# Software Design Document

## Data Visualization Dashboard

# Table of Contents

1. Introduction

- Purpose

- Scope

- Definitions, Acronyms, and Abbreviations

- References

- Overview

2. System Architecture

- Overall Description

- Major Components

- Data Flow Diagram

3. Design Considerations

- Assumptions and Dependencies

- Constraints

- Risks and Mitigations

4. Detailed System Design

- Frontend

- Component Structure

- State Management

- Interaction and Navigation

- Backend

- API Endpoints

- Data Processing

- Database Schema (if applicable)

- Desktop Application

- Electron Integration

5. User Interface Design

- UI Components

- Screenshots and Mockups

6. Testing and Validation

- Test Plan

- Test Cases

7. Deployment

- Deployment Diagram

- Steps for Deployment

8. Maintenance

- Logging and Monitoring

- Handling Updates and Patches

9. Future Enhancements

10. Conclusion

# 1. Introduction

## Purpose

The purpose of this document is to provide a detailed design for the Data Visualization Dashboard project. This document outlines the architecture, design considerations, component details, and user interface design to guide the development and future maintenance of the application.

## Scope

The Data Visualization Dashboard is designed to facilitate the comparison and analysis of multiple datasets through interactive line charts. It provides functionalities for dataset upload, axis selection, chart customization, statistics calculation, and data visualization.

## Definitions, Acronyms, and Abbreviations

- UI: User Interface  
- UX: User Experience  
- CSV: Comma-Separated Values  
- API: Application Programming Interface  
- CRUD: Create, Read, Update, Delete

## References

- React Documentation: https://reactjs.org/docs/getting-started.html  
- Node.js Documentation: https://nodejs.org/en/docs/  
- Electron Documentation: https://www.electronjs.org/docs  
- Chart.js Documentation: https://www.chartjs.org/docs/latest/

## Overview

This document provides a comprehensive overview of the design and architecture of the Data Visualization Dashboard. It includes details on the frontend, backend, and desktop application components, as well as UI design and testing strategies.

# 2. System Architecture

## Overall Description

The system architecture consists of three main components:  
1. Frontend: A React application that provides the user interface and interacts with the backend.  
2. Backend: A Node.js server that processes API requests, handles file uploads, and manages dataset processing.  
3. Desktop Application: An Electron application that packages the frontend and backend into a cross-platform desktop application.

## Major Components

- React Frontend  
 - Dataset Upload  
 - Axis Selection  
 - Chart Component  
 - Statistics Display  
 - Popup Component  
  
- Node.js Backend  
 - Express Server  
 - API Endpoints for dataset operations  
 - CSV Parsing and Data Processing  
  
- Electron Desktop Application  
 - Main Process  
 - Renderer Process

# 3. Design Considerations

## Assumptions and Dependencies

- Users will upload datasets in CSV format.  
- The application will run on modern web browsers and as a desktop application using Electron.  
- The backend server will handle data processing efficiently.

## Constraints

- The application must handle large datasets without significant performance degradation.  
- The UI must be responsive and user-friendly.

## Risks and Mitigations

- Risk: Large datasets may cause performance issues.  
 - Mitigation: Implement data aggregation and decimation techniques.  
- Risk: Cross-platform compatibility issues in Electron.  
 - Mitigation: Thoroughly test the Electron application on multiple platforms.

# 4. Detailed System Design

## Frontend

### Component Structure

- App.js: Main application component  
- DatasetUpload.js: Handles file uploads  
- DatasetList.js: Displays uploaded datasets and allows for customization  
- AxisSelection.js: Allows users to select attributes for the x and y axes  
- ChartComponent.js: Renders interactive charts using Chart.js  
- Popup.js: Displays dataset details in a popup  
- StatsCard.js: Displays calculated statistics for selected attributes

### State Management

- useState: For managing component state  
- useEffect: For handling side effects such as data fetching and event listeners

### Interaction and Navigation

- Event Handlers: For handling user interactions such as file uploads, axis selection, and button clicks  
- Responsive Design: Ensures the application is usable on different screen sizes

## Backend

### API Endpoints

- /datasets/upload: Endpoint for uploading datasets  
 - Method: POST  
 - Description: Accepts CSV files and stores them in the uploads directory  
  
- /datasets/chart-data: Endpoint for fetching processed chart data  
 - Method: POST  
 - Description: Accepts selected datasets and attributes, processes the data, and returns formatted data for charting  
  
- /datasets/delete/:fileName: Endpoint for deleting datasets  
 - Method: DELETE  
 - Description: Deletes the specified dataset file  
  
- /datasets/:dataset: Endpoint for fetching dataset details  
 - Method: GET  
 - Description: Returns the contents of the specified dataset

### Data Processing

- CSV Parsing: Uses `csv-parser` to parse CSV files and extract data  
- Statistics Calculation: Computes min, max, mean, variance, and standard deviation for selected attributes

## Desktop Application

### Electron Integration

- Main Process: Manages the lifecycle of the Electron application and creates browser windows  
- Renderer Process: Runs the React application within the Electron window  
- Preload Script: Provides a secure way to expose functionality to the renderer process

# 5. User Interface Design

## UI Components

- Header: Displays the application title and navigation links  
- Footer: Provides additional information and links  
- Dataset Upload: Allows users to upload datasets  
- Dataset List: Displays a list of uploaded datasets with options to view details and customize  
- Axis Selection: Provides dropdowns for selecting x and y-axis attributes  
- Chart Component: Displays interactive charts with zoom and pan capabilities  
- Popup: Displays dataset details in a modal overlay  
- Stats Cards: Shows statistical calculations for selected attributes

## Screenshots and Mockups

[Dataset Upload]  
[Axis Selection]  
[Chart Component]  
[Popup Component]  
[Stats Cards]

# 6. Testing and Validation

## Test Plan

- Unit Tests: Test individual components and functions  
- Integration Tests: Test interactions between different components  
- End-to-End Tests: Test the entire application flow

## Test Cases

- Dataset Upload: Verify that datasets can be uploaded and displayed correctly  
- Axis Selection: Ensure that selecting different attributes updates the chart correctly  
- Chart Rendering: Verify that charts are rendered correctly with the selected data  
- Popup Display: Test that the popup displays dataset details and can be closed properly  
- Statistics Calculation: Verify that statistical calculations are accurate and displayed correctly

# 7. Deployment

## Deployment Diagram

[Deployment Diagram]

## Steps for Deployment

1. Build the Frontend: Run `npm run build:frontend` to create a production build of the React application  
2. Package the Electron App: Run `npm run package` to package the Electron application for different platforms  
3. Deploy the Backend: Deploy the Node.js server to a hosting provider  
4. Distribute the Desktop App: Distribute the packaged Electron application to users

# 8. Maintenance

## Logging and Monitoring

- Backend Logs: Implement logging for backend operations and errors  
- Frontend Error Handling: Implement error boundaries to catch and display errors in the frontend  
- Monitoring: Use monitoring tools to track application performance and usage

## Handling Updates and Patches

- Version Control: Use version control to manage code changes and updates  
- Automatic Updates: Implement automatic updates for the Electron application using `electron-updater`

# 9. Future Enhancements

- Additional Chart Types: Integrate more chart types such as bar charts and scatter plots  
- Real-Time Data Updates: Enable real-time data updates and live chart rendering  
- User Authentication: Add user authentication and data saving features  
- Enhanced Data Processing: Implement advanced data processing techniques such as filtering and aggregation  
- Customizable Themes: Allow users to switch between different themes or create custom themes

# 10. Conclusion

The Data Visualization Dashboard provides a robust platform for visualizing and comparing multiple datasets. With its interactive charts, customizable settings, and statistical calculations, it offers valuable insights and enhances data analysis efficiency. Future enhancements will focus on expanding the functionality and improving the user experience further.