ChemData Visualizer

Complete Project Documentation

Full-Stack Application for Real-Time
Chemical Equipment Monitoring & Analysis

Technical Reference Manual

Mausam Kr

Version 1.0.0 | 2026 Edition

ChemData Visualizer - Project Documentation
Version 1.0.0

License: MIT License

Project Repository: github.com/mausamkar/chemdata-visualizer
Documentation: Built with LaTeX

Technology Stack:

- Backend: Django 5.0 + Django REST Framework
- Frontend: React 18 + Vite + Tailwind CSS
- Desktop: PyQt5
- Database: SQLite / PostgreSQL
- Deployment: Docker

For contributions, issues, or questions, please visit the project repository.

Dedicated to all developers who build solutions
that make data accessible and actionable.

Contents

Preface **7**

1 System Overview **9**
1.1 System Architecture . . . 9
1.1.1 Standard Deployment Architecture . . . 9
1.1.2 Component Responsibilities . . . 10
1.2 Key Features . . . 10
1.2.1 Security & Authentication . . . 10
1.2.2 Data Visualization . . . 11
1.2.3 Dataset Management . . . 11
1.2.4 Reporting . . . 11
1.3 Technology Stack . . . 12
1.4 Data Flow Pipeline . . . 12
1.5 Project Structure . . . 14
1.6 Design Philosophy . . . 14
1.6.1 Unified Design Language . . . 14
1.6.2 Responsive Design . . . 15
1.6.3 Performance Optimization . . . 15

2 Backend API Server **16**
2.1 Architecture Overview . . . 16
2.1.1 Core Components . . . 16
2.2 Installation & Setup . . . 16
2.2.1 Prerequisites . . . 16
2.2.2 Quick Start . . . 16
2.3 Key Features . . . 17
2.3.1 Authentication System . . . 17
2.3.2 Data Processing . . . 17
2.3.3 Statistical Analysis . . . 18
2.3.4 Report Generation . . . 18
2.4 API Endpoints . . . 18
2.4.1 Authentication Endpoints . . . 18
2.4.2 Dataset Management Endpoints . . . 18
2.4.3 Analysis Endpoints . . . 19
2.5 Database Models . . . 19

2.5.1 User Model . . . 19
2.5.2 Dataset Model . . . 19
2.5.3 EquipmentRecord Model . . . 20
2.6 Security Features . . . 20
2.7 Testing . . . 20

3 Web Frontend **22**
3.1 Architecture Overview . . . 22
3.1.1 Technology Stack . . . 22
3.2 Installation & Setup . . . 22
3.2.1 Prerequisites . . . 22
3.2.2 Quick Start . . . 22
3.3 Key Components . . . 23
3.3.1 Application Structure . . . 23
3.3.2 Core Components . . . 23
3.4 Features . . . 24
3.4.1 UI Design System . . . 24
3.4.2 Chart Types . . . 25
3.4.3 Authentication . . . 25
3.5 Routing Configuration . . . 25
3.6 API Integration . . . 25
3.7 Build & Deployment . . . 26
3.7.1 Development Mode . . . 26
3.7.2 Production Build . . . 26
3.8 Performance Optimizations . . . 26

4 Desktop Application **27**
4.1 Architecture Overview . . . 27
4.1.1 Technology Stack . . . 27
4.2 Installation & Setup . . . 27
4.2.1 Prerequisites . . . 27
4.2.2 Quick Start . . . 27
4.3 Application Structure . . . 28
4.4 Key Components . . . 28
4.4.1 Authentication Window (auth.py) . . . 28
4.4.2 Dashboard Window (dashboard.py) . . . 29
4.4.3 Centralized Styling (styles.py) . . . 29
4.5 Features . . . 29
4.5.1 Native UI Benefits . . . 29
4.5.2 Data Visualization . . . 30
4.5.3 Dashboard Interface . . . 30

4.6 Configuration . . . 31
4.6.1 config.py Settings . . . 31
4.7 Building for Distribution . . . 31
4.7.1 Creating Standalone Executable . . . 31
4.7.2 Platform-Specific Notes . . . 31
4.8 Design Parity . . . 32

5 Deployment with Docker 33
5.1 Docker Overview . . . 33
5.1.1 Why Docker? . . . 33
5.1.2 Architecture . . . 33
5.2 Prerequisites . . . 34
5.3 Quick Start . . . 34
5.3.1 Option 1: Docker Compose (Recommended) . . . 34
5.3.2 Option 2: Individual Containers . . . 34
5.4 Docker Compose Configuration . . . 34
5.4.1 Services . . . 35
5.4.2 Volumes . . . 35
5.4.3 Networks . . . 35
5.5 Environment Configuration . . . 35
5.6 Common Commands . . . 35
5.6.1 Starting Services . . . 35
5.6.2 Stopping Services . . . 36
5.6.3 Viewing Logs . . . 36
5.6.4 Executing Commands . . . 36
5.7 Production Deployment . . . 37
5.7.1 Differences from Development . . . 37
5.7.2 Production Checklist . . . 37
5.8 Troubleshooting . . . 37
5.8.1 Common Issues . . . 37

6 Database Access & Management 39
6.1 Database Overview . . . 39
6.1.1 Default Configuration . . . 39
6.1.2 Database Schema . . . 39
6.2 Accessing the Database . . . 39
6.2.1 Method 1: Django Admin Interface (Recommended) . . . 39
6.2.2 Method 2: Database Browser Tools . . . 40
6.2.3 Method 3: Django Shell . . . 40
6.3 Database Migrations . . . 41
6.3.1 Creating Migrations . . . 41

6.3.2 Applying Migrations . . . 41
6.3.3 Viewing Migration Status . . . 41
6.4 Switching to PostgreSQL . . . 41
6.4.1 Installation . . . 41
6.5 Backup & Restore . . . 42
6.5.1 SQLite Backup . . . 42
6.5.2 PostgreSQL Backup . . . 42
6.5.3 Django Fixtures . . . 42
6.6 Database Optimization . . . 43
6.6.1 Indexing . . . 43
6.6.2 Query Optimization . . . 43
6.7 Common Database Tasks . . . 43
6.7.1 Reset Database . . . 43
6.7.2 View Raw SQL . . . 43
6.7.3 Database Shell . . . 44

A Quick Reference **45**
A.1 Installation Commands . . . 45
A.1.1 Backend Setup . . . 45
A.1.2 Frontend Setup . . . 45
A.1.3 Desktop App Setup . . . 45
A.2 Docker Commands . . . 45
A.3 API Endpoints Reference . . . 46
A.4 Troubleshooting . . . 46
A.4.1 Port Conflicts . . . 46
A.4.2 Database Issues . . . 46
A.4.3 Docker Issues . . . 47
A.5 Contributing . . . 47
A.6 License . . . 47
A.7 Contact & Support . . . 47

Preface

The **ChemData Visualizer** is a comprehensive, full-stack application designed to monitor and analyze chemical equipment data in real-time. This documentation provides a complete technical reference for understanding, deploying, and extending the system.

Purpose of This Documentation

This manual serves as the authoritative guide for:

- Understanding the overall system architecture
- Setting up development and production environments
- Navigating the codebase and project structure
- Deploying the application using Docker
- Extending functionality with new features
- Troubleshooting common issues

Project Overview

ChemData Visualizer combines three distinct client applications—a web dashboard, a desktop application, and a REST API—into a unified ecosystem. Users can upload CSV datasets containing equipment measurements, visualize trends through interactive charts, and generate professional PDF reports.

The application demonstrates modern software engineering practices including:

- RESTful API design with Django REST Framework
- Component-based UI development with React
- Native desktop application development with PyQt5
- Secure authentication with JWT and OAuth2
- Data processing and analysis with Pandas
- Containerized deployment with Docker

Who Should Use This Documentation

This documentation is intended for:

- Developers joining the project team
- System administrators deploying the application
- Students learning full-stack development
- Technical evaluators assessing the system

Document Structure

The documentation is organized as follows:

Chapter 1 Provides a high-level overview of the system architecture and key features

Chapter 2 Details the Django backend implementation and API structure

Chapter 3 Explains the React web frontend and its components

Chapter 4 Covers the PyQt5 desktop application

Chapter 5 Describes the deployment process using Docker

Chapter 6 Offers guidance on database access and management

Mausam Kr
January 2026

Chapter 1

System Overview

ChemData Visualizer is a robust, full-stack application designed for real-time monitoring and analysis of chemical equipment data. The system features a secure Django REST API backend, a modern React web dashboard, and a native PyQt5 desktop application.

1.1 System Architecture

The application follows a three-tier architecture with clear separation between presentation, business logic, and data layers.

1.1.1 Standard Deployment Architecture

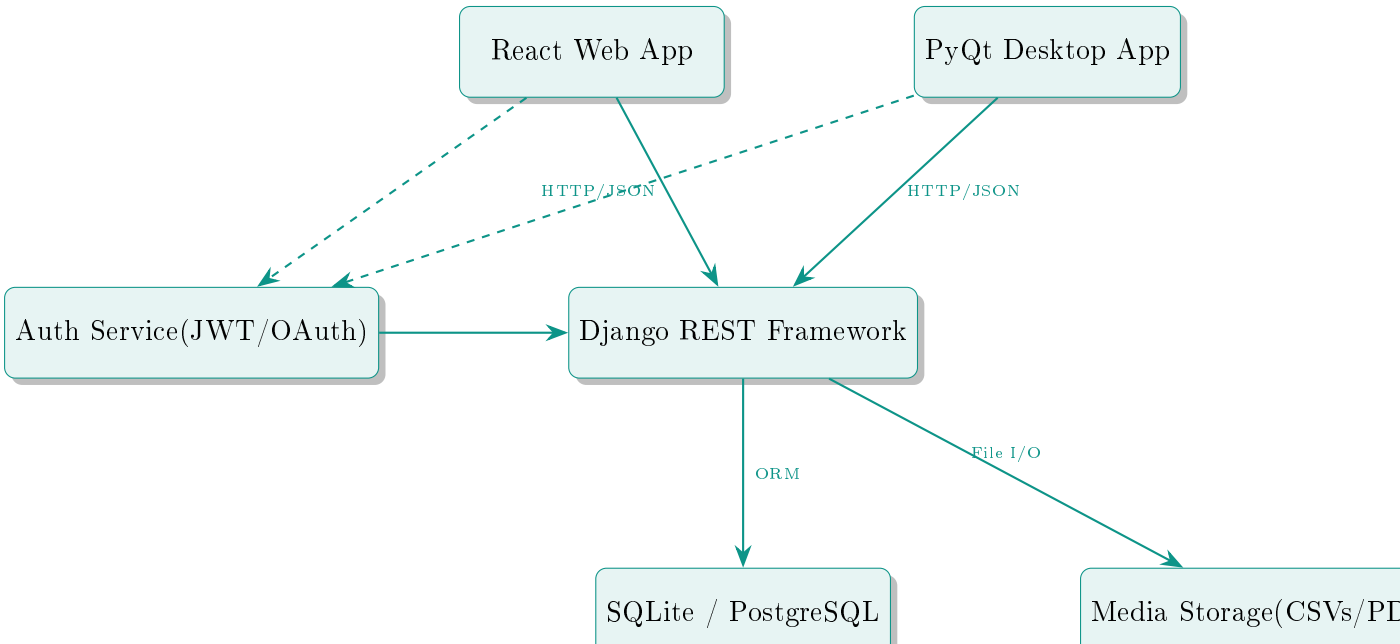

Figure 1.1: ChemData Visualizer System Architecture

1.1.2 Component Responsibilities

Backend (Django REST Framework)

- User authentication and authorization
- CSV file processing and validation
- Statistical analysis and aggregation
- PDF report generation
- RESTful API endpoints
- Database operations via ORM

Web Frontend (React + Vite)

- Responsive dashboard interface
- Interactive data visualizations (Chart.js)
- File upload with drag-and-drop
- Google OAuth2 integration
- Real-time data updates
- Mobile-responsive design

Desktop Application (PyQt5)

- Native Windows/Linux/macOS interface
- Advanced plotting with Matplotlib
- Offline capability for certain features
- System integration
- Consistent design with web frontend

1.2 Key Features

1.2.1 Security & Authentication

- **JWT-based Token Authentication**: Secure, stateless authentication for API access

- **Google OAuth2 Integration**: Social login for web application
- **User Isolation**: Each user accesses only their own datasets
- **CSRF Protection**: Built-in Django security features

1.2.2 Data Visualization

Web Dashboard:

- Interactive Chart.js graphs (Bar, Trend, Distribution, Scatter)
- Real-time KPI cards
- Equipment type filtering
- Responsive grid layouts

Desktop Application:

- Native Matplotlib charts
- Interactive metric dropdowns
- Correlation analysis tools
- Tabbed interface for multiple views

1.2.3 Dataset Management

- CSV file upload with validation
- Automatic data parsing using Pandas
- User-specific and global dataset history
- Bulk data operations
- Dataset metadata tracking

1.2.4 Reporting

- Professional PDF report generation using ReportLab
- Customizable report templates
- Statistical summaries and charts
- One-click download functionality

1.3 Technology Stack

Component	Technology	Purpose
Backend Framework	Django 5.0	Python web framework with ORM
API Framework	Django REST Framework	RESTful API development
Database	SQLite / PostgreSQL	Data persistence
Web Frontend	React 18 + Vite	Modern UI library with fast build
Styling	Tailwind CSS	Utility-first CSS framework
Desktop App	PyQt5	Cross-platform GUI framework
Charts (Web)	Chart.js	Interactive web visualizations
Charts (Desktop)	Matplotlib	Scientific plotting library
Data Processing	Pandas	Data manipulation and analysis
PDF Generation	ReportLab	Programmatic PDF creation
Deployment	Docker	Containerization platform

Table 1.1: Complete Technology Stack

1.4 Data Flow Pipeline

The following diagram illustrates how data flows through the system from upload to visualization:

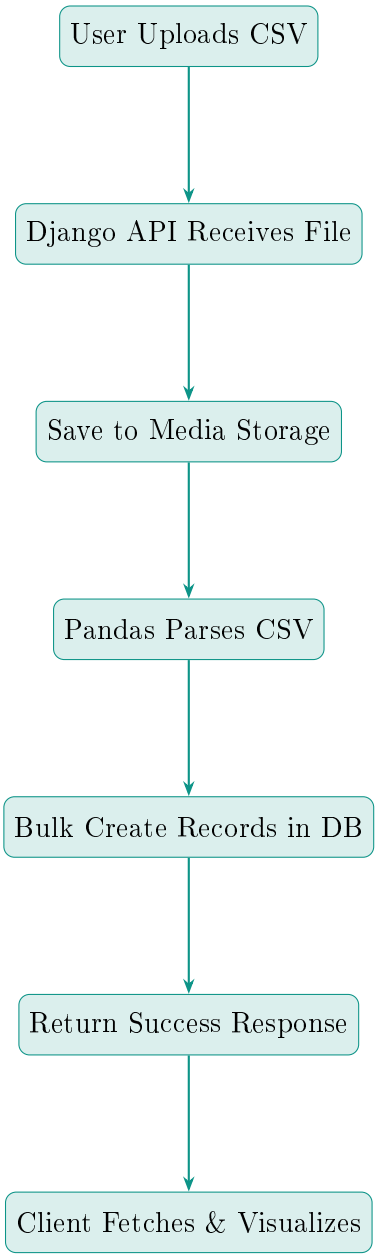

Figure 1.2: Data Upload and Processing Pipeline

1.5 Project Structure

Directory Layout

```
ChemData-Visualizer/
 backend/                  # Django Server
    api/                  # API App (models, views, serializers)
    core/                 # Project Settings & Configuration
    manage.py             # Django Management CLI
    requirements.txt      # Python Dependencies

 web-frontend/             # React Web Application
    src/                  # Source Code
       components/        # React Components
       App.jsx            # Main Application Router
       api.js             # Axios API Client
    public/               # Static Assets
    package.json          # Node Dependencies

 desktop-frontend/         # PyQt5 Desktop Application
    ui/                   # UI Components
       auth.py            # Login/Signup Windows
       dashboard.py       # Main Dashboard
       styles.py          # Centralized Styles
    main.py               # Application Entry Point
    config.py             # Configuration

 sample_equipment_data.csv # Test Dataset
 docker-compose.yml        # Docker Orchestration
 README.md                 # Project Documentation
```

1.6 Design Philosophy

1.6.1 Unified Design Language

All three client applications (web, desktop, and mobile-ready) share a consistent design language:

- **Primary Color**: Teal (`#0d9488`) for interactive elements
- **Secondary Color**: Slate (`#475569`) for text and borders
- **Typography**: Clean, modern sans-serif fonts

- **Spacing**: Generous whitespace for readability
- **Components**: Consistent card layouts, buttons, and form elements

1.6.2 Responsive Design

The web frontend adapts seamlessly across devices:

- Mobile: Stacked layouts, touch-friendly controls
- Tablet: Hybrid layouts with collapsible sidebars
- Desktop: Full grid layouts with multiple columns

1.6.3 Performance Optimization

- Efficient database queries with proper indexing
- Bulk operations for large datasets
- Lazy loading of charts and components
- Optimized Docker images with multi-stage builds

Chapter 2

Backend API Server

The Django backend serves as the central hub of the ChemData Visualizer ecosystem, handling all business logic, data processing, and API endpoints.

2.1 Architecture Overview

The backend follows Django's Model-View-Template pattern, adapted for API development using Django REST Framework.

2.1.1 Core Components

Models Define database schema and relationships

Serializers Handle JSON conversion and validation

Views Implement business logic and API endpoints

URLs Route requests to appropriate views

2.2 Installation & Setup

2.2.1 Prerequisites

- Python 3.10 or higher
- pip package manager
- Virtual environment tool (recommended)

2.2.2 Quick Start

1. **Create Virtual Environment**

```
python -m venv venv
# Windows: venv\Scripts\activate
# Mac/Linux: source venv/bin/activate
```

2. **Install Dependencies**

```
pip install -r requirements.txt
```

3. **Configure Environment Variables**

   Create a `.env` file in the backend directory:

```
SECRET_KEY=your_django_secret_key
DEBUG=True
GOOGLE_CLIENT_ID=your_google_client_id
GOOGLE_CLIENT_SECRET=your_google_client_secret
```

4. **Run Database Migrations**

```
python manage.py migrate
```

5. **Start Development Server**

```
python manage.py runserver
```

   The API will be available at `http://127.0.0.1:8000/`

2.3 Key Features

2.3.1 Authentication System

- **Token-based Authentication**: DRF's built-in token auth
- **Google OAuth2**: Social login integration via django-allauth
- **User Registration**: Custom endpoint for account creation

2.3.2 Data Processing

- **CSV Parsing**: Pandas integration for efficient file processing
- **Bulk Operations**: Optimized database inserts for large datasets
- **Validation**: Comprehensive input validation at multiple levels

2.3.3 Statistical Analysis

- **Aggregation**: Django ORM aggregation functions
- **Filtering**: Query optimization with proper indexing
- **Grouping**: Equipment type and metric-based grouping

2.3.4 Report Generation

- **PDF Creation**: ReportLab for professional documents
- **Custom Formatting**: Branded headers, tables, and styling
- **On-Demand Generation**: Real-time report creation

2.4 API Endpoints

2.4.1 Authentication Endpoints

Method	Endpoint	Description
POST	/api/auth/registration/	Register new user account
POST	/api/api-token-auth/	Login and retrieve auth token
POST	/accounts/google/login/	Google OAuth2 authentication

Table 2.1: Authentication API Endpoints

2.4.2 Dataset Management Endpoints

Method	Endpoint	Description
GET	/api/datasets/	List all user datasets
POST	/api/upload/	Upload new CSV dataset
GET	/api/datasets/global/	List global dataset history
GET	/api/datasets/{id}/	Retrieve specific dataset
DELETE	/api/datasets/{id}/	Delete dataset

Table 2.2: Dataset Management Endpoints

2.4.3 Analysis Endpoints

Method	Endpoint	Description
GET	/api/datasets/{id}/stats/	Get statistical aggregations
GET	/api/datasets/{id}/data/	Retrieve raw record data
GET	/api/datasets/{id}/pdf/	Download PDF report

Table 2.3: Data Analysis Endpoints

2.5 Database Models

2.5.1 User Model

Uses Django's built-in authentication system with the standard User model.

2.5.2 Dataset Model

Dataset Model Fields

- `user`: ForeignKey to User (owner)
- `file`: FileField (path to uploaded CSV)
- `uploaded_at`: DateTimeField (auto-generated)
- `name`: CharField (dataset name)
- `description`: TextField (optional description)

2.5.3 EquipmentRecord Model

EquipmentRecord Model Fields

- `dataset`: ForeignKey to Dataset
- `equipment_name`: CharField (e.g., "Pump-01")
- `equipment_type`: CharField with choices (PUMP, VALVE, TANK, REACTOR, HEAT_EXCHANGER)
- `flowrate`: FloatField (m^3/h)
- `pressure`: FloatField (bar)
- `temperature`: FloatField (°C)
- `timestamp`: DateTimeField (auto-generated)

2.6 Security Features

- **CSRF Protection**: Enabled by default for all POST requests
- **SQL Injection Prevention**: Django ORM parameterized queries
- **XSS Protection**: Automatic HTML escaping
- **Authentication Required**: Most endpoints require valid tokens
- **User Isolation**: Users can only access their own data
- **File Validation**: CSV file type and size restrictions

2.7 Testing

Run the test suite with:

```
python manage.py test api
```

Tests cover:

- Model creation and relationships
- API endpoint responses
- Authentication flows

- Data validation
- Statistical calculations

Chapter 3

Web Frontend

The React-based web frontend provides a modern, responsive interface for interacting with the ChemData Visualizer system.

3.1 Architecture Overview

Built with React 18 and Vite for optimal development experience and production performance.

3.1.1 Technology Stack

- **React 18**: Modern UI library with hooks
- **Vite 4**: Next-generation build tool
- **Tailwind CSS**: Utility-first styling
- **Chart.js**: Interactive data visualizations
- **Axios**: HTTP client for API communication
- **React Router**: Client-side routing

3.2 Installation & Setup

3.2.1 Prerequisites

- Node.js 18+ (LTS recommended)
- npm or yarn package manager

3.2.2 Quick Start

1. **Install Dependencies**

```
npm install
```

2. **Configure Environment**

   Create `.env` file:

   ```
   VITE_GOOGLE_CLIENT_ID=your_google_client_id
   ```

3. **Start Development Server**

   ```
   npm run dev
   ```

   Access at: `http://localhost:5173`

4. **Build for Production**

   ```
   npm run build
   ```

3.3 Key Components

3.3.1 Application Structure

Component Hierarchy

```
src/
 components/
    Login.jsx           # Authentication screen
    Signup.jsx          # User registration
    Analysis.jsx        # Main dashboard
    Features.jsx        # Feature overview page
    Support.jsx         # Support & contact page
    Sidebar.jsx         # Navigation component
 App.jsx                 # Root component & routing
 api.js                  # API client configuration
 main.jsx                # Application entry point
 index.css               # Global styles
```

3.3.2 Core Components

App.jsx

- Main application router

- Authentication state management
- Route protection logic
- Layout switching (full-screen vs. navbar)

Analysis.jsx

- Primary dashboard interface
- KPI card displays
- Chart.js visualizations
- Dataset selection
- PDF report download

api.js

- Axios instance configuration
- Automatic token attachment
- Request/response interceptors
- Error handling

3.4 Features

3.4.1 UI Design System

- **Split-Screen Authentication**: Modern login/signup with visual branding
- **Responsive Layouts**: Mobile-first design with Tailwind breakpoints
- **Interactive Graphs**: Chart.js for dynamic visualizations
- **KPI Cards**: At-a-glance statistical summaries
- **Navigation Bar**: Sticky capsule-style navigation

3.4.2 Chart Types

Chart Type	Purpose
Bar Chart	Average metrics by equipment type
Trend Chart	Time-series process monitoring
Distribution Chart	Equipment type composition (donut)
Scatter Plot	Correlation analysis between variables

Table 3.1: Visualization Types

3.4.3 Authentication

- **Standard Login**: Username/password authentication
- **Google OAuth2**: Social login integration
- **Token Storage**: LocalStorage for persistent sessions
- **Auto-Logout**: Automatic redirect on token expiration

3.5 Routing Configuration

Route	Component / Purpose
/login	Login page (public)
/signup	Registration page (public)
/dashboard	Main analysis dashboard (protected)
/features	Feature showcase page
/support	Support and contact information
/	Redirects to dashboard

Table 3.2: Application Routes

3.6 API Integration

All API calls are centralized through the `api.js` module:

- Base URL: `http://127.0.0.1:8000/api/`
- Automatic Authorization header injection
- Centralized error handling
- Response interceptors for token refresh

3.7 Build & Deployment

3.7.1 Development Mode

```
npm run dev
```

Features:

- Hot module replacement (HMR)
- Fast refresh
- Source maps for debugging
- Dev server on port 5173

3.7.2 Production Build

```
npm run build
```

Generates optimized static files in `dist/` folder:

- Minified JavaScript bundles
- Optimized CSS
- Asset compression
- Tree-shaking for smaller bundle size

3.8 Performance Optimizations

- **Code Splitting**: Lazy loading of route components
- **Image Optimization**: Compressed assets
- **Caching Strategy**: Efficient browser caching
- **Bundle Analysis**: Vite's built-in analyzer

Chapter 4

Desktop Application

The PyQt5 desktop application provides a native interface for ChemData Visualizer with platform-specific advantages and system integration.

4.1 Architecture Overview

Built with PyQt5, the desktop application mirrors the web frontend's functionality while leveraging native OS capabilities.

4.1.1 Technology Stack

- **PyQt5**: Python bindings for Qt5 framework
- **Matplotlib**: Scientific plotting and visualization
- **Requests**: HTTP library for API communication
- **Pandas**: Data manipulation and analysis

4.2 Installation & Setup

4.2.1 Prerequisites

- Python 3.10+
- Backend API running at `http://127.0.0.1:8000`

4.2.2 Quick Start

1. **Install Dependencies**

```
pip install pyqt5 matplotlib requests pandas
```

2. **Configure API URL**

   Edit `config.py`:

```
API_URL = "http://127.0.0.1:8000/api/"
```

3. **Run Application**

```
python main.py
```

4.3 Application Structure

Project Structure

```
desktop-frontend/
 ui/
    auth.py              # Login/Signup windows
    dashboard.py         # Main application interface
    styles.py            # Centralized stylesheets
 main.py                  # Application entry point
 config.py                # Configuration constants
 README.md                # Documentation
```

4.4 Key Components

4.4.1 Authentication Window (auth.py)

Features:

- Split-screen design (form + branding)
- Login and signup modes
- Token-based authentication
- Error message display
- Automatic transition to dashboard

UI Elements:

- Username/password input fields
- Remember me checkbox
- Sign in / Sign up buttons
- Teal gradient branding panel

4.4.2 Dashboard Window (dashboard.py)

Layout:

- Sidebar for dataset navigation
- Stacked content area (empty state / analysis view)
- Tabbed interface for different views
- KPI cards for quick statistics

Tabs:

Analysis Dashboard Interactive charts and visualizations

Data Logs Detailed table with styled badges

4.4.3 Centralized Styling (styles.py)

Maintains consistent design across components:

- Primary color: `#0d9488` (Teal)
- Secondary color: `#475569` (Slate)
- Background: `#f8fafc` (Light gray)
- Reusable QSS stylesheets for buttons, inputs, cards

4.5 Features

4.5.1 Native UI Benefits

- **Platform Integration**: Native window controls and system tray
- **Performance**: Direct rendering without browser overhead
- **Offline Capability**: Local data caching and processing
- **System Resources**: Access to local files and hardware

4.5.2 Data Visualization

Matplotlib Integration:

- Bar charts for equipment metrics
- Donut charts for type distribution
- Trend lines for process monitoring
- Scatter plots for correlation analysis

Interactive Features:

- Metric selection dropdowns
- Dynamic chart updates
- Zoom and pan capabilities
- Export to image files

4.5.3 Dashboard Interface

KPI Cards:

- Total Records count
- Average Flowrate
- Average Pressure
- Average Temperature

Data Table:

- Sortable columns
- Styled badges for equipment types
- Conditional formatting (e.g., high temperature in red)
- Row selection and filtering

4.6 Configuration

4.6.1 config.py Settings

Configuration Options

- `API_URL`: Backend API endpoint
- `APP_NAME`: Application name
- `VERSION`: Application version
- `WINDOW_WIDTH`: Default window width (1400px)
- `WINDOW_HEIGHT`: Default window height (800px)
- `SIDEBAR_WIDTH`: Sidebar width (300px)

4.7 Building for Distribution

4.7.1 Creating Standalone Executable

Use PyInstaller to package the application:

```
pip install pyinstaller
pyinstaller --noconsole --onefile main.py
```

Options:

- `-noconsole`: Hide console window
- `-onefile`: Single executable file
- `-icon=icon.ico`: Custom application icon
- `-name=ChemData`: Output executable name

4.7.2 Platform-Specific Notes

Windows:

- Generates `.exe` file
- May require Visual C++ redistributables

macOS:

- Creates `.app` bundle
- Code signing required for distribution

Linux:

- Produces ELF binary
- May need additional system libraries

4.8 Design Parity

The desktop application maintains visual consistency with the web frontend:

- Identical color scheme (Teal & Slate)
- Matching typography and spacing
- Similar layout patterns
- Consistent iconography
- Unified branding elements

Chapter 5

Deployment with Docker

Docker containerization enables consistent, reproducible deployments across different environments.

5.1 Docker Overview

5.1.1 Why Docker?

- **Consistency**: Identical environments from development to production
- **Isolation**: Each service runs in its own container
- **Portability**: Deploy anywhere Docker runs
- **Scalability**: Easy horizontal scaling
- **Simplicity**: Single command deployment

5.1.2 Architecture

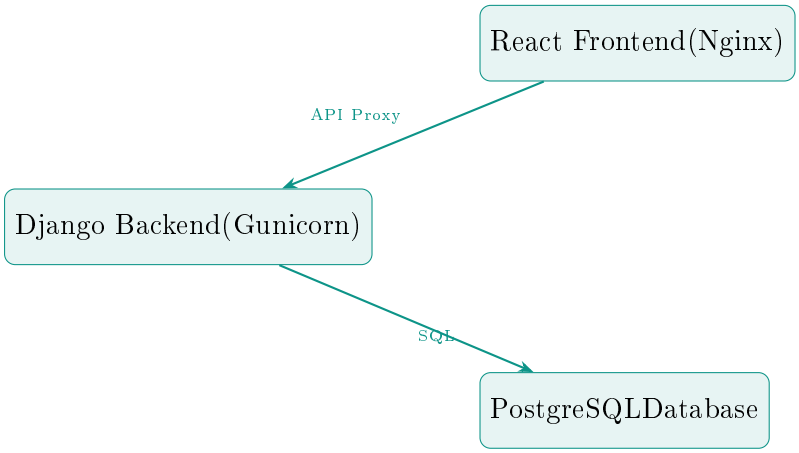

Figure 5.1: Dockerized Application Architecture

5.2 Prerequisites

- Docker Desktop installed and running
- Docker Compose (included with Docker Desktop)
- 4GB+ RAM allocated to Docker

5.3 Quick Start

5.3.1 Option 1: Docker Compose (Recommended)

Start all services with a single command:

```
docker-compose up --build
```

Access Points:

- Web Application: `http://localhost:5173`
- Backend API: `http://127.0.0.1:8000`
- Admin Panel: `http://127.0.0.1:8000/admin`

5.3.2 Option 2: Individual Containers

Build Backend:

```
cd backend
docker build -t chemdata-backend .
docker run -p 8000:8000 chemdata-backend
```

Build Frontend:

```
cd web-frontend
docker build -t chemdata-frontend .
docker run -p 5173:5173 chemdata-frontend
```

5.4 Docker Compose Configuration

The `docker-compose.yml` orchestrates all services:

5.4.1 Services

backend Django REST API server

frontend React web application

db PostgreSQL database (optional, defaults to SQLite)

5.4.2 Volumes

Persistent data storage:

- `media_volume`: User-uploaded CSV files
- `static_volume`: Collected Django static files
- `postgres_data`: Database files (if using PostgreSQL)

5.4.3 Networks

`app-network`: Bridge network for inter-container communication

5.5 Environment Configuration

Create a `.env` file in the project root:

.env File Example

```
# Django Settings
SECRET_KEY=your-secret-key-here
DEBUG=False
ALLOWED_HOSTS=localhost,127.0.0.1

# Database (optional, defaults to SQLite)
DATABASE_URL=postgresql://user:pass@db:5432/chemdata

# Google OAuth
GOOGLE_CLIENT_ID=your-client-id
GOOGLE_CLIENT_SECRET=your-client-secret
```

5.6 Common Commands

5.6.1 Starting Services

```
# Start in foreground
```

```
docker-compose up

# Start in background (detached mode)
docker-compose up -d

# Rebuild containers
docker-compose up --build
```

5.6.2 Stopping Services

```
# Stop containers
docker-compose stop

# Stop and remove containers
docker-compose down

# Remove containers and volumes (WARNING: deletes data)
docker-compose down -v
```

5.6.3 Viewing Logs

```
# All services
docker-compose logs -f

# Specific service
docker-compose logs -f backend
```

5.6.4 Executing Commands

```
# Create Django superuser
docker-compose exec backend python manage.py createsuperuser

# Run migrations
docker-compose exec backend python manage.py migrate

# Access database
docker-compose exec db psql -U user -d chemdata
```

5.7 Production Deployment

5.7.1 Differences from Development

Aspect	Development	Production
DEBUG	True	False
Server	Django runserver	Gunicorn
Static Files	Development server	Nginx
Database	SQLite	PostgreSQL
HTTPS	No	Yes (with SSL)
Logs	Console	Files + monitoring

Table 5.1: Development vs Production Configuration

5.7.2 Production Checklist

Set `DEBUG=False`

Use strong `SECRET_KEY`

Configure `ALLOWED_HOSTS`

Enable HTTPS with SSL certificates

Use PostgreSQL instead of SQLite

Set up log aggregation

Configure automated backups

Implement monitoring and alerting

5.8 Troubleshooting

5.8.1 Common Issues

Port Already in Use:

```
# Find process using port 8000
lsof -i :8000
# Kill process
kill -9 <PID>
```

Permission Denied:

```
# Run with sudo (Linux/Mac)
sudo docker-compose up
```

Database Connection Errors:

- Ensure database container is running
- Check `DATABASE_URL` in `.env`
- Verify network connectivity between containers

Build Failures:

```
# Clear Docker cache
docker-compose build --no-cache
```

Chapter 6

Database Access & Management

ChemData Visualizer uses SQLite by default for development and can scale to PostgreSQL for production.

6.1 Database Overview

6.1.1 Default Configuration

- **Database**: SQLite 3
- **Location**: `backend/db.sqlite3`
- **ORM**: Django ORM
- **Migrations**: Django migrations system

6.1.2 Database Schema

Table	Purpose
auth_user	Django user accounts
authtoken_token	API authentication tokens
api_dataset	Uploaded CSV metadata
api_equipmentrecord	Individual equipment data points

Table 6.1: Core Database Tables

6.2 Accessing the Database

6.2.1 Method 1: Django Admin Interface (Recommended)

The Django admin provides a user-friendly web interface for managing data.

Step 1: Create Superuser

```
cd backend
python manage.py createsuperuser
```

Follow prompts to set username, email, and password.

Step 2: Access Admin Panel

- URL: `http://127.0.0.1:8000/admin`
- Login with superuser credentials

Features:

- Browse all datasets and records
- Create, edit, and delete entries
- Filter and search functionality
- Bulk operations
- User management

6.2.2 Method 2: Database Browser Tools

VS Code Extensions:

- SQLite Viewer (by Florian Klampfer)
- SQLite (by alexcvzz)

Desktop Applications:

- DB Browser for SQLite (https://sqlitebrowser.org)
- DBeaver (universal database tool)

6.2.3 Method 3: Django Shell

For programmatic database access:

```
python manage.py shell
```

Example queries:

```
>>> from api.models import Dataset, EquipmentRecord
>>> Dataset.objects.all()
>>> EquipmentRecord.objects.filter(temperature__gt=100)
```

6.3 Database Migrations

6.3.1 Creating Migrations

After modifying models:

```
python manage.py makemigrations
```

This generates migration files in `api/migrations/`.

6.3.2 Applying Migrations

```
python manage.py migrate
```

This updates the database schema to match your models.

6.3.3 Viewing Migration Status

```
python manage.py showmigrations
```

Shows which migrations have been applied.

6.4 Switching to PostgreSQL

For production deployments, PostgreSQL is recommended.

6.4.1 Installation

Step 1: Install PostgreSQL

- Download from https://www.postgresql.org/download/
- Or use Docker: `docker run -p 5432:5432 postgres:15`

Step 2: Install Python Driver

```
pip install psycopg2-binary
```

Step 3: Update Django Settings

In `backend/core/settings.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'chemdata',
        'USER': 'postgres',
        'PASSWORD': 'your_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Step 4: Run Migrations

```
python manage.py migrate
```

6.5 Backup & Restore

6.5.1 SQLite Backup

Simple file copy:

```
cp backend/db.sqlite3 backend/db.sqlite3.backup
```

6.5.2 PostgreSQL Backup

```
# Backup
pg_dump -U postgres chemdata > backup.sql

# Restore
psql -U postgres chemdata < backup.sql
```

6.5.3 Django Fixtures

Export data to JSON:

```
python manage.py dumpdata > data.json
```

Import data:

```
python manage.py loaddata data.json
```

6.6 Database Optimization

6.6.1 Indexing

The models include strategic indexes for common queries:

- User + upload date for dataset listings
- Dataset + equipment type for filtering
- Equipment name for searching

6.6.2 Query Optimization

- Use `select_related()` for foreign keys
- Use `prefetch_related()` for reverse relations
- Use `only()` to fetch specific fields
- Use aggregation instead of Python loops

6.7 Common Database Tasks

6.7.1 Reset Database

```
# Delete database file
rm backend/db.sqlite3

# Recreate with migrations
python manage.py migrate

# Create superuser
python manage.py createsuperuser
```

6.7.2 View Raw SQL

```
python manage.py sqlmigrate api 0001
```

Shows the SQL that will be executed for a migration.

6.7.3 Database Shell

Direct SQL access:

```
python manage.py dbshell
```

Opens SQLite/PostgreSQL command-line interface.

Appendix A

Quick Reference

A.1 Installation Commands

A.1.1 Backend Setup

```
cd backend
python -m venv venv
venv\Scripts\activate  # Windows
source venv/bin/activate  # Mac/Linux
pip install -r requirements.txt
python manage.py migrate
python manage.py runserver
```

A.1.2 Frontend Setup

```
cd web-frontend
npm install
npm run dev
```

A.1.3 Desktop App Setup

```
cd desktop-frontend
pip install -r requirements.txt
python main.py
```

A.2 Docker Commands

```
# Start all services
docker-compose up --build

# Stop services
docker-compose down

# View logs
```

```
docker-compose logs -f backend

# Execute command in container
docker-compose exec backend python manage.py migrate

# Rebuild specific service
docker-compose build backend
```

A.3 API Endpoints Reference

Method	Endpoint	Description
POST	/api/auth/registration/	Register user
POST	/api/api-token-auth/	Login
GET	/api/datasets/	List datasets
POST	/api/upload/	Upload CSV
GET	/api/datasets/{id}/stats/	Get statistics
GET	/api/datasets/{id}/pdf/	Download report

A.4 Troubleshooting

A.4.1 Port Conflicts

Backend port 8000 in use:

```
# Windows
netstat -ano | findstr :8000
taskkill /PID <PID> /F

# Mac/Linux
lsof -i :8000
kill -9 <PID>
```

A.4.2 Database Issues

Migration conflicts:

```
python manage.py migrate --fake
python manage.py migrate
```

Reset database:

```
rm db.sqlite3
python manage.py migrate
```

A.4.3 Docker Issues

Clear Docker cache:

```
docker system prune -a
docker-compose build --no-cache
```

A.5 Contributing

To contribute to ChemData Visualizer:

1. Fork the repository
2. Create a feature branch: `git checkout -b feature/amazing-feature`
3. Commit changes: `git commit -m 'Add amazing feature'`
4. Push to branch: `git push origin feature/amazing-feature`
5. Open a Pull Request

A.6 License

This project is licensed under the MIT License.

A.7 Contact & Support

- **Author**: Mausam Kr
- **GitHub**: github.com/mausamkar/chemdata-visualizer
- **Issues**: Use GitHub Issues for bug reports
- **Documentation**: This manual and README files

Thank you for using ChemData Visualizer!

Made with love by Mausam Kr