Software Design Plan

WGU Student ID	kbui4

A. Business Case

1. Problem Statement

The web application is requesting the wrong fiscal data for business older than 5 years. It is requesting the *first* 5 fiscal years of a business' financial data, rather than the *latest* 5 years. Ticket #D480-AEN1 summarizes this issue, and provides an example case: For a business established in the year 2000, the web application is requesting financial data for the fiscal years of 2000-2004, rather than the expected 5 years of data (2018-2023).

2. Business requirements

Generating Loan Profiles for Well Established Businesses

Endothon Finance needs to be able to generate loan profiles for well established businesses. In order to do this, the application must collect the business' past 5 years of financial data in order to construct the loan profile.

Generating Loan Profiles for Newer Businesses

Endothon Finance also needs to be able to generate loan profiles for less established businesses. In order to accomplish this, the application must collect all financial data the business has, and then prompt them to provide projections for the remaining years to fill out a 5-year fiscal map for the business, and then generate the loan profile off of that.

The problem

The web application is prompting applicants for the *first* 5 years of fiscal data of a business' history, rather than the latest 5. While the application meets the requirement for newer businesses, it is failing to meet the requirement for well-established ones.

3. In-scope action items

- Examine the business logic for a business established more than 5 years ago. This will meet the requirement that this web application generates proper loan profiles for well-established businesses.
- Double check the business logic for a business established within the past 5 years. This will ensure that the application continues to generate proper loan profiles for newer businesses.

4. Out-of-scope action items

- Automating the fiscal projections for a newly established business. Some businesses might want the application to automatically generate fiscal projections to fill out their loan profile. This is somewhat related to the ticket, as it relates to inputting fiscal data, but is not something we are prioritizing at the moment.
- Redesigning the the loan application form. Some users may want the loan application form to have a different look or feel, or display differently on various devices. While this is also tangentially related to the ticket, as it pertains to the loan application process, we are prioritizing collection of proper data at the moment, and will not be addressing this issue.

B. Requirements

1. Functional requirements

The main functional requirement that needs to be addressed is the gathering of fiscal data. The web form is requesting fiscal data from the first 5 years of a business' history, when it needs to be requesting fiscal data from the latest 5 years.

A tangentially related functional requirement is the generation of the loan profile using the gathered data. Since the web form is requesting the incorrect data, the profiles being generated are invalid.

2. Non-functional requirements

One non functional requirement related to the ticket would be proper implementation of security, authentication, and authorization. Loan applicants need to have their applications and data isolated from other applicants, to respect their data privacy. Furthermore, our servers need to be secured against loan applicants somehow accessing admin only portions of the page and performing unauthorized actions, like altering the generated loan profiles directly.

Another non functional requirement related to the ticket would be performance. If the form takes longer than 5 seconds to load, this will aggravate potential applicants and possibly even mislead them into thinking the site is down. Though not directly related to the ticket, ensuring the application remains performant will ensure a smooth and reliable data collection process.

C. Software Design

1. Software behavior

One input that needs to be examined is the date picker that allows the applicant to select their established business date. This control is expected to accept a date via either direct text entry, or selecting the date via the pop-up calendar that appears when the input field is focused. Constraints on this input are that the date selected must be before the current date. Dates in the future are rejected, as this application is for existing businesses.

Once a valid business established date is selected, 5 more input fields for historical or projected fiscal data appear. It is expected that these 5 fields prompt the applicant to enter the fiscal data for the past 5 years for established businesses. For businesses younger than 5 years, the fields should prompt the applicant to enter fiscal data for all years that the business has existed, as well as fiscal projections into the future for the remaining years required to fill out the 5 year profile. One constraint on these controls is that they do not appear until the business established date is set, as that dictates the nature of the controls displayed.

2. Software structure

As mentioned previously, the page itself is a razor page. The identity of the logged in applicant is accessible via the login service, which is interacted with elsewhere in the site (mainly the login page), stores the user identity, and is injected into the page via dependency injection, giving the page access to the identity of the logged in user.
The loan application page itself is defined as a single cs class that inherits from PageModel with its definition split between two partial classes, one contained in a .razor.cs file containing pure cs code, and another contained in a .razor file which is composed of razor

syntax: a unique blend of html and embedded cs that enables the dynamic behavior described in section C1.

D. Development Approach

1. Planned deliverables

The main deliverable is an updated method that is triggered when the established business date is set. This method takes in the date the business was established, calls methods to construct the 5 controls regarding fiscal data, and then fires a state change event for the page, which triggers a partial page re-render to display the newly constructed controls. The deliverable will be generated by examining and updating the method to ensure that it is using the correct years to dynamically construct the input controls.

Another deliverable will be the method that actually constructs a fiscal input field. It is possible that the main event handler is calling the method correctly, but that this method is handling the incoming data incorrectly. The deliverable will be created by examining the logic of this method will be examined and updated if necessary.

The previous 2 methods exist in the .razor.cs code-behind file for the loan application page. The html template is defined in the .razor file for the loan application page. This is entire file is a 3rd deliverable for the software update, and will be generated by inspecting the razor code and updating if necessary.

2. Sequence of deliverables

The sequence of deliverables should be implemented in the following order:

1. Fiscal input field constructor
2. Established business date set event handler
3. Razor page template (.razor file)

First, we will to implement the method that constructs a single fiscal input field. If it is not generating the input field as expected from the given parameters (year to prompt for, prompt for projection / historical data), it should be updated first

Second, we will implement the event handler that fires when the business established date is set. We will check how it calls the fiscal input field constructor after processing the business established date, and modify it if necessary.

Finally, we will double check the html template for the loan application page, and verify that it is displaying the expected controls properly. If the controls being displayed are still incorrect, we will modify and implement the template itself.

By ordering the development of deliverables in this manner, we will examine the code flow from the lowest level first and then move up the chain of abstraction. This will give us a clear picture of what exactly is going on in the software at each stage of development. Going in the reverse order (from the top down), we would observe the unwanted behavior, but will not have a clear idea of where the bug is until digging down further into the code.

3. Development environment

Development Languages

The languages employed during development will be razor markup and c sharp. With the exception of Javascript and Typescript, these are the only languages supported for Blazor applications, which our application is built on. We do not employ any custom Javascript or Typescript components, so those languages are unnecessary for our environment

IDE
Microsoft Visual Studio will be the primary IDE for development of this update. It has the most feature rich support and tooling for Blazor, and is the obvious choice. Developers running on Linux or Mac may also use JetBrains Rider as an alternative. It is fully cross platform and supports Blazor development. Either IDE will produce code that is compatible with the other, so both are valid choices.

Dependencies
The application uses an OIDC authentication Nuget package to integrate with our organization's identity provider. This allows us to easily add authentication and authorization security to the application. The package itself is already defined in the .csproj file, and is automatically downloaded and installed, so no deliberate action is necessary to get the OIDC integration working properly.

4. Development Methodology
As this is a small bug fix with a very narrow scope, we will be using the Agile method to complete this update.

Our first iteration will implement the first deliverable (the function that generates a single fiscal data input field). If the method needs updating, we can quickly deploy a fix and test the application again.

The next iteration will implement the second deliverable (the event handler that triggers when an established business date is selected), if necessary, the handler will be examined and updated if necessary. The application can again be deployed and tested to see if this resolves the issue.

If the issue still is not resolved, we can move to the final iteration to implement the third deliverable (the razor page and html template). The template and code bindings will be examined and updated if necessary, and the application deployed and tested again. At this point the issue should be resolved. If the issue is still not resolved, we will have to brain storm and reassess where the underlying bug could be.

Agile was selected over alternatives like Waterfall for its faster development lifecycle and flexibility. Because the scope of the issue is extremely narrow, there is no need to wait for a formerly documentation laying out requirements and design before beginning development. The requirements are already given in the issue ticket, and the design for the application is already complete.