e-content |
| Department | TEXT | Indicates whether the course belongs to the "DR" or "J" |

**Table Name: sheet2**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| Year | INT | The year in which the talk or seminar took place. |
| Name\_of\_the\_Faculty | VARCHAR | The name of the faculty member delivering the talk. |
| Title\_of\_the\_Talk | VARCHAR | The title or topic of the talk presented by the faculty. |
| Nature | TEXT | The nature or format of the talk (e.g., lecture, panel discussion, workshop). |
| Type\_of\_Programme | TEXT | The type of program in which the talk was delivered |
| Title\_of\_the\_Seminar | VARCHAR | The title of the seminar or event where the talk was presented. |
| Institution\_and\_Place | VARCHAR | The name of the institution and location where the seminar was conducted. |
| Date | DATE | The specific date on which the talk took place. |

**Table Name: sheet3**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Year | INT | The year in which the talk or seminar took place. |
| Name\_of\_the\_Faculty | VARCHAR | The name of the faculty member delivering the talk. |
| Title\_of\_the\_Talk | VARCHAR | The title or topic of the talk presented by the faculty. |
| Nature | TEXT | The nature or format of the talk |
| Type\_of\_Programme | TEXT | The type of program in which the talk was delivered |
| Title\_of\_the\_Seminar | VARCHAR | The title of the seminar or event where the talk was presented. |
| Institution\_and\_Place | VARCHAR | The name of the institution and location where the seminar was conducted. |

**Entity Relationship Diagram (ERD)**

The ER Diagram represents the relationship between entities:

**5. CODING & DEBUGGING**

**5.1 Flask API Endpoints**

|  |  |  |
| --- | --- | --- |
| **Route** | **Method** | **Function** |
| /upload | POST | Uploads and stores Excel data in SQLite. |
| /get\_tables | GET | Fetches available tables from the database. |
| /get\_columns | POST | Retrieves column names for chart generation. |
| /generate\_chart | POST | Creates and displays charts based on user selection. |
| /export | POST | Exports charts to PNG/PDF. |

* 1. **Frontend (HTML + JavaScript)**

**User Interface**:

* Sidebar Navigation: Import, Category, Charts, Export.
* Dropdown selections for table, X-axis, Y-axis, and chart type.
* Popup messages for user alerts.

**5.3 JavaScript Functions**

* showContent(page): Displays the selected content section.
* populateTableDropdown(): Retrieves tables from SQLite.
* navigateToCharts(): Fetches data and generates charts dynamically.
* selectFolder(): Allows user to choose an export location.

**6. TESTING**

**6.1 Types of Testing**

* Unit Testing: Each Flask API route tested separately.
* Integration Testing: Ensured smooth interaction between SQLite, Flask, and Frontend.
* UI Testing: Checked dropdown selections, chart generation, and file exports.

|  |  |  |
| --- | --- | --- |
| **Test Scenario** | **Expected Outcome** | **Result** |
| Upload valid Excel file | Data stored in SQLite | Pass |
| Upload invalid file | Show error message | Pass |
| Generate Bar Chart | Chart displayed successfully | Pass |
| Export Chart as PDF | File saved in selected folder | Pass |

**6.2 Test Cases**

**7. USER MANUAL**

**7.1 How to Run the Project**

1. **Install dependencies:**

pip install flask pandas sqlite3 matplotlib openpyxl

1. **Run the Flask app**:

python app.py

1. **Access the dashboard**:

Open http://127.0.0.1:5000 in a browser.

**7.2 Features**

* Upload Excel files for data storage.
* Select columns dynamically for visualization.
* Generate charts (bar, line, pie).
* Export charts to PNG or PDF.

**7.3 Error Handling**

|  |  |  |
| --- | --- | --- |
| **Error** | **Cause** | **Solution** |
| "Invalid file format" | Uploaded file is not .xlsx | Upload a valid Excel file. |
| "Table not found" | No data exists in SQLite | Ensure data is uploaded first. |
| "No data available" | Chart fields empty | Select valid columns before generating a chart. |