

Designing an Interactive Interface for FACET:

Personalized Explanations in XAI

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Computer Science and Data Science

By:

Katharine Dion
Belisha Genin
Randy Huang
Alexander Pietrick
Jacob Reiss

Date:

3/1/2024

Project Advisors:

Dr. Elke Rundensteiner
Peter VanNostrand
Dennis Hofmann
Worcester Polytechnic Institute

This report represents work of one or more WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on the web without editorial or peer review. For more information about the projects program at WPI, see <http://www.wpi.edu/Academics/Projects>.

Abstract

In response to the trend of integrating machine learning into critical decision-making processes, explainable AI methods like counterfactual explanations have emerged. However, existing approaches often neglect user preferences and lack actionable solutions. To address this, a novel framework, FACET (Fast Actionable Counterfactuals for Ensembles of Trees), was developed to offer personalized and robust counterfactual explanations for situations like bank loan applications, ensuring explanation robustness and providing users with actionable solutions. An interactive interface was developed for FACET, empowering users to adjust constraints and prioritize preferences for enhanced user experience. The application was tested to ensure usability across various scenarios and datasets.

Authorship Table

Section	Author(s)	Editor(s)
Abstract	B.G.	J.R.
1.0 Introduction	B.G	K.D.
2.0 Background		
2.1 What is FACET	R.H.	K.D. J.R.
2.2 User Interface Research	K.D.	B.G., J.R., R.H.
2.3 Preliminary FACET User Interface Prototype	A.P.	K.D., J.R., R.H.
3.0 Methodology		
3.1 Objectives	J.R.	R.H.
3.2 Application Workflow	R.H., J.R.	J.R.
3.3 User Interface	K.D.	B.G.
3.4 Frontend	ALL	ALL
3.4.1 Welcome Screen	A.P., B.G.	B.G., J.R.
3.4.2 My Application	K.D.	A.P., B.G.
3.4.3 Feature Controls	B.G	J.R.
3.4.4 Explanations	R.H.	B.G., J.R.
3.4.5 Suggestions	A.P.	B.G., R.H.
3.4.6 Scenarios	R.H.	B.G.,
3.4.7 Application Styling	J.R.	B.G., R.H.
3.5 Backend	R.H., J.R.	
3.5.1 Initialization	R.H.	B.G., J.R.
3.5.2 Integration with Frontend	R.H.	B.G., J.R.
3.5.3 RESTful API Endpoints	R.H.	J.R.

Section	Author(s)	Editor(s)
3.5.4 Explanation Generation Process	R.H.	J.R.
4.0 Results		
4.1 Testing Plan	J.R.	B.G.
4.2 Testing Procedure	J.R.	B.G.
4.3 Testing Results	J.R.	B.G.
5.0 Conclusion		
5.1 Future Work	B.G., J.R., R.H.	B.G., J.R.
5.1.1 Accessibility Styling	J.R., K.D.	B.G.
5.1.2 User Testing	B.G.	J.R.
5.1.3 Feature Filter	R.H.	J.R.
5.1.4 Scenario Comparison	B.G.	J.R.
5.1.5 Explaining App Functionality to Users	B.G.	J.R.
5.1.6 Loading Categorical Data	B.G.	J.R.
5.2 Conclusion	J.R.	B.G.

Table of Contents

Abstract.....	2
Authorship Table.....	3
Table of Contents.....	5
1.0 Introduction.....	7
2.0 Background.....	9
2.1 What is FACET.....	9
2.2 User Interface Research.....	10
2.3 Preliminary FACET User Interface Prototype.....	11
3.0 Methodology.....	14
3.1 Objectives.....	14
3.2 Application Workflow.....	15
3.3 User Interface.....	18
3.4 Frontend.....	20
3.4.1 Welcome Screen.....	20
3.4.2 My Application.....	23
3.4.3 Feature Controls.....	23
3.4.4 Explanations.....	27
3.4.5 Suggestions.....	28
3.4.6 Scenarios.....	30
3.4.7 Application Styling.....	31
3.5 Backend.....	31
3.5.1 Initialization.....	31
3.5.2 Integration with Frontend.....	33
3.5.3 RESTful API Endpoints.....	34
3.5.4 Explanation Generation Process.....	34
4.0 Results.....	35
4.1 Testing Plan.....	35
4.2 Testing Procedure.....	36
4.3 Testing Results.....	37
5.0 Conclusion.....	39
5.1 Future Work.....	39
5.1.1 Accessibility Styling.....	39
5.1.2 User Testing.....	40
5.1.3 Feature Filter.....	40

5.1.4 Scenario Comparison.....	43
5.1.5 Explaining App Functionality to Users.....	44
5.1.6 Loading Categorical Data.....	45
5.2 Conclusion.....	46
Citations.....	47
Appendices.....	49
Appendix A.....	49
Appendix B.....	50
Appendix C.....	51
Appendix D.....	52
Appendix E.....	54
Appendix F.....	58

1.0 Introduction

The incorporation of machine learning into critical decision-making processes has seen a significant rise in recent years, impacting various domains and industries, such as healthcare and recruitment (Millard, 2023; White, 2024). From 2017 to 2022, the adoption of AI has more than doubled, indicating a growing reliance on automated decision-making systems. As companies continue to witness tangible returns on their investments, this trend is expected to continue its upward trajectory (McKinsey, 2022). Consequently, explainable artificial intelligence (XAI) methods have gained prominence, aiming to enhance transparency and fairness in decision-making processes by elucidating the rationale behind AI-generated decisions (Forbes, 2021).

Counterfactual explanations represent one such xAI technique, addressing queries in the form of hypothetical scenarios, exemplified by questions like “Why was my loan application declined instead of approved?” (Molnar, 2023). These explanations explore the impact of altering certain input features on the model's decision outcome. However, current approaches to generating counterfactual explanations often overlook user preferences and fail to produce actionable solutions. Notably, they neglect to consider which feature modifications users would prioritize and may propose changes that are impractical or implausible to realize under real-world conditions (Keane, 2021).

To address these shortcomings, the novel counterfactual region was devised. This framework extends beyond individual point-based explanations, encompassing a broader space of potential solutions. FACET (Fast Actionable Counterfactuals for Ensembles of Trees) offers

personalized and robust counterfactual explanations (VanNostrand, 2023). Through the use of counterfactual regions, FACET ensures explanation robustness and provides users with a multitude of actionable and relevant solutions tailored to their specific circumstances. FACET efficiently navigates complex models and user queries, delivering real-time insights into decision-making processes.

It is crucial to address user understandability and provide an intuitive method for interacting with FACET and querying it. To overcome this hurdle, the team developed an interactive user interface that empowers users to adjust constraints and prioritize features during their interactions with FACET. This user-friendly interface facilitates seamless engagement with FACET and allows users to save outputs for future reference. To ensure the interface worked as intended, a comprehensive testing plan was devised, covering general functionality, flexibility, and edge case testing across different scenarios, datasets, and parameters. The creation of an interactive user interface, along with thorough testing across diverse scenarios, underscores the importance of enhancing user experience and ensuring the effectiveness of FACET in diverse contexts.

2.0 Background

2.1 What is FACET

In today's world of artificial intelligence, it is becoming more and more pivotal to understand the reasoning behind the decisions made by machine learning models. This understanding helps to build trust, ensure accountability, and comply with regulations. To meet the increasing need for XAI, a new tool called FACET was developed by Peter VanNostrand. This tool generates counterfactual explanations for complex machine learning systems, which answer questions based on hypothetical scenarios. By providing multiple counterfactual explanations and organizing them by region, FACET illustrates how the output of a model may change under different conditions. The tool accomplishes this feat by precomputing and indexing counterfactual regions in the feature space, which enables users to interactively query and explore these regions.

FACET distinguishes itself from other counterfactual models by prioritizing user-friendliness and actionability. It allows users to incorporate personalized constraints and priorities, ensuring the explanations are directly applicable to their unique real-world scenario. Unlike other models that focus solely on generating counterfactual instances, FACET's approach to region-based explanations provides a more holistic and interpretable view of the model's behavior. The Counterfactual Region Explanation Index (COREX) is a critical component of FACET. It is a bit vector-based index that efficiently encodes high-dimensional spatial data used

to represent counterfactual regions. It plays a crucial role in quickly retrieving relevant regions from the input space, even for complex models and large explanation spaces.

FACET's effectiveness was determined through empirical evaluations and experiments conducted in the associated paper (VanNostrand, 2023), where it performed better than several other standard algorithms. These experiments typically involve using real-world or synthetic datasets and assessing FACET's performance in generating actionable counterfactual explanations.

2.2 User Interface Research

When designing FACET's user interface (UI), the team considered the many elements that contribute to an effective UI. Primarily, the UI should provide information about its functionality and features without overwhelming the user with clutter (Henry, n.d.). Achieving the right balance between these two factors is critical to creating a user-friendly and visually appealing UI.

In order to create a robust UI, it is vital to understand the user's objectives, skills, preferences, and tendencies. This understanding sets the foundation for designing an interface that meets users' needs and expectations. To achieve this, designers must follow the core principles of design, such as keeping the interface simple and eliminating unnecessary elements. They must also maintain consistency in colors, text fonts, and other design elements, strategically placing items to draw attention to information and providing clear explanations for permitted or prohibited actions (Chamorro, 2023).

By following these principles, designers can create an intuitive and engaging UI that enables users to achieve their goals seamlessly. Ultimately, a well-designed UI can enhance user experience, increase engagement and satisfaction, and foster a positive perception of the product or service. When designing a user interface, it is essential to avoid incorporating elements that could negatively impact the user experience. These could range from excessively bright or high-contrast color schemes, typography that impedes readability, unresponsive interactions, inconsistent design styles, and a lack of adaptability to different screen sizes. All of these factors can hinder users from reading content, navigating the UI, and comprehending presented data, ultimately leading to a suboptimal user experience.

Incorporating accessibility features within a user interface is an integral aspect of UI design. Accessibility is critical in creating user interfaces that accommodate a wide range of individuals, particularly those with disabilities. By offering an accessibility option, users can optimize user interfaces' effectiveness, efficiency, and satisfaction.

To create a consistent and user-friendly UI design, it is necessary to adhere to best practices and avoid these harmful elements. By doing so, the UI can ensure a user experience that meets expectations. This attention to detail creates a polished appearance that resonates with our audience and promotes user engagement.

2.3 Preliminary FACET User Interface Prototype

Peter VanNostrand designed a preliminary prototype for the user interface for FACET. This prototype, displayed in Figure 1, was created with the purpose of visualizing FACET and experimenting with ways in which a user could interact with it as a system. It serves as a

conceptual model for the user interface, allowing for testing and refinement of the system's design.

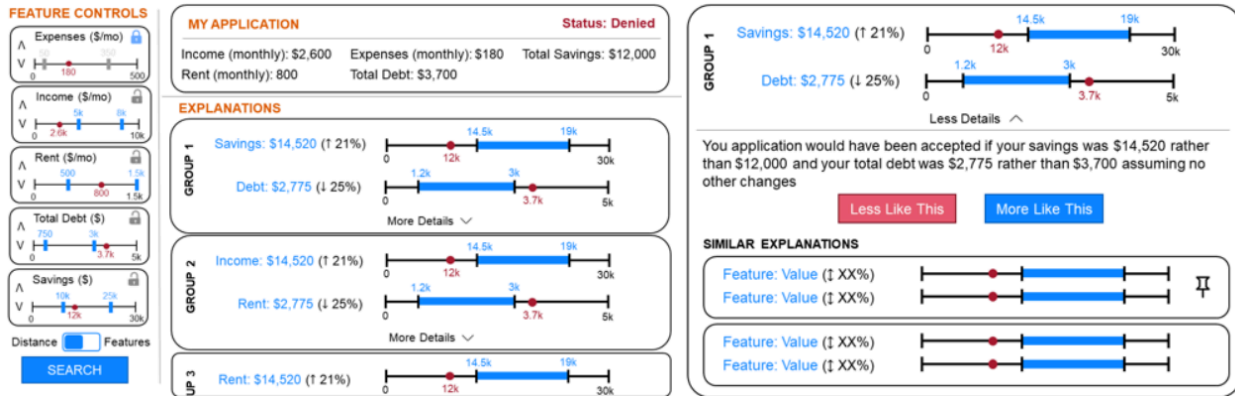


Figure 1. Peter VanNostrand's User Interface for FACET.

The prototype features four main components:

- 1) **Feature Controls:** On the left side of the screen, you will find a list of features and their range of values that FACET uses to generate counterfactual explanations. You can lock the range of values or lock the value as a constant using a switch. A locking icon is displayed in the top right corner of each feature to signify which features cannot be changed.
- 2) **My Application:** In the top-middle section of the screen, you will see My Application, which provides insight into the current application that FACET uses for generating counterfactual explanations. All feature values are displayed here, along with the overall status of the application, whether it is rejected or accepted.
- 3) **Explanations:** Under My Application, you will find Explanations, which lists feasible counterfactual explanations that align with the critique of feature control and change the

application status to accepted. The features related to the application's status change are depicted as number lines on the right side of each explanation, and the applicant's current value is marked with a red dot.

- 4) **Detailed Explanation:** On the right side of the screen, you will find Detailed Explanation, which provides a more detailed explanation compared to the Explanations section. This section includes number lines and a detailed explanation of which values need to change for the application to be accepted. Additionally, this explanation offers two buttons to find more explanations, which can be either similar or different from the given one, and these buttons are located at the bottom of the section.

This prototype presents a compelling visual representation of FACET, but there are some limitations and design elements that could be improved for better usability. One of the key considerations is to make the display more user-friendly for individuals with little prior knowledge of FACET. Currently, VanNostrand's visualization heavily relies on having an existing understanding of the FACET system, which may not be true of all users. In addition, there are numerous interactive elements that could be more intuitive, making it challenging for the average user to fully grasp the relationships between the feature control, detailed explanation, and explanations sections. The information conveyed by these elements may also be considered inscrutable, which poses a challenge for users trying to make sense of the data.

3.0 Methodology

3.1 Objectives

In order to meet the design goals set early on in the project's development, the team set out to accomplish the following objectives:

- 1) **Preliminary research:** The team held weekly code interviews with the developer of FACET to ensure a shared understanding of the project and the tasks to be performed. Further research was conducted into various web development tools and HCI practices to aid in development of a high-quality user interface.
- 2) **Designed preliminary UI prototypes:** The team created and iterated on many UI prototypes, each improving on the previous with feedback from the team and the project advisors. Key UI prototype iterations can be found in Appendices A-D.
- 3) **Initializing the backend:** The team worked quickly and efficiently to link the FACET model to a Python backend that could successfully communicate with a Javascript/React frontend, allowing for initial testing and feature implementation.
- 4) **Integration of a fully functional UI:** The team worked together to create and integrate each individual component into a central user interface, with frequent refinements made to both the system and visuals.

3.2 Application Workflow

Before implementing the details of the application, the team modeled application workflows anticipated users may experience and outlined which intermediate processes would need to be included when creating the application. The rough flow of communication between the user and FACET was initially captured with UML diagrams, which can be seen in figures 2 and 3 below. Although the naming of various functions has been simplified in these figures for readability, the general workflow is still accurate. The two primary workflows that occur are when (1) the application is initialized and (2) the user modifies constraints to generate new explanations that reflect the changes.

The first workflow occurs when the application is initialized and before the user can view the interface. When the application is opened, the backend is initialized, which loads in and instantiates the dataset that will be used to train the FACET core. The dataset is split into a training and testing set. After FACET is done training on the first data subset it is ready to provide a counterfactual explanation region. The application's user interface opens and welcomes the user by displaying a dropdown populated with instances from the testing dataset. After a specific instance is selected the user continues past the selection screen to the home screen. The instance, along with other initial parameters, are used to query FACET, in order to retrieve explanations. These explanations are showcased within the 'Explanations' section, where for every feature in the dataset the set of counterfactual regions are displayed, accompanied by relevant suggestions.

The second workflow is captured by the ‘Feature Controls’ section, which contain panels that allow the user to constrain the ranges of the counterfactual regions received from FACET. Similar to the initialization workflow, the constraints are passed along with the instance to FACET to generate a new explanation set, replacing the old explanations.

In addition to these workflows, there are other features that are offered in the application. If a user views a certain explanation that they like, they can save the explanation as a scenario, which may be opened in the future. Additionally, as the counterfactual explanations may be difficult to understand — especially for non-technical users — there is also a section that provides a textual interpretation of the counterfactual explanations, giving the user a suggestion of actionable steps to take in order for their instance to be “accepted.” A more detailed explanation of how the backend, scenarios, and constraints, and explanations are implemented will be described in their respective sections later.

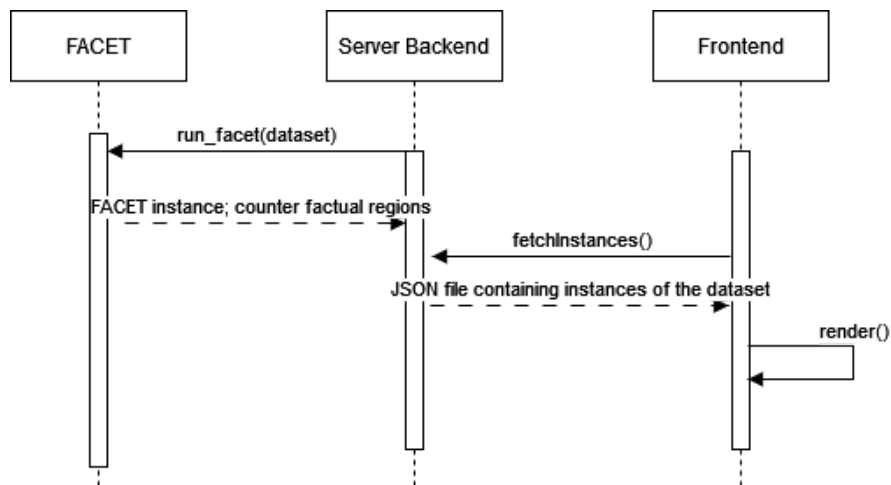


Figure 2. Initializing the application.

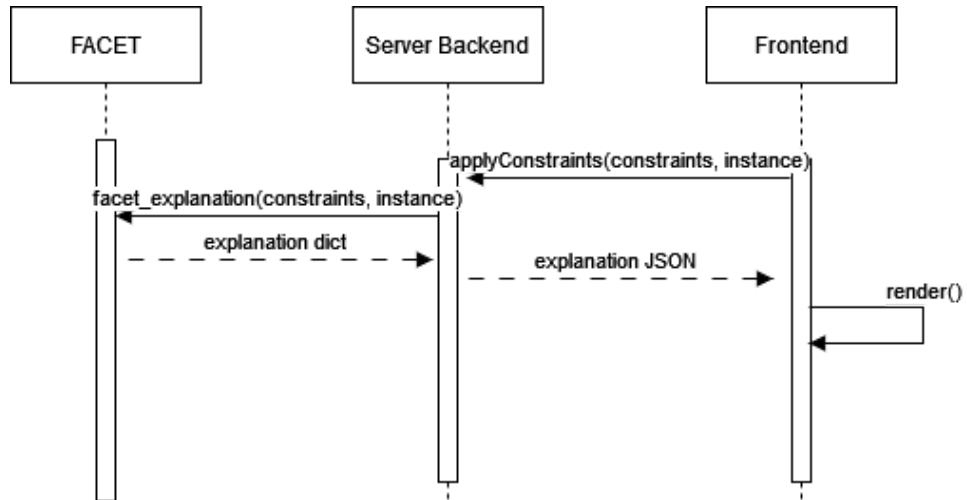


Figure 3. Generating explanations with constraints.

3.3 User Interface

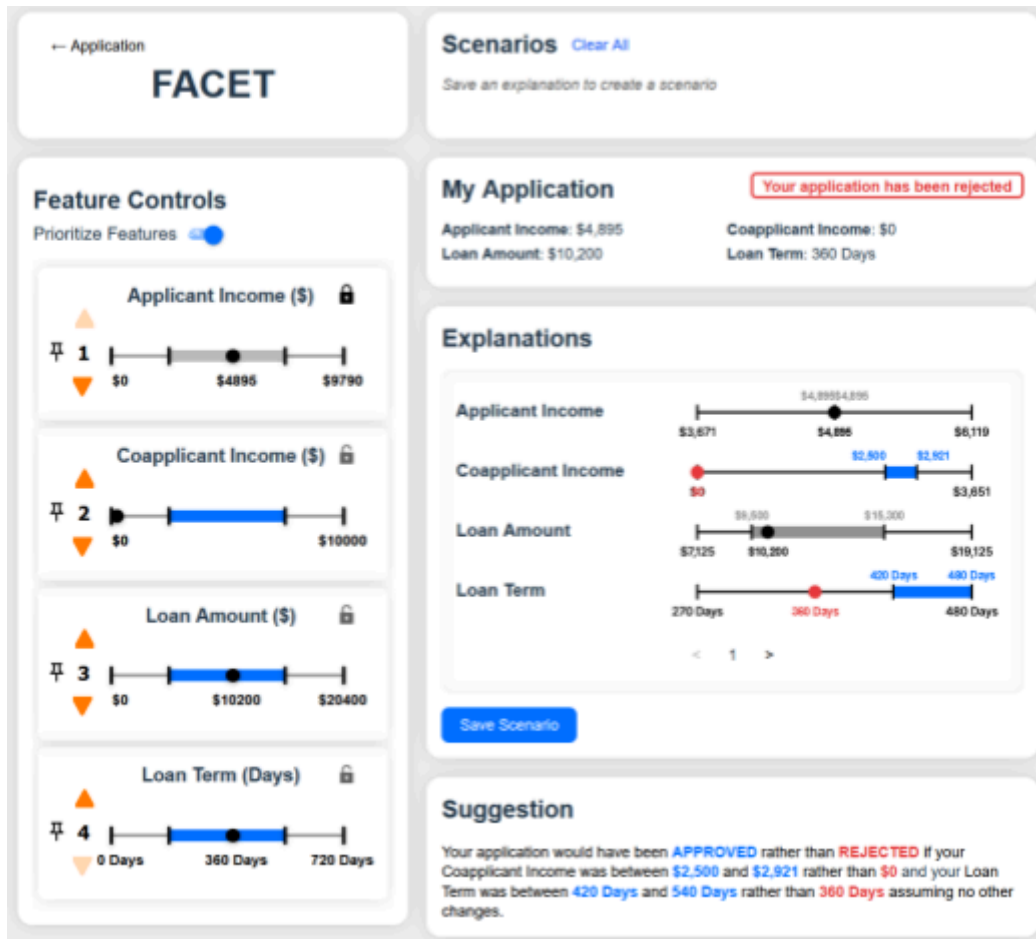


Figure 4. Layout of the final application

The current UI of the application can be found in Figure 4. Its layout can be broken down into five key components:

- 1) **Feature Controls:** These controls consist of a list of features and their corresponding range of values. Each component in the list contains a locking icon at the top right corner, which indicates that the feature value cannot be changed. The feature control elements also include arrows that allow users to change the priority of each value as desired and

pick which values they are more willing to change than others. Users are additionally able to pin a feature at its current priority by clicking the pin icon which prevents a feature's priority from being reordered by the other features.

- 2) **Scenarios:** Users can save multiple instances for different application explanations, or scenarios. Each scenario represents a specific counterfactual region. Users can create a new application, in addition to the default one, that enables them to apply any desired changes using the 'Feature Controls' feature or by accepting suggestions generated by the 'Suggestion' component. This allows users to personalize and customize their experience based on their preferences and needs.
- 3) **My Application:** My Application provides valuable information about FACET's current application for generating counterfactual explanations. It displays all the feature values and shows the overall status of the application. The only changes made to this component were adding a user-friendly option to switch between different applicants and a detailed tutorial on correctly using the component. This component is a vital application feature that helps users understand its performance and stay informed. "My Application" also includes an expand option that allows users to view all values within the application without having to scroll.
- 4) **Explanations:** Provides a more comprehensive list of possible counterfactual explanations under the 'Explanations' tab, making the information more user-friendly and easier to understand. This feature allows users to better understand why the system approved or denied their application and use the information to improve future applications

- 5) **Suggestions:** To better understand the visuals from the explanations, suggestions are created to provide helpful recommendations for users to improve the current application. These suggestions are aimed to synthesize the key points of the explanations and communicate them to the user in text form that is much easier to interpret.

The user interface (UI) underwent a series of design updates throughout the project, with each iteration building upon the previous one. Each section of the interface was meticulously refined to present the necessary information only without causing any disruptions to the user experience. An example of this can be seen by contrasting the first prototype (Appendix B) with the second iteration (Appendix C). By adding whitespace around the components, the team ensured that each component within the UI was not cluttered together, thus making the UI more visually appealing and user friendly. These design changes were implemented to create a more streamlined and intuitive user interface, ultimately resulting in a more seamless and user-friendly experience for all users.

3.4 Frontend

3.4.1 Welcome Screen

The Welcome Screen serves as the user's first interaction with FACET's user interface and allows the user to select the starting applicant values for FACET to generate explanations. It is pivotal that this screen is easily interpretable and eases the user into understanding the common design language within FACET's UI. Functionally, the Welcome Screen allows the user

to either pick from a list of preloaded applicants (Figure 5) or manually input values for each of the variables (Figure 6).

The screenshot shows a form titled "Applicant Selection" with a close button (X) in the top right corner. Under the heading "Applicant Type", there are two toggle buttons: "DROPDOWN" (which is active and highlighted) and "CUSTOM". Below these is a dropdown menu showing "Applicant 0" with a downward arrow. To the right, there are four input fields: "Applicant Income" (4895), "Coapplicant Income" (0), "Loan Amount" (10200), and "Loan Term" (360 Days). Each income and loan amount field has a dollar sign (\$) on the right. A blue "Continue" button is located at the bottom right.

Figure 5. Welcome Screen: Applicant Dropdown

The screenshot shows the same "Applicant Selection" form. In this view, the "CUSTOM" toggle button is active and highlighted, while "DROPDOWN" is inactive. The dropdown menu now shows "Custom Applicant" with a downward arrow. The input fields on the right are all set to "0": "Applicant Income", "Coapplicant Income", "Loan Amount", and "Loan Term" (0 Days). The "Continue" button remains at the bottom right.

Figure 6. Welcome Screen Custom Applicant

The Welcome Screen has two unique tabs for the user: the pre-loaded applicant tab and the custom applicant tab. The pre-loaded applicant tab allows the user to pick from a dropdown with applicants loaded into the program before continuing to the FACET system. The custom applicant tab allows the user to enter custom values for each of the scenario's features. These text input boxes are validated to make sure that the user is entering valid numeric values prior to being sent to the application.

To streamline the codebase and reduce reliance on custom subcomponents, the team opted to integrate components from the Material UI (MUI) library. The `ToggleButtonGroup` ('Toggle Button Group Material UI', 2024), `Autocomplete` ('Autocomplete Material UI', 2024), and `TextField` ('Text Field Material UI', 2024) components were utilized to facilitate the switching between custom and dropdown tabs, dropdown menus, and text boxes. To differentiate tabs, the text fields are disabled when a preset applicant is selected and the dropdown is disabled when the custom applicant is being created. This adoption of new components led to a more refined overall design, enhancing user experience and functionality.

After selecting or providing details for an application, the user can then confirm by pressing the button, triggering FACET to populate with the new applicant values and generate explanations based on them. The user can return to the Welcome Screen at any point and select another application through clicking the Welcome Screen button on the main page (Figure 7).

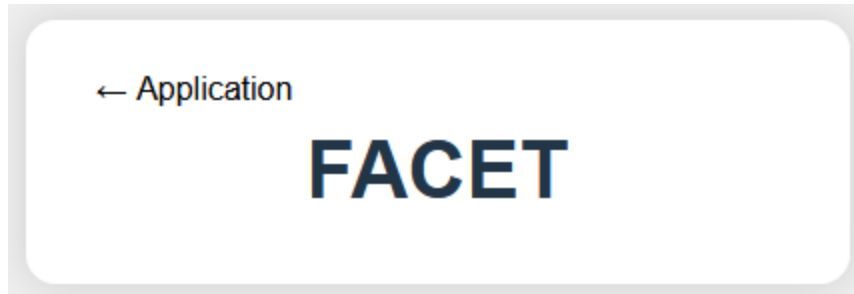


Figure 7. Return to Welcome Screen button

3.4.2 My Application

The 'My Application' section is a vital aspect of our application platform, providing users with relevant and up-to-date information on the values present within their applications (Figure 8). This tab presents all values in an organized manner without the need to read through the Feature Controls and Explanation sections on the user interface.

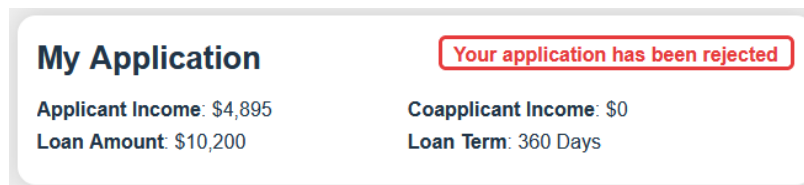


Figure 8. The 'My Application' section

3.4.3 Feature Controls

The Feature Controls section serves as the primary tool for end-users to interact with the FACET system. By setting and adjusting the constraint values within the Feature Controls, users define the accepted range. This range, determined by the user, signifies the spectrum of values they are willing to adapt to. For instance, in our loan application dataset, if a user's current rent is

\$2,800 per month and they set their accepted range between \$2,000 and \$3,400, it indicates their flexibility in adjusting their rent within that range to secure loan approval.

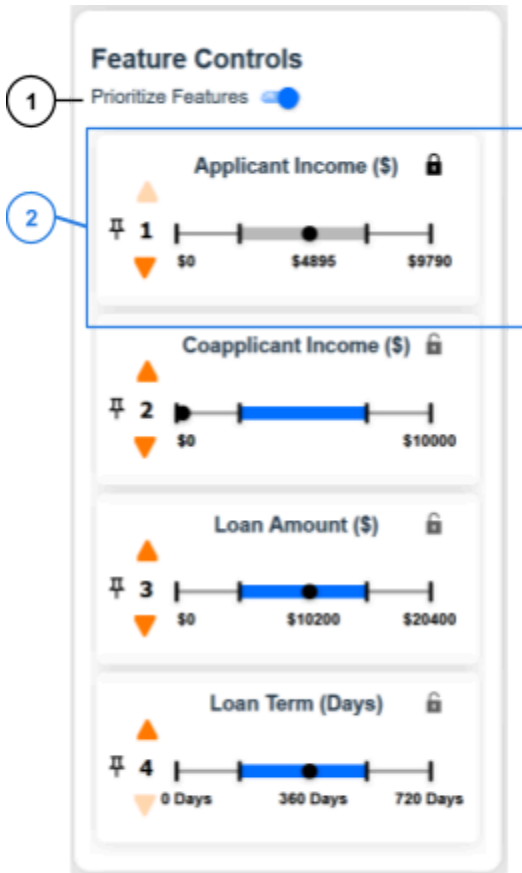


Figure 9.1. Feature Controls

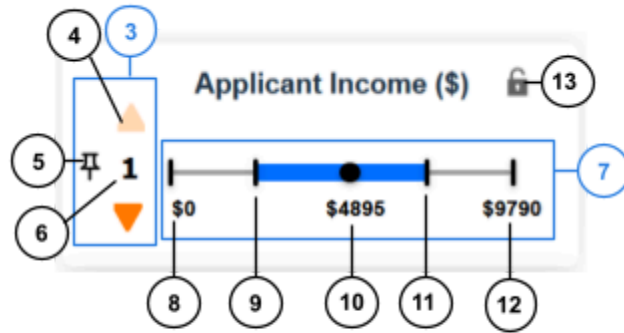


Figure 9.2. A Feature Control

Figure 9. Feature Controls break down

As FACET can be trained on a dataset containing any number of features that affect the outcome of the model, a corresponding feature control (Figure 9.2) exists to allow users to explore how different values can affect the model's outcome. A feature control comprises the following elements: the range slider [7], which includes the minimum and maximum values [8, 12], the set range [9], [11], and the current value of the feature [10], the priority value [6] along with pins [5], and the lock [13].

- 1) **Range Slider:** The range slider [2] allows users to define a range of values to be transmitted to FACET, indicating the spectrum of values considered in explanations.
- 2) **Priority Value:** The priority value corresponds to a feature's weight, representing the user's preference or ability to modify a selected feature, with '1' indicating the highest difficulty in making changes. Users can adjust the priority value by inputting a value in the input box [6] or by using the arrows [4] to increment or decrement the priority.
- 3) **Priority Pinning:** By pinning a priority [5], users can lock a feature to the specified priority, preventing it from being altered by adjustments to other features' priorities.
- 4) **Prioritize Features:** Should users prefer to assign equal weight to each feature, they can toggle off the priorities using the switch located at the top of the controls [1].
- 5) **Feature Locking:** The lock [13] allows users to fix a feature at its current value, indicating their preference for keeping that feature's value unchanged.

The Feature Controls section interfaces primarily with three other components. Firstly, the applicant information, as selected or entered in the Welcome Screen, is displayed as the current value in the slider. Secondly, when users make changes to the constraints, the adjusted constraints are related to FACET, which subsequently updates the Explanation section in real-time. Lastly, when users load a saved scenario from the Scenario tab, the constraints update to reflect the constraint values saved within that scenario, and any edits made to an open scenario are saved accordingly.

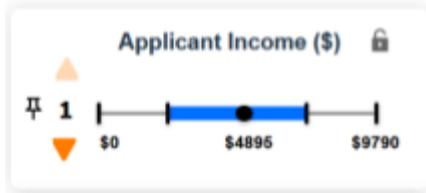


Figure 10.1. Default Feature

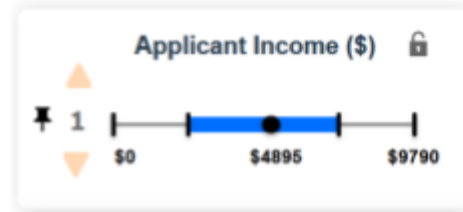


Figure 10.2. Pinned Feature

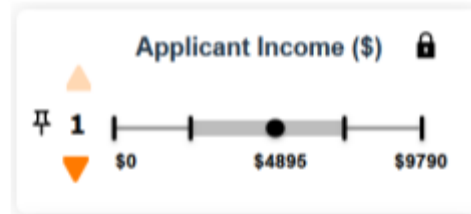


Figure 10.3. Locked Feature

Prioritize Features

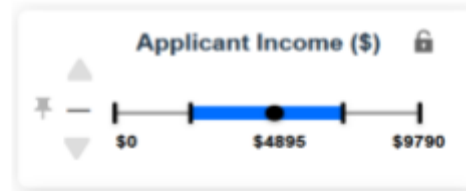


Figure 10.4. Feature Priorities Toggled Off

Figure 10. Feature with different constraints changed

For the second iteration of the Feature Controls section, the appearance was refined and code was streamlined through the incorporation Slider ("Slide Material UI," 2024), Icon Buttons ("IconButton API," 2024), and toggle Switch ("Switch Material UI," 2024) components from the MUI library. To seamlessly integrate with the existing interface, the styles of the components were overridden. The layout of the Feature Controls was further revised and the pinning button was relocated to the left of the priority value to improve clarity regarding its function. Visual cues were implemented to denote the changes made when pinning a feature (Figure 10.2), locking a feature (Figure 10.3), and toggling the weights off or deprioritizing a feature (Figure 10.4). By providing clearer functionality cues and improving usability, these enhancements ensure that users can effectively manage constraints, thereby enhancing their experience with the system.

3.4.4 Explanations

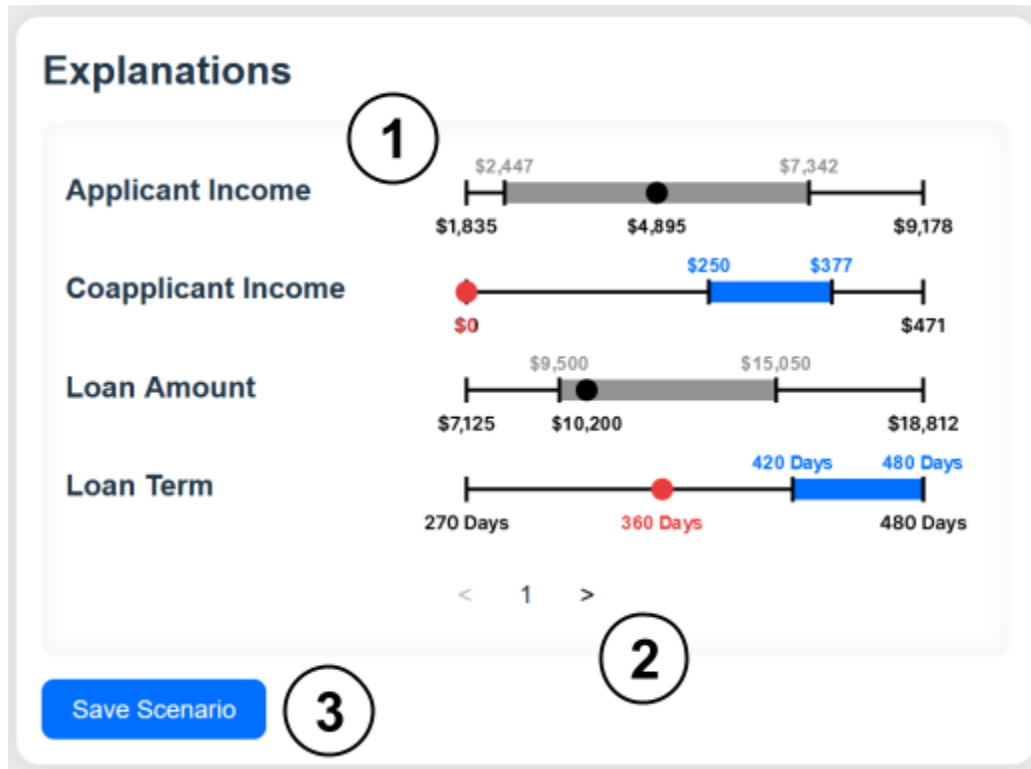


Figure 11. Explanation Section

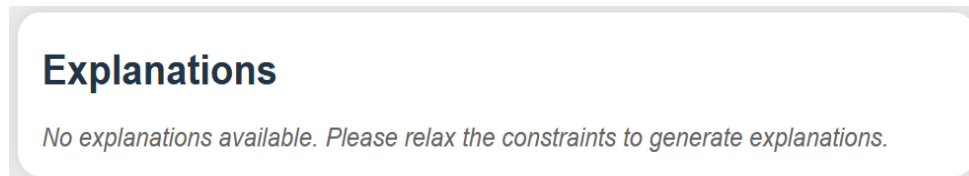


Figure 12. Explanation Section: no explanations generated

The Explanations section (Figure 11) provides users with a visual display of the generated counterfactual explanations for the user selected instance. Explanations offers functionalities for navigating through multiple explanations and saving scenarios based on the provided explanations. An overview of its key functionalities is as follows:

- 1) **Explanation Display:** Explanations [1] are presented in a user-friendly format, with each feature's explanation visualized using a custom number line, which was partially built

using existing code from VanNostrand's previous user study using the D3.js library. This allows users to interpret and analyze the impact of individual features on the model's predictions.

- 2) **Navigation Controls:** Users can navigate through multiple explanations using next and previous buttons [2]. The component dynamically updates the displayed explanation based on the current explanation index.
- 3) **Scenario Saving:** Users have the option to save scenarios based on the current explanation. By clicking the "Save Scenario" button [3], users capture the current instance, feature constraints, and generated explanations to cache for future reference and analysis.
- 4) **Error Handling:** If no explanations are available or if constraints prevent explanation generation, the component displays a message to guide users to relax the constraints to provide FACET a larger space to generate explanations (Figure 12).

3.4.5 Suggestions

The suggestion section is designed to distill the main points of the current explanation and communicate them to the user in text form. This is achieved by crafting a sentence outlining the actionable steps the user needs to take for their instance to be accepted. An example of the suggestion section is shown below in Figure 13.

Suggestion

Your application would have been **APPROVED** rather than **REJECTED** if your Applicant Income was between **\$1,013** and **\$1,519** rather than **\$4,895** and your Loan Amount was between **\$9,450** and **\$10,000** rather than **\$10,200** assuming no other changes.

Figure 13. Suggestion Box

The suggestion is dynamically generated, building each sentence component to address necessary changes to each feature value. Each suggestion adheres to a consistent template, including static introductory and concluding text. For feature values, the suggestion employs the following format: "your FEATURE_NAME was between LOWER_BOUND and UPPER_BOUND rather than CURRENT_VALUE." The presented suggestion reflects the currently loaded explanation and updates accordingly when a new explanation is selected, as demonstrated in Figures 14 and 15.

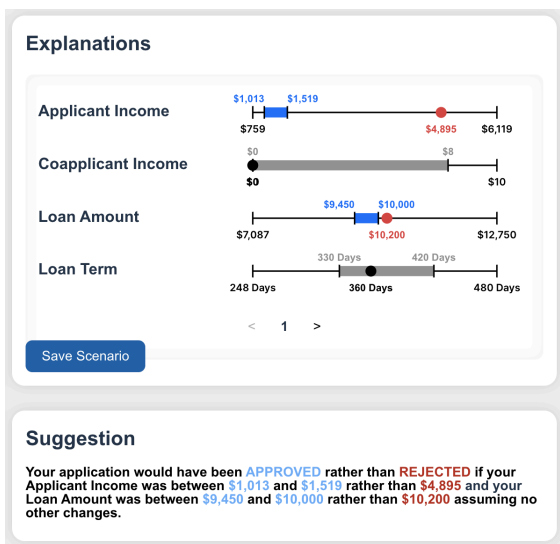


Figure 14. Suggestion generated for the 1st explanation

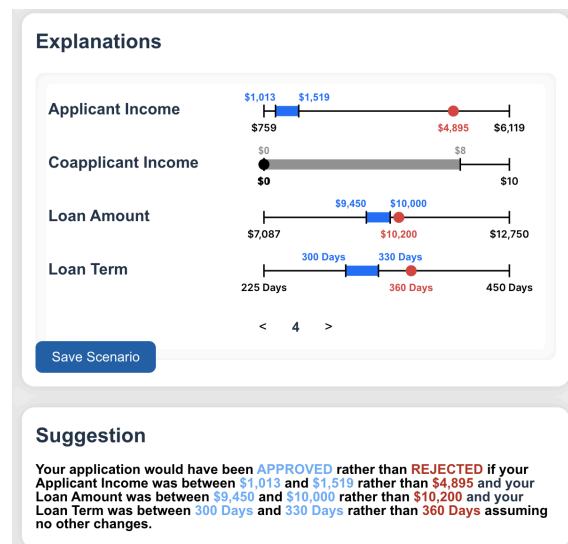


Figure 15. Suggestion generated for the 4th explanation

3.4.6 Scenarios

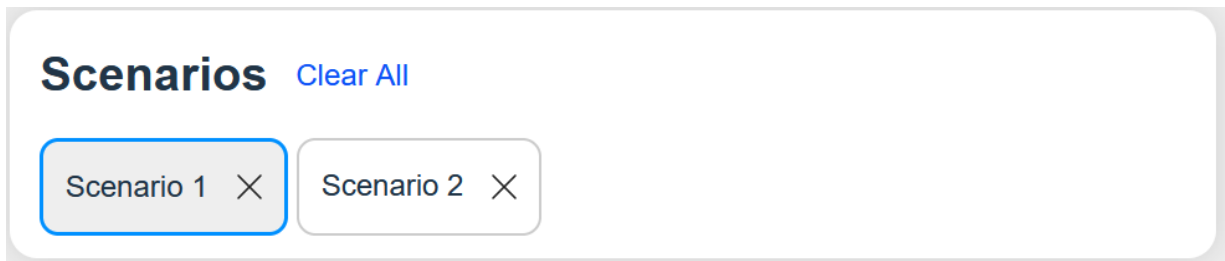


Figure 16. Scenario Section with Scenario 1 selected

The Scenarios tab allows users to manage and interact with saved scenarios based on generated explanations (Figure 16). Users can effectively manage and explore different scenarios based on generated explanations, facilitating a deeper understanding of model behavior and predictions. When a scenario is saved, it is assigned a unique ID for convenient reference, and the scenario's explanation, constraints, and values are serialized to a JSON object and stored for later viewing. In regards to managing scenarios, the following features are provided:

- 1) **Scenario Selection:** Users can select a scenario by clicking on its corresponding tab in the scenario list. Upon selection, the component updates the application state to reflect the selected scenario's instance, explanations, and constraints.
- 2) **Scenario Deletion:** Users have the option to delete individual scenarios by clicking the close button (x) on each scenario tab. This action removes the selected scenario from the list of saved scenarios and updates the application state accordingly.
- 3) **Scenario Clearing:** The "Clear All" button allows users to clear all saved scenarios at once. Clicking this button resets the saved scenarios list and clears any selected scenario, providing users with a clean slate for creating new scenarios.

- 4) **Instructions Text:** When no scenarios are saved, the component displays an instructional text prompting users to save an explanation to create a scenario. This helps guide users on how to interact with the scenarios feature.

3.4.7 Application Styling

To style the application a general grid layout was implemented, where each component's code could be injected into its respective section. A CSS file was then made for each component and filled out to replicate the design created in Figma. As development progressed, in-line styling was abstracted out into a CSS file for easier manipulation and better sustainability of the project. The final styling reflected a relatively more modern look compared to the initial Figma prototype.

3.5 Backend

The backend of the FACET system primarily serves as a middle piece connecting the frontend to the FACET core system for explanation generation while also providing access to the dataset and its metadata for analysis. The backend server, implemented using Python and Flask—a lightweight and flexible web framework—provides a robust infrastructure for handling API requests and conveniently integrates with the FACET core system, which is also written in Python.

3.5.1 Initialization

When the application starts, the backend is first initialized in order to set up the environment and prepare the system for operation. This involves several key steps:

- 1) **Loading Configuration Parameters:** The backend loads configuration parameters from a JSON file (config.json). These parameters include the port for the RESTful explanation API and the name of the dataset. This ensures that the backend is configured according to the specified settings.
- 2) **Setting up File Paths:** Paths to the dataset details and human-readable information files are established. These files contain essential information about the dataset, such as column names and formatting details, which are necessary for preprocessing data and converting raw explanation data—which is not interpretable by humans—into information lay users can understand without advanced technical experience.
- 3) **Configuring Flask Application:** The Flask application is configured to handle incoming requests and provide responses accordingly. This involves setting up routes, middleware, and other necessary components to ensure smooth operation of the backend.
- 4) **Enabling CORS:** Cross-Origin Resource Sharing (CORS) is enabled to allow the frontend to make requests to the backend from a different origin. This ensures that the frontend can communicate with the backend without encountering security restrictions.
- 5) **Initializing FACET Core:** The FACET core system is initialized, which involves loading data, training models, and indexing explanations. This step is essential for generating explanations based on user queries and dataset characteristics.
- 6) **Loading Sample Data:** Sample data is loaded into memory, providing a set of instances for testing and demonstration purposes. This allows users to interact with the system and explore its capabilities before using their own datasets.

3.5.2 Integration with Frontend

The backend integration of the FACET system involves seamless communication with both the frontend user interface and the FACET core system. It serves as the central component that facilitates data flow between the frontend and the core system, ensuring efficient generation of explanations and access to dataset files.

The backend integrates closely with the frontend user interface to provide a cohesive user experience. This integration involves:

- 1) **Handling API Requests:** The backend serves as the middleware that handles API requests from the frontend. It receives requests for generating explanations, accessing dataset files, and other functionalities, and processes them accordingly.
- 2) **Processing User Inputs:** User inputs from the frontend such as the selected instances, feature values, and constraints, are processed by the backend to prepare them for input into the FACET core system. This involves scaling data, formatting requests, and ensuring compatibility with the core system's requirements.
- 3) **Generating Explanations:** Upon receiving a request for explanation generation the backend communicates with the FACET core system to generate explanations based on the provided inputs. It passes the processed data to the core system, receives the generated explanations, and returns them to the frontend for display.
- 4) **Providing Feedback:** The backend provides feedback to the frontend based on the results of API requests. This includes success messages, error handling, and status

updates to inform users of the system's status and any issues encountered during operation.

3.5.3 RESTful API Endpoints

The backend provides several RESTful API endpoints to serve various functionalities required by the FACET system:

- 1) **Fetching Test Instances (GET /instances):** This endpoint returns sample instances for testing purposes.
- 2) **Fetching Human Format Information (GET /human_format):** Returns human-readable formatting information for display purposes.
- 3) **Serving Data Files (GET /data/{dataset_name}):** Provides access to data files required by the application, such as datasets and formatting files.
- 4) **Generating Explanations (POST /facet/explanations):** The main API endpoint for explaining instances. It receives input data in JSON format, processes it, and returns explanations for the provided instance.

3.5.4 Explanation Generation Process

The backend's core functionality lies in the explanation generation process. When a request is received at the /facet/explanations endpoint, the backend preprocesses the input data, scales it if necessary, and passes it to the FACET core system for explanation generation. Upon receiving the explanations, the backend processes them, formats them into a JSON-compatible structure, and returns them to the client for display.

4.0 Results

Due to the nature and size of the project, extensive testing was essential to ensure users a high quality application using FACET's algorithms. Consequently, a comprehensive testing plan was devised to test all features of the application.

4.1 Testing Plan

The testing plan was broken into three main testing phases:

- 1) **General Functionality:** Each component of the application was tested with realistic situations to ensure that the app functioned as intended. This included, but was not limited to, ensuring that constraints properly update explanations and suggestions, saved scenarios retain all important information and load correctly, and explanations are both accurate and reasonable.
- 2) **Flexibility:** One of FACET's key features is the ability to work with any dataset, given the necessary files are provided. To ensure this functionality, another dataset was loaded into FACET with different features, units, and metrics to determine if the application met this standard.
- 3) **Edge Case Testing:** Each component was tested with extreme situations to ensure that robustness under rare circumstances, such as when no explanation can be found or when relatively large numbers were inputted.

The planning document that contains more details can be found in Appendix E.

4.2 Testing Procedure

Testing initially began in the general functionality phase, where each component of the app was rigorously tested to weed out any bugs and unintended features. In the Feature Controls section, it was crucial to ensure proper implementation of functions such as pinning, locking, priority switching, toggling, and range adjustment. Various combinations of different ranges, values, and permutations of pinning, locking, and priority switching were tested, both individually and in combination with each other. Validation of these functions was conducted using generated explanations and backend code to confirm their intended functionality.

For the saved scenarios component, different constraints, explanations, and applications were saved as individual scenarios and compared to one another to ensure each one reflected their original states. Throughout all of testing, the explanations and suggestions components were checked to ensure they properly depicted the changes made by the feature controls and the saved scenarios.

The next phase involved testing the flexibility of the app. A `human_readable.json` file was created for another dataset with different units, semantic values, and other parameters. This dataset was then loaded into FACET and the app was checked over to ensure that valid ranges were presented and general functionality persisted. Additionally, the original dataset was tested with different parameters in its respective `human_readable.json` file to ensure hard coded values no longer existed within the application.

The final phase of the testing plan involved edge case testing. Edge cases are rare circumstances that do not normally show up during regular usage of the application but still have

a small chance of occurring. These cases are often extreme and involve unexpected behaviors. Edge cases that were investigated included, but were not limited to, scenarios where negative values or values greater than 1E6 were input, cases with no valid explanations, and instances with more than 100 saved scenarios, and situations with unreasonable constraints.

This procedure was repeated twice. In the first wave of testing any unintended behaviors were documented and classified as either high or low priority. High priority issues were patched whereas low priority issues were left for future development. After the initial bugs were found a second wave of testing was conducted to ensure the bugs were fixed.

4.3 Testing Results

The results of the testing were promising; All formatting and styling bugs tracked throughout the duration of the project were addressed and resolved, resulting in a UX friendly appearance. No single section took up overwhelming space on the screen and the UI maintains an aesthetically pleasing symmetrical layout that flows from section to section in a logical way.

The testing process did, however, reveal critical issues within the application, including performance bottlenecks, unintended feature exploration limitations, and occasionally invalid explanations.

Generating explanations dynamically takes an unacceptably long time depending on the constraints, causing noticeable lag in displaying new explanations and various issues with the rest of the application. Furthermore, the application cannot handle just one explorable feature, incorrectly loading NaNs into the feature ranges and failing to load a single instance of the

dataset. Conversely, loading more than four explorable features results in the explanations having undefined values and ranges. Lastly, under certain constraints, the application provides unreasonable explanations, such as only accepting a loan if you drop your income to the thousands from the tens of thousands.

The results of the tests and the steps to recreate them can be found in greater detail in Appendix F.

5.0 Conclusion

5.1 Future Work

5.1.1 Accessibility Styling

By providing accessibility options, the application can better accommodate users with various disabilities and enhance user experience. Through research, the team recognized the significance of inclusivity and accessibility in UI design and is committed to ensuring the platform is accessible and user-friendly for everyone. However, due to time constraints, the team was unable to meet all of the set accessibility goals.

The team had planned to address accessibility issues faced by users with motor skills difficulties, visual impairments, age-related limitations, anxiety concerns, language barriers, and other mental/physical impairments. To achieve these goals, elements and interactive components of the UI can be reevaluated to be sensitive to people with motor control difficulties and resizable for visual impairments. Additionally, the color palette for the application can be redone to accommodate other visual impairments and improve contrast. Although not implemented, alternative style sheets were designed to meet the Web Content Accessibility Guidelines (WCAG), a set of principles intended to accommodate various disabilities that impair access to websites (W3C, 2020). These style sheets provided alternative color palettes for different forms of color blindness, meeting the WCAG AAA color contrast standard.

Furthermore, the language choice used by the application can be improved upon. While the team wanted to ensure that all communication within the UI was polite and informative without appearing patronizing, changes can be made to make the application more friendly to those who may face language barriers and to improve understandability of advanced technical terminology.

To further accommodate visual impairments, the application can benefit from features such as text-to-speech, image descriptions, and the ability to highlight content.

5.1.2 User Testing

Although the application underwent testing for general functionality, handling of edge cases, and adaptability to various datasets, it notably lacked user testing. User testing is highly significant due to its ability to provide insights into user experience, preferences, and potential usability issues that might not be apparent through other testing methods. For future iterations of the app it is imperative to conduct user testing to refine the layout, uncover latent bugs, and enhance the overall user experience. Through involving users in the testing process, developers can gain valuable feedback and identify areas for improvement for the FACET application.

5.1.3 Feature Filter

The feature filter component was initially planned as a significant enhancement to the existing prototype, offering users more detailed and refined controls for managing and navigating feature controls within FACET. While the preliminary implementation includes a toggle for each

filter to show or hide it in the Feature Control and the option to remove all applied filters entirely there are several areas for further development to fully realize the potential of this feature.

Current Implementation

The current implementation of the Feature Filter component, which is shown in Figure 17, includes the following functionalities:

- a. **Individual Filter Toggles:** Users can toggle individual filters to show or hide specific features in the feature control. This functionality provides users with granular control over the visibility of features based on their preferences and requirements.
- b. **Toggle All Filters:** Users have the option to toggle all filters on or off simultaneously, enabling them to quickly adjust the visibility of multiple features at once. This feature enhances user efficiency by streamlining the process of managing feature visibility.

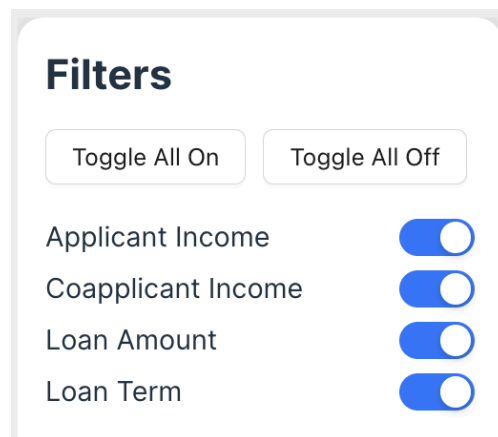


Figure 17. Preliminary implementation of feature filters

Next Steps

Building upon the preliminary implementation, the following enhancements and additions can be incorporated into the Feature Filter component to further improve usability and functionality:

- 1) **Search Bar:** Integrate a search bar within the Feature Filter component to allow users to quickly search for specific filter options. This feature enhances user accessibility and efficiency by enabling rapid navigation and retrieval of desired filter options.
- 2) **Submenus and Categorization:** Implement submenus within the Feature Filter component to categorize and organize filter options into logical groupings. This organizational structure enhances user navigation and comprehension, facilitating easier identification and selection of relevant filters.
- 3) **Applied Filters Menu:** Introduce a menu or display area that shows the filters currently applied to the feature controls. This feature provides users with visibility into the active filters, allowing them to track and manage applied filters effectively.
- 4) **Filter Removal Options:** Enable users to remove individual filters that are no longer needed or relevant to their analysis. Additionally, incorporate functionality to clear all applied filters with a single action, providing users with flexibility and control over filter management.

5.1.4 Scenario Comparison



Figure 18. Scenario comparison layout and sample walkthrough

Scenario Comparison is a component added late in the development of the application. It enables users to compare the explanations of two saved scenarios. The user selects two scenarios for comparison from the dropdowns (Figure 18.1) and the explanation index (Figure 18.2) to compare explanations. Users can choose to compare the explanations of the same scenario by selecting the same scenario for each dropdown.

Once two scenarios and explanations are selected, Scenario Comparison groups each explanation by feature and displays them one beneath the other (Figure 18.3-5). For each feature within an explanation, the number line is displayed with the accepted range (Figure 18.5). The ID of the associated scenario is displayed to the left of the number line (Figure 18.3). To the right of each number line, the percent change between the application's current value and the closest value in the explanation range (min or max range) is displayed (Figure 18.4). The lowest percent change between the two features is highlighted in blue to emphasize that it requires the least amount of change to be met. When the current applicant value falls within the explanation range,

the number line is grayed out (Figure 18.6), similarly to how it is implemented in the Explanation Section.

Despite implementing the component, it was omitted from the final version of the app due to the need to revise the app interface to accommodate it and for user testing. Additional user testing of the Scenario Comparison feature may offer insights into how end-users would utilize scenario comparison and how information could be presented more effectively to support them. Incorporating these insights and adjustments may lead to the integration of a revised version of the Scenario Comparison feature into the app in the future.

5.1.5 Explaining App Functionality to Users

In the course of the project, various methods for explaining the app's functionality to users were considered. Initially, options such as embedding video explanations or incorporating information buttons were explored during the development of the V1 prototype of the user interface. Eventually, the decision was made to include information buttons that users could click on for each significant component (Appendix B). However, due to time constraints, these information buttons were never implemented. Furthermore, during the redesign of the app layout, the information buttons were not refined and thus were excluded from the final version.

Moving forward, future iterations could explore the implementation of tooltip animations. These animations would appear when users hover over specific components and would provide brief explanations of their purpose. These tooltips could be programmed to activate at the start of each new session of the web app and could also be accessible by hovering over relevant areas of the screen. The exact development approach for this aspect of the app remains undetermined, but

it presents an opportunity for enhancing user understanding and interaction in subsequent versions.

5.1.6 Loading Categorical Data

The current iteration of the web app exclusively handles numerical data, yet alternative versions of the FACET system exist that are designed to accommodate categorical data. To enhance the app's capabilities, future iterations could be adapted to incorporate support for categorical data. This could involve implementing a slider with a restricted number of choices as illustrated in Figure 19, allowing users to only select predefined categorical values.



Figure 19. Example of slider with predefined values of size

Incorporating support for categorical data into future iterations of the app is essential for various reasons. Firstly, it would significantly enhance versatility by expanding the utility of the app, enabling users to analyze a broader range of datasets effectively. Additionally, this enhancement would improve user accessibility and satisfaction, meeting their evolving needs for a more comprehensive data analysis tool. Furthermore, integrating categorical data support would streamline the FACET system, consolidating multiple functionalities into one cohesive and user-friendly experience. These enhancements are crucial to ensure the app stays relevant and caters to the evolving needs of its users.

5.2 Conclusion

With the continued adoption of artificial intelligence across industries the demand for Explainable AI only grows. Using a novel method to generate counterfactual regions, FACET provides a number of flexible and actionable solutions explaining how users can obtain their desired outcome. To allow for users to set their own requirements and query FACET, the team developed an interactive user interface and application. The application then underwent extensive testing to ensure proper functionality in a number of scenarios and proved to meet the set goals for functionality.

While the application may achieve all the foundational goals FACET was designed for, the application can still be improved. The UI can be made more accessible to people with various disabilities that would otherwise hinder their experience. This process can be better aided through user testing, allowing a wide range of audiences to experience the application in its current state and provide much needed feedback and critique to improve the overall experience. Furthermore, the application itself has a few remaining issues that need to be resolved. Alongside styling and other improvements, FACET itself can be expanded further to be able to process non-numerical data, allowing for more accurate explanations and a wider range of features for users to explore. The creation of the FACET interface presents a promising first step into creating an application users can use to understand and use AI, ending the ‘black-box’ era of the AI.

Citations

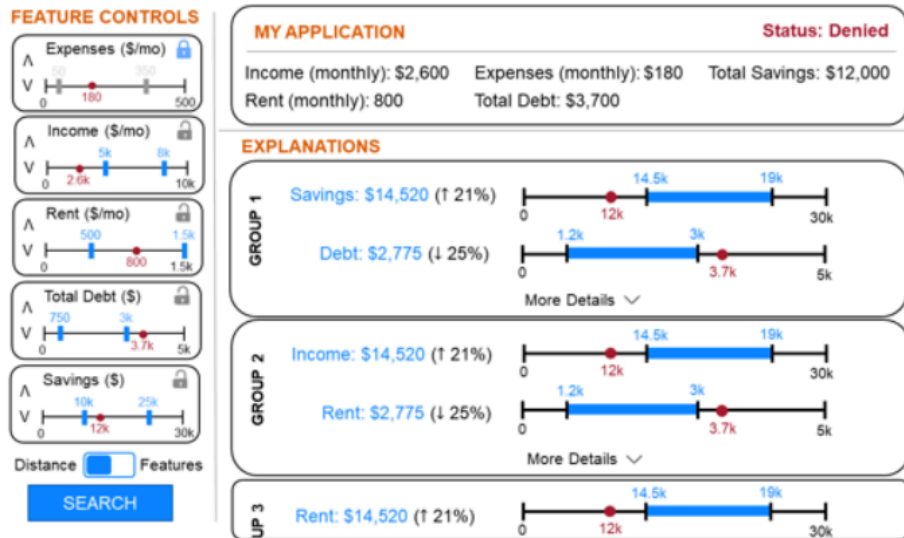
- Abiola, M. (2023). Node.js vs. Flask: Key Advantages, Disadvantages, and Differences. HostAdvice, <https://hostadvice.com/blog/web-hosting/node-js/node-js-vs-flask/>.
- Chamorro, P. (2023). 5 simple ways to increase UI/UX accessibility. BairesDev Blog, <https://www.bairesdev.com/blog/increase-ui-accessibility/>.
- Chui, Michael, et al. "The State of AI in 2022-and a Half Decade in Review." *McKinsey & Company*, McKinsey & Company, 6 Dec. 2022, www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2022-and-a-half-decade-in-re-view.
- Dandl, S., Molnar, C. (2023). Interpretable machine learning, ch. 9.3. Github, <https://christophm.github.io/interpretable-ml-book/counterfactual.html>
- Henry, S.L. (n.d.). Accessibility in User-Centered design. Just ask: Integrating accessibility throughout design, <http://www.uiaccess.com/accessucd/background.html>.
- Hod, Dr. Adi. "Council Post: Explainable AI: The Importance of Adding Interpretability into Machine Learning." *Forbes*, Forbes Magazine, 5 Oct. 2023, www.forbes.com/sites/forbestechcouncil/2023/01/13/explainable-ai-the-importance-of-adding-interpretability-into-machine-learning/?sh=475fdaf2a9e4.
- Keane, Mark T. et al. "If Only We Had Better Counterfactual Explanations: Five Key Deficits to Rectify in the Evaluation of Counterfactual XAI Techniques." *International Joint Conference on Artificial Intelligence* (2021).
- Laing, Keith. "Tesla (TSLA) Ordered by NHTSA to Address New Issue over Autopilot Feature." *Bloomberg.Com*, Bloomberg, 29 Aug. 2023, www.bloomberg.com/news/articles/2023-08-29/tesla-ordered-by-regulators-to-address-new-issue-over-auto-pilot.
- Ludlow, Edward. "Paid Driverless Taxis Are Slowly Becoming a Reality." *Bloomberg.Com*, Bloomberg, 16 Aug. 2023, www.bloomberg.com/news/newsletters/2023-08-16/paid-driverless-taxis-are-slowly-becoming-a-reality.
- Miliard, Mike. "One Early Adopter's Tips for AI Deployment Success." *Healthcare IT News*, 7 Dec. 2023, www.healthcareitnews.com/news/one-early-adopters-tips-ai-deployment-success.
- Molnar, Christoph. "Interpretable Machine Learning." *9.3 Counterfactual Explanations*, 21 Aug. 2023, <https://christophm.github.io/interpretable-ml-book/counterfactual.html>
- React Autocomplete Component - Material UI." *React Autocomplete Component - Material UI*, mui.com/material-ui/react-autocomplete/. Accessed 1 Mar. 2024.

- React Slider Component - Material UI.” *React Slider Component - Material UI*, mui.com/material-ui/react-slider/. Accessed 1 Mar. 2024.
- React Switch Component - Material UI.” *React Switch Component - Material UI*, mui.com/material-ui/react-switch/. Accessed 1 Mar. 2024.
- React Text Field Component - Material UI.” *React Text Field Component - Material UI*, mui.com/material-ui/react-text-field/. Accessed 1 Mar. 2024.
- React Toggle Button Group Component - Joy Ui.” *React Toggle Button Group Component - Joy UI*, mui.com/joy-ui/react-toggle-button-group/. Accessed 1 Mar. 2024.
- Usability.gov. (n.d.). User Interface Design Basics. Usability.gov, <https://www.usability.gov/what-and-why/user-interface-design.html>
- VanNostrand, Peter M., et al. “Facet: Robust Counterfactual Explanation Analytics.” *Proceedings of the ACM on Management of Data*, vol. 1, no. 4, 8 Dec. 2023, pp. 1–27, <https://doi.org/10.1145/3626729>.
- W3C. (2020). Web Content Accessibility Guidelines (WCAG) 2 Level AAA Conformance. <https://www.w3.org/WAI/WCAG2AAA-Conformance>
- White, Tom. “Redrob CEO Felix Kim Explains It All: AI Is Transforming Global Recruitment.” *Benzinga*, 14 Feb. 2024, www.benzinga.com/general/24/02/37126730/redrob-ceo-felix-kim-explains-it-all-ai-is-transforming-global-recruitment.
- Ye, S. (2017). 6 Bad UI Design Examples & Common Errors of UI Designers. MockPlus, <https://www.mockplus.com/blog/post/bad-ui-design-examples>.
- “IconButton API.” *Material UI*, mui.com/material-ui/api/icon-button/. Accessed 1 Mar. 2024.

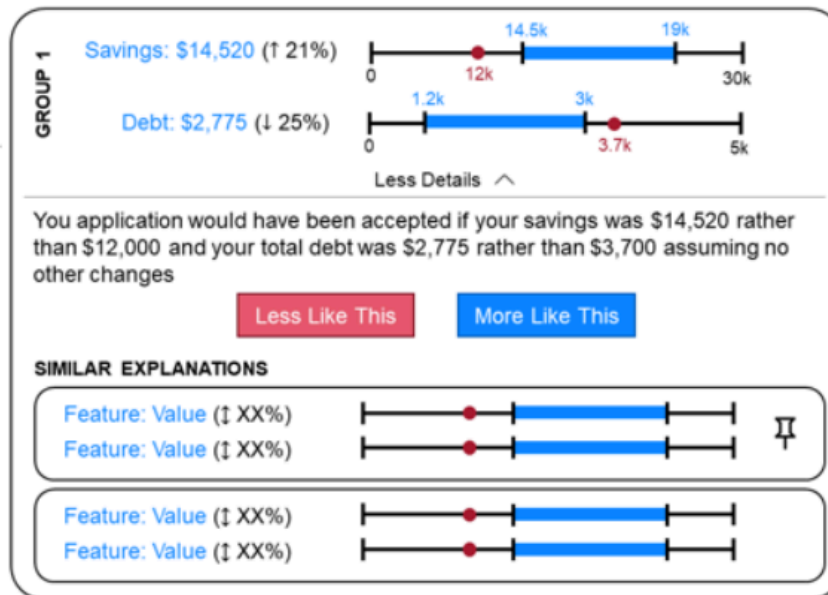
Appendices

Appendix A

Initial FACET UI: FACET interface created by Peter VanNostrand



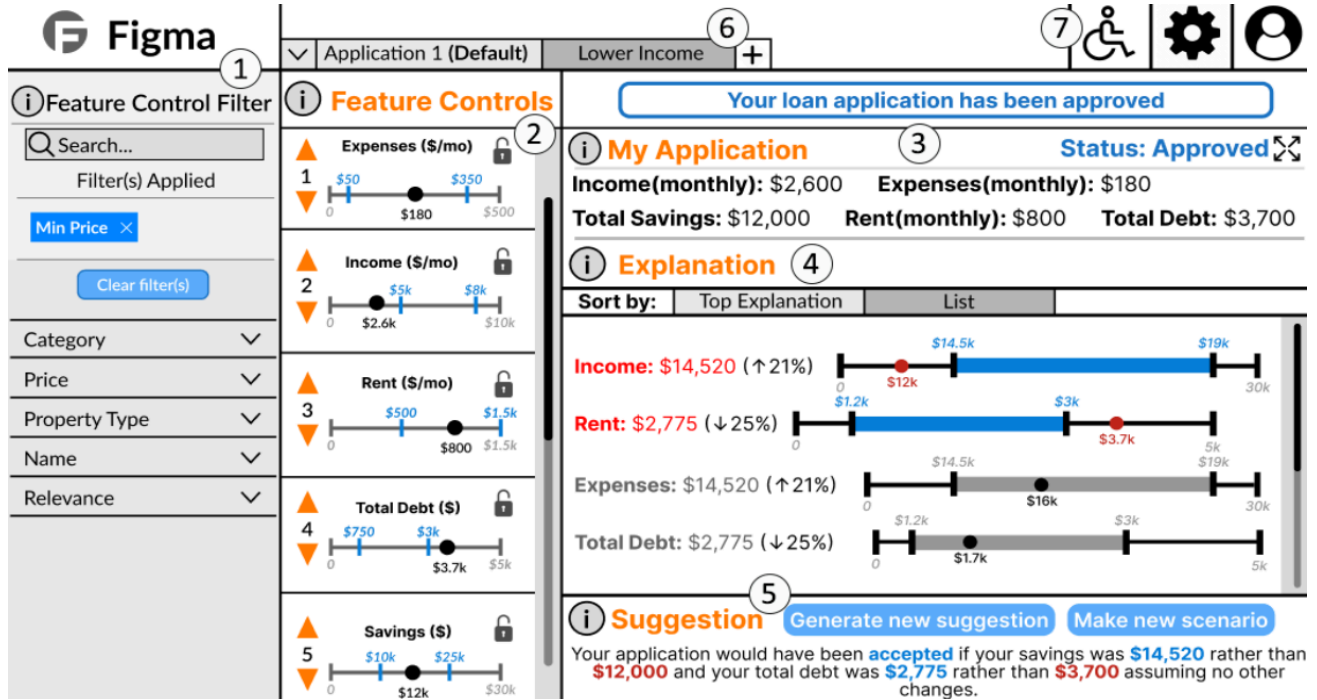
Section 1: Feature Controls, My Application, and Explanations



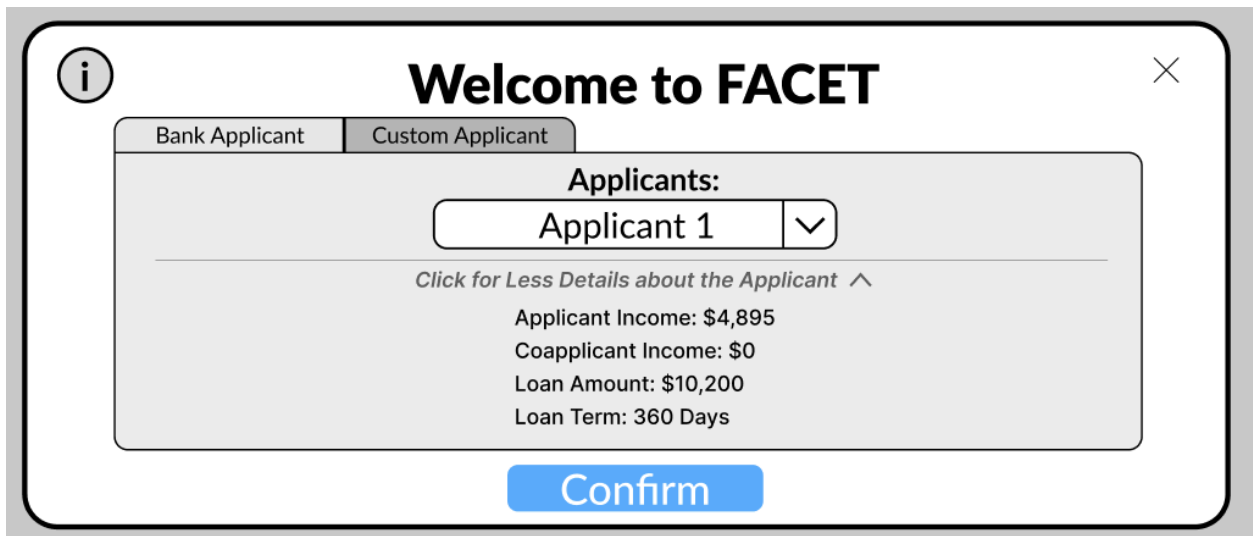
Section 2: Explanations (cont.), Suggestion, and Similar Explanations

Appendix B

Version 1: First Prototype of FACET Web App Layout Designed in Figma



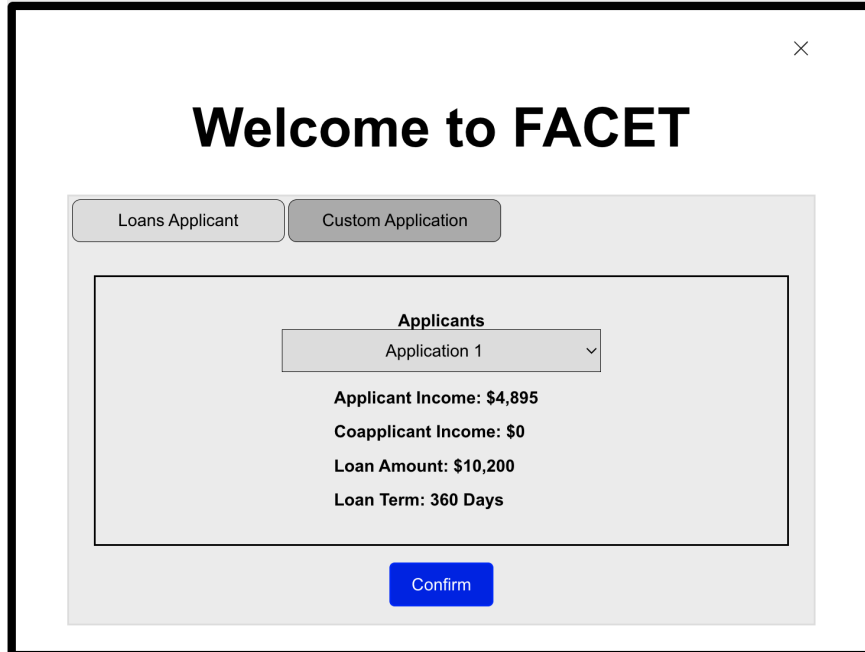
Section 1: Feature Control Filter [1] Feature Control [2] My Application [3] Explanations [4] Suggestion [5] Saved Scenarios [6] Navigation Bar: Accessibility, Settings, Profile [7]



Section 2: Welcome Screen with Applicant Dropdown tab selected

Appendix C

Version 1: Implemented Welcome Screen



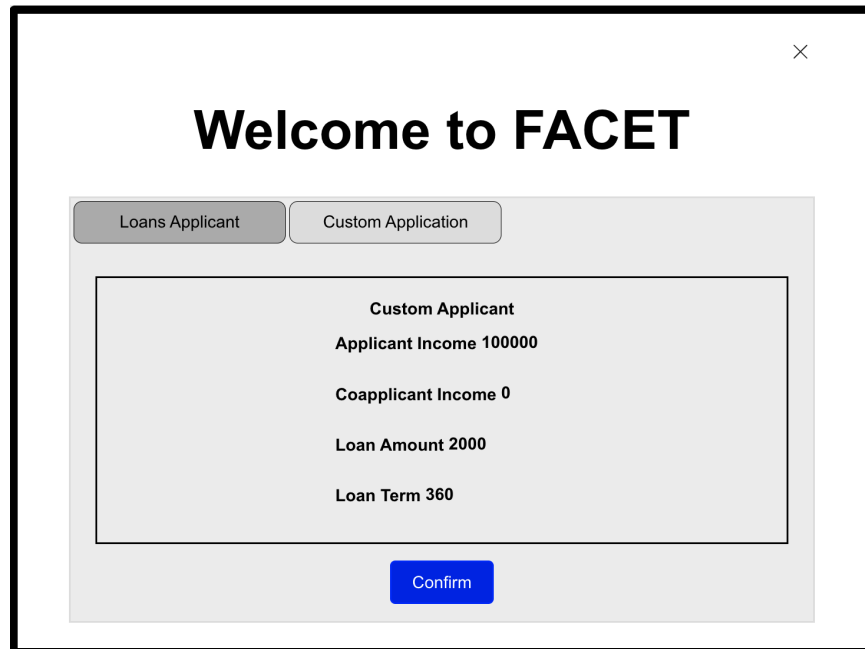
The screenshot shows a window titled "Welcome to FACET" with a close button in the top right corner. Below the title are two tabs: "Loans Applicant" (which is selected) and "Custom Application". The main content area displays the following information:

Applicants
Application 1

Applicant Income: \$4,895
Coapplicant Income: \$0
Loan Amount: \$10,200
Loan Term: 360 Days

A blue "Confirm" button is located at the bottom center of the content area.

Welcome Screen with Applicant Dropdown Selected



The screenshot shows a window titled "Welcome to FACET" with a close button in the top right corner. Below the title are two tabs: "Loans Applicant" and "Custom Applicant" (which is selected). The main content area displays the following information:

Custom Applicant
Applicant Income 100000
Coapplicant Income 0
Loan Amount 2000
Loan Term 360

A blue "Confirm" button is located at the bottom center of the content area.

Welcome Screen with Custom Applicant Selected

Appendix D

Version 2: Implemented Layout of FACET Application

The screenshot displays the FACET application interface with the following components:

- Top Left (6):** Application navigation bar with a back arrow and the text "Application FACET".
- Top Right (2):** "Scenarios" section with a "Clear All" link and the instruction "Save an explanation to create a scenario".
- Middle Left (1):** "Feature Controls" section titled "Prioritize Features" with a slider icon. It contains four sliders:
 - Applicant Income (\$):** Range from \$0 to \$9790, with a current value of \$4895.
 - Coapplicant Income (\$):** Range from \$0 to \$10000, with a current value of \$0.
 - Loan Amount (\$):** Range from \$0 to \$20400, with a current value of \$10200.
 - Loan Term (Days):** Range from 0 Days to 720 Days, with a current value of 360 Days.
- Middle Right (3):** "My Application" section with a red warning box: "Your application has been rejected". It displays:
 - Applicant Income: \$4,895
 - Coapplicant Income: \$0
 - Loan Amount: \$10,200
 - Loan Term: 360 Days
- Bottom Right (4):** "Explanations" section showing four horizontal range charts:
 - Applicant Income:** Range from \$3,671 to \$6,119, with a current value of \$4,895.
 - Coapplicant Income:** Range from \$0 to \$3,651, with a highlighted blue region between \$2,500 and \$2,921.
 - Loan Amount:** Range from \$7,125 to \$19,125, with a current value of \$10,200.
 - Loan Term:** Range from 270 Days to 480 Days, with a highlighted blue region between 420 Days and 540 Days.
- Bottom Right (5):** "Suggestion" section with the text: "Your application would have been APPROVED rather than REJECTED if your Coapplicant Income was between \$2,500 and \$2,921 rather than \$0 and your Loan Term was between 420 Days and 540 Days rather than 360 Days assuming no other changes."
- Bottom Center:** A blue "Save Scenario" button.

Feature Controls [1] Saved Scenarios [2] My Application [3] Explanations [4] Suggestion [5]
Return to Welcome Screen button [6]

Applicant Selection ✕

Applicant Type

DROPPDOWN

CUSTOM

Applicant 0 ▾

Applicant Income \$

Coapplicant Income \$

Loan Amount \$

Loan Term Days

Continue

Welcome Screen with Applicant Dropdown selected

Applicant Selection ✕

Applicant Type

DROPPDOWN

CUSTOM

Custom Applicant ▾

Applicant Income \$

Coapplicant Income \$

Loan Amount \$

Loan Term Days

Continue

Welcome Screen with Custom Applicant selected

Appendix E

Detailed Testing Criteria and Plan

1. Flexibility:

- App should work with other datasets
 - Cancer dataset
 - Alternate loan JSON
- App should properly read from json file
 - App should work with different feature amounts (less, more)
 - App should handle different units
 - App should handle non-existent units
 - App should handle different semantic mins and maxes
 - App should handle displaying different decimal places for feature values
 - App should handle exponential weighting

2. Edge Cases:

- App should work when no valid explanations are given
- App should be able to save extremely large amounts of scenarios (99+)
- App should handle arbitrary number of constraints (lock/pin/min/max changes)

3. Formatting:

- App should appear UX friendly with different zooms

4. General Functionality:

- Locking feature constraints should work as intended

- Feature is considered unchangeable
 - Explanations should have same range as the original value
- Pinning feature constraints should work as intended
 - Priority should not change
 - Explanations should maintain same priority as if pin did not happen and feature was in the same priority
- Saving a scenario should work as intended
 - Changing values in a saved scenario should be reflected and saved in said scenario
 - Scenario should load with constraints and explanations saved with it
 - Scenario should be able to be closed
 - If there are n scenarios already that fill up the given space, the (n+1)th scenario should have a scroll bar
- Feature controls should display features as intended
 - If there are n features already that fill up the given space, the (n+1)th feature should have a scroll bar
 - Changing the ranges should change the explanations to reflect the constraints
- Explanations section should properly display explanations
 - Each explanation should reflect the given constraints and their priority
 - Each explanation, when entered into the feature controls, should be approved

- If there are $< k$ explanations, it should only display the unique explanations found
- If $k' > k$ explanations is given, then k' explanations should be shown (if they exist)
- If a scenario is selected and the explanation cycle button is pressed, a correct explanation should be provided
- My Application's status section should properly reflect whether or not a loan is approved
 - If a loan is rejected, it should announce it
- Suggestions section should provide a text summary of the explanation section
 - Changes to the constraints and explanations should be reflected here
 - Cycling through the explanations should cycle through the suggestions as well

Procedure:

- (1) Test functionality on loan dataset + default JSON
 - (a) Test some edge cases here:
 - (b) Saving large amounts of scenarios
 - (c) No valid explanations found
- (2) Test flexibility here with different JSON load outs
 - (a) Ensure app works for all of them
 - (b) Fix issues that arise, and retest
- (3) Test on cancer dataset + cancer JSON

- (a) Test more edge cases here
 - (i) Giving many features in controls
 - (b) Test more of flexibility here
- (4) Repeat testing if fixes arise

Appendix F

Testing Documentation

Round 1:

Test Performed	Procedure	Result	Status
Cancer Dataset	Edited config file to specify "cancer" dataset	Errors (see dataset-swap.log) Cause: Hard coded constraints obj Alt Cause: dataset_details.json is not generated correctly when data file does not have header row	Need fixing
Feature Locking	Lock icons are clicked, multiple explanations are viewed to confirm they adhere to the constraint, priority changed, pin applied	Bugs: <ul style="list-style-type: none"> - Layout on 4th feature control is off. Card phases over background - Certain combinations of locks do not generate suggestions, but do generate explanations (ex: all locked except loan term days) 	Need fixing + tracked bug fixing
Feature Pinning	Pinned multiple permutations of the feature controls, tried using other buttons on the control to ensure compatibility	Bugs: Arrows are not grayed out under and below pinned feature cards; consider letting them "jump" over or gray out the button	Need fixing (not doing)
Feature Priority Swapping	Swap features around and ensure that the explanations that get generated appear to reflect a change	Works as intended	OK
"Prioritize	Set various priorities to features,	Works as intended	OK

Features” toggle	and toggled the switch. Repeated a few times		
Setting Feature constraints	Changed the ranges on the feature controls to see if explanations correctly get updated to reflect the constraints	Bugs: - Explanation range gets grayed out when range includes current value on all but last feature (include screenshot)	Need fixing
Handle no valid explanations	Locked all feature controls to signal facet that no changes can be made	FACET reported that no explanation could be found and to relax the constraints in the explanations section, and reported that no suggestions could be given.	OK
Saving large number of explanations	Repeatedly spammed the “save scenario” button 100 times	No noticeable lag, and each saved scenario loads in a reasonable amount of time	OK
Saving explanations	Generated various explanations from different constraints, saved the scenario, and then swapped between scenarios to ensure the explanations lined up	Intended behavior observed	OK
Saving a specific explanation given constraints	Generated various explanations, chose one that wasn’t the first and saved it; swapped between scenarios	Intended behavior observed	OK
Scenario numbering	Saved 10 scenarios, closed different permutations of them; repeated 7 times	Intended behavior observed; numbering continued based on the total number of scenarios saved in a session	OK
Scenario saving constraints	Set a bunch of constraints for the features (includes locking and pinning), saved scenarios, and swapped between them	Different ranges specified correctly get saved Bugs: - Priority order does not save and remains constant - Locking does not save and remains constant	Need fixing

		<ul style="list-style-type: none"> - Pinning does not save and remains constant 	
Swapping to different applicant with saved scenarios	Saved a bunch of scenarios with constraints, went back to application screen, chose applicant 8, 12, and 7, saved scenarios, and swapped between them	<p>Intended behavior observed ONLY when a scenario was NOT selected;</p> <p>Bugs:</p> <ul style="list-style-type: none"> - When viewing a scenario, you cannot load a new applicant by going to application screen and selecting a new applicant; it loads back to the scenario. (includes custom applicants) 	Need fixing
Application Screen loading custom applications	A custom explanation was generated with all positive values	<p>Bugs:</p> <ul style="list-style-type: none"> - Entering absurdly large values breaks the ranges for all future custom applications - When entering values that were reported to be approved, the explanations suggest lowering a feature, including income - When providing realistic values, no explanation can be generated unless you lower ranges for a feature, such as income (see screenshots) 	Need fixing
Working with 1 feature control	In human_readable.json, delete all features except for 1 (TEST WITH MODIFIED CSV)	<p>Bugs:</p> <ul style="list-style-type: none"> - Feature controls still loads in the other features - Explanation section contains ranges for these ghost features, except all values and ranges are 'undefined' 	Need fixing

Working with 2 feature controls	In human_readable.json, delete all but 2 features	Same as above	Need fixing
---------------------------------	---	---------------	-------------

Round 2:

Test Performed	Procedure	Result	Status
Cancer Dataset	Edited config file to specify "cancer" dataset	Bugs: <ul style="list-style-type: none"> - Status section reads "Your Diagnosis has been Malignant"; hardcoded "has been" - Explanation section ranges have hard coded units as "\$" and "days" - Explanation section cannot handle ranges for more than the first 4 features - Feature controls + explanations orderings are off; unsure if related to frontend 	Need fixing
Feature Locking	Lock icons are clicked, multiple explanations are viewed to confirm they adhere to the constraint, priority changed, pin applied	Works as Intended	OK
Feature Pinning	Pinned multiple permutations of the feature controls, tried using other buttons on the control to ensure compatibility	Bugs: <ul style="list-style-type: none"> Arrows are not grayed out under and below pinned feature cards; consider letting them "jump" over or gray out the button 	Need fixing
Feature Priority	Swap features around and ensure that the explanations that get	Works as intended	OK

Swapping	generated appear to reflect a change		
“Prioritize Features” toggle	Set various priorities to features, and toggled the switch. Repeated a few times	Works as intended	OK
Setting Feature constraints	Changed the ranges on the feature controls to see if explanations correctly get updated to reflect the constraints	Bugs: - Lag in generating explanations causes constraints to reset to state when the first constraint was set	Need fixing
Handle no valid explanations	Locked all feature controls to signal facet that no changes can be made	FACET reported that no explanation could be found and to relax the constraints in the explanations section, and reported that no suggestions could be given.	OK
Saving large number of explanations	Repeatedly spammed the “save scenario” button 100 times	No noticeable lag, and each saved scenario loads in a reasonable amount of time	OK
Saving explanations	Generated various explanations from different constraints, saved the scenario, and then swapped between scenarios to ensure the explanations lined up	Intended behavior observed	OK
Saving a specific explanation given constraints	Generated various explanations, chose one that wasn’t the first and saved it; swapped between scenarios	Intended behavior observed	OK
Scenario numbering	Saved 10 scenarios, closed different permutations of them; repeated 7 times	Intended behavior observed; numbering continued based on the total number of scenarios saved in a session	OK
Scenario saving constraints	Set a bunch of constraints for the features (includes locking and pinning), saved scenarios, and swapped between them	Works as intended	OK
Swapping to different	Saved a bunch of scenarios with constraints, went back to	Intended behavior observed ONLY when a scenario was NOT selected	Need fixing

applicant with saved scenarios	application screen, chose applicant 8, 12, and 7, saved scenarios, and swapped between them	<p>Bugs:</p> <ul style="list-style-type: none"> - When viewing a scenario, you cannot load a new applicant by going to application screen and selecting a new applicant; it loads back to the scenario. (includes custom applicants) 	
Application Screen loading custom applications	A custom explanation was generated with all positive values	<p>Bugs:</p> <ul style="list-style-type: none"> - Entering absurdly large values breaks the ranges for all future custom applications - When entering values that were reported to be approved, the explanations suggest lowering a feature, including income - When providing realistic values, no explanation can be generated unless you lower ranges for a feature, such as income (see screenshots) - Values (0-9) are not considered valid positive integers - Does not accept negative values for ANY dataset - When NaN is entered, it cannot be deleted or overwritten without highlighting it and then overwriting it 	Need fixing
Working with 1 feature control	In human_readable.json, delete all features except for ApplicantIncome, and modify the CSV to only have this parameter and the results	<p>Bugs:</p> <ul style="list-style-type: none"> - Feature control loads in the single feature control, but the ranges are NaN - No explanations are found - No suggestions are found 	Need fixing

		<ul style="list-style-type: none"> - Cannot explore different instances Errors (backend): <ul style="list-style-type: none"> - Tuple index out of range in dataset.py, line 162 	
Working with 2 feature controls	In human_readable.json for loans, delete all features except for ApplicantIncome and CoapplicantIncome, and modify the CSV to only have these parameters and the results	Bugs: <ul style="list-style-type: none"> - Initial load up has obscene range values from -9E12 to +1E9 	Need fixing
Changing units, semantic ranges, and decimal places	In human_readable.json for the loans dataset, set the following values: <pre> "feature_units": { "ApplicantIncome": "", "CoapplicantIncome": "Gigatons", "LoanAmount": "€", "Loan_Amount_Term": "Days" }, "feature_decimals": { "ApplicantIncome": 2, "CoapplicantIncome": 3, "LoanAmount": 0, "Loan_Amount_Term": 4 }, "semantic_min": { "ApplicantIncome": 7000, "CoapplicantIncome": 0, "LoanAmount": -4, "Loan_Amount_Term": 0 }, "semantic_max": { "ApplicantIncome": 99999, "CoapplicantIncome": null, "LoanAmount": null, "Loan_Amount_Term": null};}; </pre>	Bugs: <ul style="list-style-type: none"> - Units for “€” in feature controls and status section are glitched, whereas the explanation section is fine - A minimum value for a feature causes visual overlapping of the explanation’s minimum range and the features current value - When ApplicantIncome’s unit was set to “€”, it also appeared glitched out Semantic ranges, decimal places, and units with standard ASCII values work as intended	Need fixing

