# Software Design Plan

Written By: John McGinnes

## A. Business Case

### 1. Problem Statement

There is currently an issue with the web application used by Endothon Finance. This web application is intended to gather financial information on the most recent five fiscal years of the business. It is currently gathering data on the first five fiscal years that the business was in operation. A software bug is causing the app to request the wrong fiscal years, it is not accounting for the business's age in relation to the current year.

With the current functionality, if the data was gathered from a business established in 2000, it would return data from 2000 through 2004. The intended functionality for the same company would be to gather data from 2018 through 2022 (though it is currently 2025, the example given uses 2023 as the current year).

The application should identify the correct years based on when the business was established. If the business is less than five years old, the app should forecast future financial data based on what is available to still provide five years of data.

### 2. Business requirements

The business requirements for the web application are as follows:

1. The app should request the most recent five years of financial information, excluding the current year, for any business older than five years.
2. If a business is less than five years old, the app should gather as much information as possible on previous years and use forecasted financial data to fill in up to five years.

The app currently fails to meet these two requirements as it is requesting data from the first five years of business operations rather than the most recent five years (for point one above). It is also unable to gather the forecasted data for any business less than five years old due to the software bug (failing to meet the standards for point two above).

### 3. In-scope action items

Given the business requirements listed above, the following items would be considered part of the scope of this issue:

1. Update the program's logic to correctly gather data on the most recent five fiscal years (not including the current year) for any business older than five years. This is one of the primary functions of the application and would directly fulfill the requirements for the first item listed in the business requirements above.
2. Correct the program's logic to forecast future financial data for any business that is less than five years old. As the total data will need to span five years, this will fill in the timeline for any business that does not have enough data. This will directly address the second business requirement listed above.
3. Use test cases to verify the data for both primary scenarios listed in the business requirements. Once the logic has been updated, testing the cases where a business

is older than five years or younger than five years will align with the goals of this software fix to be sure the data is accurate for the timeline needed.

4. Implement error handling for cases when the business is less than five years old to correctly trigger the logic to forecast the data. This will help to meet the needs of the second business requirement, as well as maintain the five full years of data regardless of the age of the business.

## 4. Out-of-scope action items

One potential out-of-scope item for this issue would include requests to redesign the user interface for the application. While a better user interface might make data entry and retrieval easier for the users, the current problem is related to a bug in the programming logic, this would not help to solve or test that concern.

A second out-of-scope item may be adding new features to the application workflow. While new features like adding additional questions may enhance the user experience, it would not be directly related to the current software bug concerns.

## B. Requirements

### 1. Functional requirements

Functional requirements for this specific request would include the following:

1. The application should be able to collect the most recent financial data over the course of the last five years, excluding the current year. Because this is the most relevant data, it is a core requirement of the software functionality.
2. For any business that is less than five years old, the app should gather what years are available and fill in the remainder of the data requirement with forecasted financial data. This will address the issue listed in the ticket where the forecasted data is not populating correctly in the program.
3. The web application must use the corrected logic to populate the loan profiles for both the newer and older businesses. This will ensure that the current information is correct and address the software bug where the incorrect logic is causing the wrong data to be added.
4. The user interface must also contain proper fields to contain the database on the age of the business. This will allow the users of the software to view the data once the bug has been corrected.

### 2. Non-functional requirements

Non-Functional requirements that will be considered would be:

1. The web application must be able to retrieve the financial data within 2 seconds. This requirement will ensure a smooth experience for the user. It is critical to the application because longer delays could cause user frustration or issues with loan processing.
2. The app code should also be built with a modular structure in mind. This will make future updates easier as well as troubleshooting. This is necessary in most corporate applications in case the logic requires further enhancements or adjustments.

## C. Software Design

### 1. Software behavior

Each of the following would be classified as an event or input of the software operation:

1. Input: Date of Business Establishment
    a. Intended Response: The application requires a complete five-year profile for each business; thus, it would need input to determine the date that the business was created. If the business is older than five years it will gather the most recent five years of financial data, if it is younger than five years it will fill in the data using forecasted financial data.
    b. Associated Constraints: When entering the business date, the format would be important and must be valid. Using an "MM/DD/YYYY" date is standard in the United States. The app would also need to correctly handle leap years and similar edge cases.
2. Input: Uploading or Gathering Financial Data:
    a. Intended Response: The application will need to allow the users to upload the data from the requested years and provide validation that the years are correct. It will need to display an error if the data is incorrectly formatted.
    b. Associated Constraints: The application would need a list of acceptable formats for the uploaded data (CSV, Excel, etc.). Within the data, specific columns may be required or there may be constraints within the columns, like numeric values only.
3. Input: User Authentication:
    a. Intended Response: As financial data is sensitive, users will have to log in to access the information. This log-in process will need to verify that the user has a valid account and the proper credentials. An error message will need to be displayed if the username or password is incorrect.
    b. Associated Constraints: Passwords will need to meet security standards and the user must be registered with the application using a valid email address before attempting to log in.
4. Event: Loan Application Submission:
    a. The web application will require input validation that all the required fields have been completed with the correct formatting for dates, contact information, and numeric fields. It will need to display error messages if information is missing.
    b. Associated Constraints: The application for the loan will have required fields that will need to be completed before they are allowed to apply.

### 2. Software structure

The ideal structure for the web application would be a modular architecture. This type of architecture would work well for maintenance and scalability by dividing the software into separate modules that are each responsible for a part of the overall functionality. The modules could easily be segmented into the following categories:

- User Interface Module: The primary form of user interaction, this module will include displaying the loan application and the input fields, as well as providing input validation for any items needed for the loan considerations.
- Authentication Module: This module would manage the user authentication concerns and account verification needed to access the application.

- Logic Module: This piece of the app would hold the logic for calculating the age of the business and determining whether historical data is needed or forecasted data to fill the five-year profile.
- Loan Profile Module: This module will create and store information related to the overall loan profile and compile the financial data for each business.
- Data Processing Module: The final piece of the program would handle the storage and retrieval of the financial data, for both the historical and forecasted options.

A modular structure is easily justified in this instance as it provides an easy-to-maintain codebase for troubleshooting, convenient flexibility that can allow code to be tested without disrupting the overall functionality, and each module would be self-contained and reusable if needed.

## D. Development Approach

### 1. Planned deliverables

Planned deliverables for this endeavor would include:

1. The Financial Data Retrieval Module
   a. This module would be responsible for the logic related to retrieving the correct financial data for the given timeline, whether historical or forecasted.
      i. Step 1: Define the Data Logic. Begin by implementing the logic to determine the business age and identify what the balance will be of historical and forecasted data.
      ii. Step 2: Integrate the Data. Create a process to upload the financial data from the user.
      iii. Step 3: Validation. Verify that the provided data meets the format and structure needed.
      iv. Step 4: Testing. Complete various tests with each case that could be accepted by the program to be sure the correct data is retrieved.
      v. Step 5: Integration. Integrate this module with the other pieces of the web application, ensuring the functionality is stable.
2. Loan Profile Generation Module
   a. This module will create a loan profile for each business based on the financial data that has been provided. It will verify that the correct data is used for the profile.
      i. Step 1: Define Profile Structure. Confirm the necessary fields and format for the profile.
      ii. Step 2: Data Integration. Retrieve the relevant financial data from the Data Retrieval Module and use it to populate the profile information.
      iii. Step 3: Error Handling. Implement checks to be sure the required fields are filled, and the formatting is correct.
      iv. Step 4: Testing. Move through the profile generation process with multiple businesses to be sure the data is complete and correct for the five years.
      v. Step 5: Integration. Verify that the loan profile can be shared with lending vendors without issues.
3. User Interface for Loan Application Form.
   a. A user-friendly interface will be needed to display the information that is intended to be visible to the user, especially where input is required.
      i. Step 1: UI Design. Create the layout for the application form, ensuring that the interface is clear and easy to follow.
      ii. Step 2: Data Display. Implement the behavior to display the financial data based on the business creation date.
      iii. Step 3: Validation. Add input validation to be sure users can only submit correctly formatted data.
      iv. Step 4: Testing. Test the UI in different scenarios to ensure functionality.
      v. Step 5: User Feedback. Provide feedback to the user in the form of error messages during incorrect data entry or formatting.
4. Documentation for Data Logic and User Instructions
   a. Documentation will be needed to explain the logic and user instructions for operating the application.

i. Step 1: Create Documentation. Document the algorithm used and the conditions for determining the years to request for the different business ages.
ii. Step 2: Create User Instructions. Provide step-by-step instructions for the users to follow when filling out the application form.
iii. Step 3: Testing and Review. Review the documentation with the developers and end users to ensure accuracy and clarity.
iv. Step 4: Distribute Documentation. Be sure the documentation is made available to the developers and the end users in an easy-to-find location.

## 2. Sequence of deliverables

The most logical sequence of deliverables would be as follows:

1. User Interface for Loan Application Form.
    a. This deliverable should be implemented first because it provides the foundation for user interaction. This form will determine what data needs to be collected from the users (specifically business creation date and financial data). Without the user interface, the app would not be able to gather or display loan profile data.
2. Financial Data Retrieval Module.
    a. This module would be the next step as it handles the core logic for determining which data needs to be requested based on the age of the business. This module will process the data collected from the user interface. Without this piece, the application will not know which data to request or when to implement the forecasted data.
3. Loan Profile Generation Module.
    a. Once the data retrieval module is in place, the Profile Generation Module can be implemented. This module will take the data provided by the conditions of the previous modules and build the loan profile to be submitted to the lending vendors. The loan profile depends on the correct data being retrieved, so it would not be sensible to begin development before the other processes are completed.
4. Documentation for Data Logic and User Instructions.
    a. The final remaining deliverable would be the documentation. At this point, all the components should be complete. The documentation will provide the developers and users with guidelines and best practices. This will need to be developed last as some information may need to be added for each module throughout the implementation to form the complete documentation.

## 3. Development environment

For the front-end development, the best language to use would be JavaScript. This would provide an excellent basis for the user interface needed for the loan application form. I would also recommend the React library as this allows UI components to be rendered dynamically and efficiently, including form fields for user input. Visual Studio Code would be the preferred IDE as it provides support for JavaScript with React.

When it comes to the back-end development, Node.js (JavaScript) would be well-suited to handling the data retrieval and processing. Using JavaScript in both the front and back-end development would also simplify the process and allow the same IDE to be used for this step, Visual Studio Code. It would be ideal to include the Express.js library to create

RESTful APIs that will coordinate between the front-end and the back-end, as well as the Oracle Database, which would be excellent for storing financial data.

GitHub should also be used for version control and change tracking, which will allow sharing of the codebase with the team. For testing, Jest is a helpful tool for performing unit testing on JavaScript programs, as well as using Postman to test the RESTful API endpoints developed with Express.js. These tools allow for troubleshooting and debugging to be sure the data is handled correctly and without errors.

## 4. Development Methodology

The agile development methodology would work best for this implementation. This methodology will allow for iterative development and frequent feedback, which is crucial for addressing software bugs and requirements.

The sequence of deliverables listed in section D2 would easily support an agile methodology as each step builds on the next, starting with the user interface, followed by the data retrieval module, profile generation, and documentation. This will allow for feedback and adjustments at each stage.

Agile will also allow the team to adapt to changing requirements or issues as they arise without delaying the overall project. Each deliverable can be refined based on user feedback.

In comparison to the Waterfall methodology, which is better suited to projects with a well-defined set of unchanging requirements, agile development will allow the software issues to be fixed as soon as they arise with multiple testing throughout the process. Waterfall is not an ideal methodology in these conditions.