Database Setup Instructions for Microsoft Azure

1. Azure SQL Database Setup:

Step 1: Create Azure Account & Resource

1. Go to Azure Portal (portal.azure.com)
2. Create new Resource Group
3. Create SQL Database resource
   - Choose subscription (the free one is fine)
   - Select resource group
   - Choose database name
   - Create/select server

2. Server Configuration:

Server Details (upload.js & database.js):

- Server name: [your-server].database.windows.net
- Admin login: [admin-username]
- Password requirements:
  * Minimum 8 characters
  * Combination of letters, numbers, symbols
  * No login name

Note: for security and scalability, creating a .env file with variables for the server name, database name, username, and password of the database would be best practice. You would then reference these in the database.js and upload.js file like this:

```
// Replace this config object in all relevant files
const config = {
  user: 'YOUR_AZURE_USERNAME',           // Change from 'azureuser'
  password: 'YOUR_AZURE_PASSWORD',       // Change from 'b4y3r83}9'
  server: 'YOUR_SERVER.database.windows.net', // Change from 'bayer-mssql'
  database: 'YOUR_DATABASE_NAME',        // Change from 'mySampleDatabase'
  options: {
    encrypt: true,
    trustServerCertificate: true,
  },
};
```

config.env:

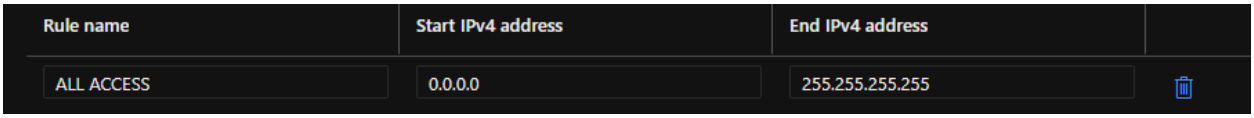
```
DB_USER=YOUR_AZURE_USERNAME
DB_PASSWORD=YOUR_AZURE_PASSWORD
DB_SERVER=YOUR_SERVER.database.windows.net
DB_NAME=YOUR_DATABASE_NAME
PORT=8080
```

3. Firewall Settings:

Configure Firewall Rules:

1. Go to SQL Server → Networking
2. Add client IP address

Example:



3. Enable "Allow Azure services"

Note: installing Microsoft SQL (SMSS) on your computer and then logging into the database from there is a great way of visualizing the table and structure of the database. The Azure website is not as well suited for this but it's mainly preference.

Instructions for starting up the app:

How to start local node server:

1. install node from node.js website go through the install wizard
2. open up project in vscode and ensure you are in the 'project' directory
    - you can do this by typing in ‘cd Project’ in the vs code terminal. Both bash and powershell should work
3. type npm install in vs code terminal
    - this will install all necessary node.js libraries found in the .json file
4. type npm run build in vs code terminal
    - this will compile the code
    - if errors show up because of a unknown module, install that module
5. type npm run server in vs code terminal
    - this will start up the server
6. go to localhost:8080 in browser
7. choose the excel file and hit upload
8. you will see an alert message that the file was uploaded on the website (estimated 10s) – the upload time is dependent on the hardware of your machine since this is running off a local source

Note: upload.js will create the table for you in the database if it doesn’t already exist

---

Public directory:

- The public directory contains static assets and configuration files that are directly accessible by the browser:

index.html: The main HTML file served to clients

Purpose:

- Entry point for the React application
- Provides initial HTML structure

- Contains necessary meta tags and links
- Displays fallback content when JavaScript is disabled

manifest.json: Web app manifest file that defines how the application appears when installed on a device

Purpose:

- Configures Progressive Web App (PWA) behavior
- Defines app icons for different platforms and sizes
- Sets app display properties and colors
- Enables "Add to Home Screen" functionality

Server directory:

upload.js

Handles the upload and processing of Excel files containing survey responses.

Key Features:

- Multiple file upload support (up to 10 files)
- Excel file parsing
- Data sanitization
- Database upsert operations

Main Functions:

- uploadFile(req, res): Processes uploaded Excel files and inserts data into database
    - Generates unique UUID for each upload batch
    - Sanitizes HTML content
    - Validates and formats dates
    - Performs upsert operations to avoid duplicates

server.js

Express server implementation providing API endpoints for the application.

Endpoints:

1. GET /api/data
    - Returns all survey responses
    - Response: JSON array of survey records
2. GET /api/allData
    - Returns complete dataset
    - Response: JSON array of all records
3. GET /api/uploadedData/:uploadId
    - Returns data for specific upload batch
    - Parameters: uploadId (UUID)
    - Response: JSON array of records matching uploadId
4. POST /upload
    - Handles file uploads
    - Accepts: multipart/form-data
    - Returns: uploadId and processed data

database.js

Manages database operations and data retrieval.

Functions:

1. fetchData()
    - Returns all survey responses
    - Formats dates to ISO string
2. fetchAllData()
    - Returns complete dataset
    - Formats dates to ISO string

3. fetchUploadedData(uploadId)
    - Parameters: uploadId (UUID)
    - Returns records for specific upload batch
    - Formats dates to ISO string

Src (Source) directory:

App.css

Global styles for the main application container.

Purpose:

- Defines core application styling
- Handles animations and responsive design

Key Styles:

1. Layout:
    - Center-aligned content
    - Responsive container sizing
    - Flexible component positioning
2. Animations:
    - Logo spin animation
    - Smooth transitions
    - Motion preferences support
3. Theme Elements:
    - Header styling with dark background
    - Link color definitions
    - Font size calculations
    - Responsive spacing

App.js

Main application component that serves as the root of the React application.

Purpose:

- Manages routing between different views
- Handles state management for uploaded data
- Controls data flow between components
- Maintains consistent color mapping across views

Key Features:

1. State Management:
    - uploadedData: Stores currently uploaded file data
    - singleFilteredData: Stores filtered data for single file view
    - allFilteredData: Stores filtered data for all data view
    - globalOrgColorMap: Maintains consistent color mapping for organizations
2. Routing:
    - /: Single file view with uploader
    - /all-data: All data view
3. Components Integration:
    - DataDisplay: Component for displaying and filtering data
    - LineGraph: Displays trend analysis over time
    - BarGraph: Shows comparative data analysis
    - PieGraph: Presents distribution analysis
    - CustomUploader: Handles file uploads
    - Header: Application navigation and branding
4. Data Flow:

- handleUpload: Processes newly uploaded data
- handleSingleFilteredData: Manages single view filtered data
- handleAllFilteredData: Manages all view filtered data
- handleColorMapUpdate: Maintains consistent organization colors

App.test.js

Test suite for the main App component.

Purpose:

- Ensures core application functionality
- Validates component rendering

Testing Features:

1. Component Testing:
    - Renders App component
    - Checks for React elements
    - Validates component presence in document
2. Setup:
    - Uses React Testing Library
    - Implements screen queries
    - Manages component rendering
3. Test Cases:
    - Basic render test
    - Text content verification
    - Document structure validation

Index.css

Global CSS styles applied to the entire application.

Purpose:

- Defines base styling for HTML elements
- Sets up font families and smoothing
- Establishes default margins and spacing

Key Styles:

1. Body Styling:
    - Removes default margin
    - Sets system font stack:
        - -apple-system
        - BlinkMacSystemFont
        - Segoe UI
        - Roboto
        - Oxygen
        - Ubuntu
        - Cantarell
        - Fira Sans
        - Droid Sans
        - Helvetica Neue
    - Enables font smoothing for better readability
2. Code Block Styling:
    - Sets monospace font stack:
        - source-code-pro
        - Menlo
        - Monaco
        - Consolas
        - Courier New

    - Used for any code display within the application
3. Browser Optimization:
    - Webkit font smoothing for Apple devices
    - Mozilla font smoothing for Firefox
    - Cross-browser compatibility considerations

Index.js

Main entry point for the React application.

Purpose:

- Initializes the React application
- Renders the root App component
- Sets up performance monitoring

Key Features:

1. Imports:
    - React core libraries
    - Root App component
    - Global styles (styles.css)
    - Web vitals reporting utility
2. Application Setup:
    - Uses React.StrictMode for development checks
    - Mounts App component to root DOM element
    - Initializes performance monitoring
3. Rendering:
    - Targets 'root' element in index.html
    - Implements strict mode for better development experience
    - Sets up the basic application structure

reportWebVitals.js

Performance monitoring utility for web vital metrics.

Purpose:

- Measures and reports core web vitals
- Tracks performance metrics
- Enables performance optimization

Key Metrics:

1. Core Web Vitals:
    - CLS (Cumulative Layout Shift)
    - FID (First Input Delay)
    - FCP (First Contentful Paint)
    - LCP (Largest Contentful Paint)
    - TTFB (Time to First Byte)
2. Implementation:
    - Conditional loading of web-vitals
    - Performance callback handling
    - Asynchronous metric reporting
3. Usage:
    - Accepts performance entry callback
    - Lazy loads web-vitals library
    - Reports metrics individually
4. Function Parameters:
    - onPerfEntry: Optional callback function for performance metrics
    - Validates callback is a proper function
    - Handles metric reporting through callback

setupTests.js

Configuration file for Jest testing framework.

Purpose:

- Sets up testing environment
- Adds custom Jest DOM matchers
- Enables DOM-specific assertions in tests

Features:

1. Testing Utilities:
    - Imports testing-library/jest-dom
    - Provides DOM element assertions
    - Enables matching text content
2. Custom Matchers:
    - toHaveTextContent
    - toBeInTheDocument
    - toHaveStyle
    - toBeVisible
    - toBeDisabled

styles.css

Global stylesheet for custom components and layouts.

Purpose:

- Defines custom styling for data visualization components
- Manages layout for data containers and buttons
- Implements consistent UI styling across the application

Key Styling Components:

1. Data Container (.data-container):
    - Layout Properties:
        - Width: 94% of parent
        - Centered with automatic margins
        - Relative positioning
    - Display Properties:
        - Hidden overflow
        - Responsive width
        - Auto margins for centering
2. Button Container (.button-container):
    - Layout Properties:
        - Right-aligned text
        - Fixed height of 40px
    - Purpose:
        - Contains view controls
        - Maintains consistent button positioning
3. View Buttons (.view-all-btn, .view-single-btn):
    - Visual Styling:
        - Padding: 8px 16px
        - Background: #007bff (blue)
        - White text color
        - Rounded corners (4px radius)
    - Interactive Features:
        - Cursor pointer on hover
        - Smooth background transition (0.3s)

        - Darker blue (#0056b3) on hover
    - Typography:
        - 14px font size
        - Sans-serif font family
    - Button Structure:
        - No border
        - Text decoration removed
        - Consistent padding

Components directory:

DataDisplay.js

Component for displaying and filtering tabular data with date range functionality.

Purpose:

- Renders data in a paginated table format
- Provides date range filtering
- Handles both single file and all data views
- Manages expandable text cells

Key Components:

1. ExpandableCell Component:
    - Handles long text content (especially comments)
    - Features:
        - Collapsible text (200 character limit)
        - View more/less toggle
        - Preserves line breaks
        - Custom text formatting

        - Maintains proper spacing
2. DataDisplay Component:
    - State Management:
        - displayData: Raw data for display
        - filteredData: Date-filtered data
        - columns: Dynamic column configuration
        - pageSize: Table pagination size
        - dateRange: Selected date range for filtering
3. Data Processing:
    - Functions:
        - processData: Formats raw data for display
        - formatDate: Standardizes date formatting
        - fetchData: API call for single file data
        - fetchAllData: API call for all data
4. Data Grid Configuration:
    - Features:
        - Auto-height rows
        - Customizable page size (5, 10, 20 options)
        - Editable cells
        - Custom styling for density
        - Shadow effects
        - Responsive width
5. Navigation:
    - Toggle between views:
        - Single file view
        - All data view

    - Styled navigation buttons
    - Intuitive UI placement

Props:

- view: 'single' or 'all'
- data: Initial data for single view
- onDataFiltered: Callback for filtered data updates

Dependencies:

- @mui/x-data-grid: For table display
- @mui/material: For UI components
- axios: For API calls
- react-router-dom: For navigation

CustomUploader.js

File upload component with drag-and-drop functionality and progress tracking.

Purpose:

- Handles file uploads with visual feedback
- Supports drag-and-drop and manual file selection
- Manages upload status and progress
- Provides file management interface

Key Features:

1. File Input Handling:
    - Multiple file support
    - Drag and drop functionality
    - File type restriction (.xlsx, .xls)
    - Browse button option
    - Hidden native file input

2. State Management:
    - files: Array of selected files
    - dragActive: Drag interaction state
    - uploadStatus: Upload progress tracking
    - fileInputRef: Reference to file input element
3. Upload Functionality:
    - Progress tracking per file
    - FormData handling
    - Axios upload with progress
    - Error handling
    - Success callback
    - Upload status indicators
4. File Management:
    - Individual file removal
    - Batch file clearing
    - File list display
    - Status indicators:
        - Completed (check mark)
        - Uploading (spinner)
        - Pending (remove option)
5. Visual Features:
    - Drop zone highlighting
    - Progress indicators
    - Status icons
    - Loading animations
    - Responsive layout

Props:

- onUpload: Callback function for successful uploads

Methods:

- handleDrag(): Manages drag events
- handleDrop(): Processes dropped files
- handleChange(): Handles manual file selection
- handleUpload(): Processes file upload
- removeFile(): Removes individual files
- handleClear(): Clears all files

Dependencies:

- axios: For HTTP requests
- bootstrap-icons: For UI icons
- CustomUploader.css: For styling

CustomUploader.css

Styling for the file upload component with a clean, modern design.

Main Components:

1. Container Styling (.custom-uploader):
    - Width: 100% (max 500px)
    - Centered layout
    - Font: Arial/sans-serif
    - Margin bottom: 20px
2. Upload Content (.uploader-content):

- White background
- Rounded corners (8px)
- Soft box shadow
- Padding: 20px
- Hidden overflow

3. Drop Zone (.drop-zone):
    - Dashed border (2px)
    - Centered text
    - Light background
    - Interactive cursor
    - 20px padding
    - 4px border radius
4. File List (.file-list):
    - Light gray background (#f0f2f5)
    - Rounded corners
    - Border separators
    - Custom styling for:
        - Single file
        - First file
        - Last file
5. Interactive Elements:
    - Browse Button:

        - White background
        - Gray border
        - Hover effect
    - Upload Button:
        - Blue background (#007bff)
        - White text
        - Darker on hover
    - Clear Button:
        - Text-only style
        - Blue color
        - Underline on hover
6. Status Indicators:
    - Loading Spinner:
        - 2px border
        - Blue accent
        - Continuous rotation animation
    - Remove Button:
        - Red icon
        - Hover fill effect
        - Smooth transition

Daterange.js

Date range picker component for filtering data by date intervals.

Purpose:

- Provides date range selection functionality
- Handles UTC date normalization
- Enables date-based data filtering
- Offers clear date selection option

Key Features:

1. Date Picker Components:
    - Start date selector
    - End date selector
    - Clear button
    - UTC-aware date handling
2. State Management:

    const [startDate, endDate] = dateRange;

    - Controlled through parent component
    - Maintains UTC consistency
    - Handles null states
3. Date Handling Functions:

    handleStartDateChange(): Normalizes start date to day start

    handleEndDateChange(): Normalizes end date to day end

handleClear(): Resets date selections

4. Layout Structure:
    - Flex container layout
    - Responsive design
    - Consistent spacing
    - Vertical alignment
5. Styling Features:
    - Material-UI components
    - Responsive gaps
    - Flexible wrapping
    - Consistent margins
    - Top margin offset

Props:

- dateRange: Array containing [startDate, endDate]
- setDateRange: Callback for updating date range

Dependencies:

- @mui/x-date-pickers: Date picker components
- @mui/material: UI components
- dayjs: Date manipulation
- dayjs/utc: UTC support

Usage Example:

<DateRange

```
  dateRange={[startDate, endDate]}
  setDateRange={handleDateRangeChange}
/>
```

GraphWrapper.js

Component wrapper for graphs with download functionality.

Purpose:

- Provides consistent container for graph components
- Enables graph image download functionality
- Maintains uniform styling across graphs
- Wraps content in Material-UI Paper component

Key Features:

1. Graph Container:
    - Paper component wrapper
    - Consistent margins and padding
    - Light gray background (#f5f5f5)
    - Centered content alignment
2. Download Functionality:

    handleDownload():

    - Captures graph content as image

    - Converts to PNG format

- Triggers automatic download

- Uses custom filename

3. Styling:
    - Outer Container:
        - 20px margin
        - 20px padding
        - Outlined variant
        - Light gray background
    - Inner Container:
        - Light border
        - 10px padding
    - Download Button:
        - No text transform
        - 10px top margin
        - Primary color
4. Reference Handling:

const graphRef = useRef();

    - Maintains reference to graph content
    - Used for image capture

Props:

- children: Graph component to be wrapped
- title: Optional title for downloaded file

Dependencies:

- html2canvas: For graph capture
- @mui/material: For UI components (Paper, Button)

Usage Example:

```
<GraphWrapper title="MyGraph">
  <LineGraph data={data} />
</GraphWrapper>
```

Header.js & Header.css

Responsive header component with scroll-based size adjustment.

Purpose:

- Displays application title and logo
- Provides sticky header functionality
- Implements smooth size transitions on scroll
- Maintains consistent branding

Key Components:

1. Header Component (Header.js):

   const [isShrunk, setIsShrunk] = useState(false);

   - Scroll state management
   - Sticky positioning
   - Logo integration

- Placeholder for scroll compensation

2. Scroll Handling:

   handleScroll():

   - Monitors window scroll position

   - Triggers header shrinking at 50px scroll

   - Cleanup on unmount

3. Styling (Header.css):
   - Base Header:
     - Font: Neue Helvetica/Arial
     - Color: #10384f
     - Initial font size: 26px
     - White background
     - Centered text
   - Sticky State:
     - Fixed positioning
     - Top layer (z-index: 1000)
     - Soft shadow
     - Reduced font size (20px)
   - Logo Styling:
     - Vertical alignment
     - 3px bottom offset
     - 8px right margin

        - Fixed dimensions (27x27)

4. Transitions:

   transition: all 0.3s ease;

    - Smooth size changes
    - Fluid position adjustments
    - Shadow transitions

Line Graph Documentation

File Structure:

```
src/
├── components/
│   └── LineGraph.js      # Main graph component
└── utils/
    └── LineGraphConfig.js # Configuration and data processing
```

Overview:
Interactive line graph component for visualizing satisfaction ratings over time with organization filtering and time frame controls.

LineGraphConfig.js (Utils)

Core Functions:

1. Time Frame Configuration:

   timeFrameOptions = [

    'daily', 'weekly', 'biweekly',

    'triweekly', 'monthly', 'custom'

]

2. Data Processing:

   processSelectedOrgs(): Filters chart data by selected organizations

   DefaultSatisfactionGraph(): Main data processing function

3. Data Aggregation Features:
   - Time-based grouping
   - Organization filtering
   - Data averaging
   - Date normalization

LineGraph.js (Component)

Features:

1. Visual Components:
   - Interactive line chart
   - Organization selector
   - Time frame controls
   - Download functionality
2. Color Management:
   - Dynamic color palette generation
   - Color persistence
   - Color shifting algorithm
3. Interactive Controls:

   // Time Controls

   timeFrame selection

   customDays input

   // Organization Controls

Multi-select dropdown

Select All functionality

4. Styling:
    - Responsive layout
    - Material-UI components
    - Custom containers
    - Download button

Props:

```
{
 data: Array,         // Input data
 orgColorMap: Object,   // Organization color mapping
 onColorMapUpdate: Function // Color update callback
}
```

Key Methods:

1. Data Handling:

   handleOrgChange(): Manages organization selection

   generateColorPalette(): Creates color schemes

   shiftColor(): Modifies colors for uniqueness
2. Visual Features:

   handleDownload(): Exports graph as PNG

   renderValue(): Formats selected organizations
3. Chart Configuration:
    - X-axis: Time-based with date formatting
    - Y-axis: 0-5 satisfaction rating scale
    - Tooltips with value formatting
    - Legend positioning

- Line styling

Pie Graph Documentation

File Structure:

```
src/
├── components/
│   └── PieGraph.js       # Main graph component with all functionality
└── utils/
    └── PieGraphConfig.js  # (Unused) Initial configuration attempt
```

Overview:
Interactive pie chart visualization for organization ratings distribution with comparison capabilities.

PieGraph.js (Component)

Core Features:

1. Main Visualization:

   // Primary pie chart showing organization distribution

   PieChart with:

   - Organization percentages

   - Interactive segments

   - Custom color mapping

   - Outside labels

2. Comparison Tool:

   // Dual pie charts for organization comparison

   Features:

   - Organization selection dropdowns

   - Rating distribution breakdown

- Side-by-side visualization

3. Data Processing:

   processData(): Main data transformation

   processRatings(): Rating breakdown calculation

   handlePieClick(): Click interaction handler

4. Color Management:

   // Color system

   baseColors: Dutch Field palette

   generateColorPalette(): Dynamic color generation

   shiftColor(): Color modification algorithm

   getOrgColor(): Color assignment handler

State Management:

```
{
  data: [],           // Processed chart data
  selectedOrg1: '',    // First comparison org
  selectedOrg2: '',    // Second comparison org
  filteredData1: [],   // First org breakdown
  filteredData2: [],   // Second org breakdown
  gridData: []         // Detailed data for grid
}
```

Visual Components:

1. Main Container:
   - Light gray background
   - Rounded corners
   - Soft shadow
   - Responsive width

2. Chart Container:
    - White background
    - Border radius
    - Shadow effects
    - Fixed height (510px)
3. Interactive Elements:
    - Organization dropdowns
    - Download button
    - Clickable pie segments

Props:

```
{
  data: Array,         // Input data array
  orgColorMap: Object    // Organization color mapping
}
```

Key Methods:

1. Data Handling:

   handleOrg1Change(): First org selection

   handleOrg2Change(): Second org selection

   handleDownload(): PNG export
2. Data Processing:

   processData(): Raw data transformation

   processRatings(): Rating calculations

Note about PieGraphConfig.js

While present in the utils directory, this file is not actively used as all functionality was consolidated into the main PieGraph.js component. The configuration approach was abandoned in favor of a more integrated solution.

Package Configuration Documentation

package.json

Project Details:

```
{
  "name": "file-upload-app",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://localhost:8080"
}
```

Key Dependencies:

1. UI Framework:

   ```
   {
     "@emotion/react": "^11.13.3",
     "@emotion/styled": "^11.13.0",
     "@mui/material": "^6.1.3",
     "@mui/x-charts": "^7.20.0",
     "@mui/x-data-grid": "^7.21.0",
     "@mui/x-date-pickers": "^7.22.0"
   }
   ```

2. Core React:

   ```
   {
     "react": "^18.3.1",
     "react-dom": "^18.3.1",
     "react-router-dom": "^6.27.0",
     "react-scripts": "5.0.1"
   ```

}

3. Data Handling:

{

"axios": "^1.7.7",

"dayjs": "^1.11.13",

"mssql": "^11.0.1",

"mysql": "^2.18.1",

"xlsx": "^0.18.5"

}

4. Utilities:

{

"html2canvas": "^1.4.1",

"sanitize-html": "^2.13.1",

"express": "^4.21.1",

"multer": "^1.4.5-lts.1"

}

Scripts:

{

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject",

"server": "node server/server.js",

"dev": "concurrently \"npm run server\" \"npm start\""

}

package-lock.json Overview

Purpose:

- Ensures consistent installation of dependencies
- Locks exact versions of installed packages
- Records dependency tree
- Guarantees same dependencies across all development environments

Key Aspects:

1. Version Control:
    - Contains exact versions of all packages
    - Includes nested dependencies
    - Records integrity hashes
2. Structure:

```
{
 "name": "file-upload-app",
 "version": "0.1.0",
 "lockfileVersion": 2,
 "requires": true,
 "packages": {
  // Detailed dependency tree
 }
}
```

3. Security:
    - Includes integrity checksums
    - Prevents malicious package injection
    - Ensures dependency consistency
4. Benefits:
    - Reproducible builds

- Deterministic installations
- Conflict prevention
- Version consistency

5. Maintenance:
    - Automatically generated
    - Should be committed to version control
    - Updated with npm install/update
    - Contains complete dependency resolution

Future Plans:

See future_plans.png in the Github: https://github.com/EAST-Dev-Group/EmployeeAnalyticalSurveyTool