**Software Design Plan**

# A. Business Case

## 1. Problem Statement

The new web app pulls the first five fiscal years of financial data after creating a business. That’s a problem because it’s supposed to pull the five most recent completed years, not the earliest. So, if a company started in 2000, the app grabs data from 2000 to 2004, but it *should* be grabbing 2018 to 2022 (assuming it’s 2023 now). That breaks the whole point of getting up-to-date info for loan profiles.

## 2. Business requirements

* The app should always pull the five most recent completed fiscal years of financial data and not include the current year.
* If the business is under five, the app should pull all the available past years and then ask for projected (forecasted) financial data for the remaining years to total five.

Right now, the app doesn’t follow either of those rules. It starts from year one of the business and goes forward, even if that data is old and irrelevant. That does not help make loan decisions today.

## 3. In-scope action items

* Update the logic to pull the most recent five years of data, matching the first business requirement.
* Add a condition for newer businesses that don’t have five years of history, which matches the second requirement.
* Generate forecast years when needed so the total adds up to five, supporting the second requirement.
* Write unit tests to ensure the new logic works correctly and handles edge cases.

## 4. Out-of-scope action items

* Redesigning the loan profile UI, I get that it shows the data, but this bug is all about the backend logic. The UI isn’t the issue here.
* Adding or changing lender API integrations, lenders use this info, but that’s a separate system. This task is only about fixing how we pick the right financial years.

# B. Requirements

## 1. Functional requirements

* Retrieve the five most recent completed fiscal years for businesses that are 5+ years old (no current year).
* Detect when a business is under 5 years old and determine how many years of real data it has.
* Request projected financial data for however many future years are needed to hit five.
* Store and display the selected years in the loan profile to use the logic output downstream adequately.

## 2. Non-functional requirements

* The logic should respond fast, even when users submit data from slower devices. It shouldn’t add noticeable delay to the loan application process.
* The logic should be testable and maintainable, with clear conditions and output, so future developers (or even me later) can understand and adjust it if needed.

# C. Software Design

## 1. Software behavior

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Event | App Response | |  |  | **Constraints** | | --- | --- | --- | |
| Business age is 5+ years | Pull 5 most recent completed years of financial data | Excludes current year. Uses system year to calculate range. |
| Business age is less than 5 years | Pull available years and request forecast data for the missing years | Forecast only goes up to make a total of 5 years. |
| User submits company creation year + current year | System runs logic to figure out what years to request | Year values must be valid numbers and not future-dated. |
| Year logic runs | Output list of correct 5 years, marked as historical or forecast | Year values must be valid numbers and not future-dated. |

## 2. Software structure

* A function or class to calculate the five years (can take the business start year + the current year as input).
* A separate helper method to decide if forecast years are needed and how many.
* Keep each piece modular so it can be reused or swapped out later.

# D. Development Approach

## 1. Planned deliverables

* **Fiscal year calculation function**

**Steps:** Write the function, add conditionals for age check, and handle both paths (5+ years and under 5).

* **Forecast logic handler**

**Steps:** If the business is < 5 years old, this calculates how many years of forecasts are needed.

* **Unit tests for the year logic**

**Steps:** Write test cases using dummy inputs for different edge cases (new businesses, old ones, invalid data).

* **Documentation/comments**

**Steps:** Inline comments and a short doc page explaining logic and where it’s used in the app.

## 2. Sequence of deliverables

1. First, write and test the year calculation function since everything depends on whether that is correct.
2. Then build the forecast handler, which will only be called sometimes.
3. After that, write the unit tests, testing typical and edge case behaviors.
4. Finally, finish up with internal documentation once it’s all working.

This order works best because the forecast handler depends on the year function, and both need to be done before writing tests or docs.

## 3. Development environment

|  |  |
| --- | --- |
| Tool | Why I’m Using It |
| Java | The backend for the app is Java-based (or assuming it is for this task). |
| Intellij IDEA | For writing, testing, and debugging the logic. |
| JUnit | For unit testing the year/forecast logic. |
| Git | For version control and tracking changes during development. |

## 4. Development Methodology

I’m going with Agile, and here’s why:

* Agile lets me build and test small chunks at a time, which is perfect for this logic fix.
* I can test each part (year calc, forecast logic, tests) as I build, then quickly fix issues if anything breaks.
* I didn’t choose Waterfall because this isn’t a long, step-by-step development project; it’s more of a quick, iterative fix that benefits from testing as I go.