User and Developer Manual

Form Buster: Web-Based Registration, Approval, and Tracking

Team Members

Daniel Acosta (dacosta2022@my.fit.edu)
Christopher Demuro (cdemuro2022@my.fit.edu)
Alex Merino (amerino2022@my.fit.edu)
Luka Miodrag Starcevic (lstarcevic2022@my.fit.edu)

Faculty Advisor

Phillip Bernhard (pbernhar@fit.edu)

Table of Contents

1. Introduction

The purpose of this document is to provide instructions and pertinent information for both users and future developers of the Form Buster system. Form Buster's goal is to be a centralized form filling, tracking, and editing system for Florida Institute of Technology. The current system requires emailing pdf files to advisors or registration staff and then not getting any feedback on the progress of the form until its completion. Form Buster aims to solve all the problems that are wrong with the current system and improve on the parts that already work with the system.

2. User Manual

This section of the document is dedicated to providing insight to users on how the web application works and how to use the application for many different tasks. There will be basic tasks that all users can accomplish, then user type specific tasks for students, advisors, and university staff.

2.1. Student/General User Functions

All general pages and functions can be used by any user type, but the student users are restricted to only these general functions. The Advisor and Administrator user types have access to all of these functions plus a few more which will be explained in more detail in future sections.

2.1.1. Authentication

When a user first uses the website, they will only be able to view the landing page, which describes the functionality and overview of the website. In the top right of the page there are two buttons; *"Sign in"* and *"Register"*. The purpose of the buttons are to send the user to the desired authentication page.

2.1.1.1. Sign In

The sign in page is the key to accessing the entire system for any user. There are two input fields which ask for users to input user email and password. If the email and password combination input is incorrect then the web application will show an alert and tell the user which input is affected. The *"Submit"* button allows for the submission of the email and password to authenticate. If the user does not have an account there is a link to the ***Registration*** page.

2.1.1.2. Registration

On the ***Registration*** page, when it first loads, there are three options for users to choose which user type fits them best. After selecting the user type they believe is correct, they are then asked to fill out the personal information required to complete an account. Some information differs depending on the user type, but most information is uniform across all types. If the wrong user type happens to be selected there is a *"Return to Previous Page"* button to choose a different user type.

2.1.2. Dashboard

After signing in, the ***Dashboard*** is the first page that is visible to users. The dashboard contains the simplified views of active forms. On each

form there are four main parts. Going from left to right, the form type (FERPA, Class Registration, etc.) and the date and time the form was created are on the left. Then there are a certain number of circles signifying the amount of signatures required for the completion of the form. The circles can be four colors; green (signed), red (rejected), light gray (unsigned), and dark gray (not applicable). When the mouse is hovered over a circle, the user that the signature belongs to, along with the date the signature occurred will be displayed. The next component, if visible, is the notifier which warns the user that they have not signed the form after 2 days. This component is only available if the two day limit is passed. The last component, displayed as three dots, allows users to view more information about the form or sign the form if they still need to.

2.1.3. Signature Graph

When viewing more information about a form from the dashboard, the user will get sent to a page which has more detailed information of the form, including the ***Signature Graph***. The ***Signature Graph*** is a novel feature to form tracking where the user can see every signature required in a flow chart like structure. The graph displays user type information on default and the current signature status.When a node (circle) in the graph is clicked, a popup to the right of the graph will display more information about the user that the signature belongs to. There are different line types which connect two signatures, with the explanations of each available on the left side of the page in a legend.

Above the ***Signature Graph*** is information about the form itself, (name and date created) along with two links to different parts of the website. The link titled *"View Form Here"* will send the user to the ***Form View*** page. The

2.1.4. Form View

The ***Form View*** page which displays the form in its entirety. A form display consists of the form template, the form data that was filled out, and an input for all signatures that are required for the form. A form template consists of field labels (name, ID, birthdate, etc.) and the corresponding input types (text, checkbox, date, etc.). Since the form has been initialized there is related data that has been filled out, so that data is placed into the matching inputs. Then below the form are signature boxes, which are filled if the signature has been signed or empty if the user has not signed their designated signature. All of these components combined create the ***Form View***.

2.1.5. Menus

On each page, after signing into the application, there is a gray bar on the top which has the user's name which can be clicked. When clicked, the user is sent to the ***Account*** page. This is the only current functionality of the top bar as there are no other general functions that can be placed there. The other menu that is available on each page is the ***Side Menu***. By default, the side menu is closed and is represented by the red box with

three lines (sometimes referred to as a hamburger menu). When clicked, the ***Side Menu*** will open and display page options. The page options for all users include the ***Dashboard, Inbox, Start a Form,*** and ***Settings***. When any of these are clicked in the menu, the application will take you to the corresponding page.

2.1.6. Inbox

The ***Inbox*** page provides a central location for all notifications that are sent to a user. One might receive a notification if:

- All signatures for a form have been signed
- A user rejects a form
- A form is paused/put on hold
- The user can now sign a form

There will be a specific message attached to each type of notification in the ***Inbox*** along with a small simplified status in the top right of each notification. With the ***Inbox*** a user can do three actions with notifications; *Mark All Read*, *Refresh*, and *Delete All*. When any of the buttons are clicked, the described action will occur to the notifications.

2.1.7. Start a Form

Starting a form consists of two parts; the ***Form List*** and the ***Form Initializor***. When clicking on *"Start a Form"* from the ***Side Menu*** the website goes to the ***Form List*** which displays all possible forms that can be filled out. When selecting any form, the user will be taken to the ***Form Initializor*** which contains the current form template selected.
Another novel feature to form registration is the technology of autofill. The information that the autofill uses comes solely from the data submitted during account creation. The website does not gather or track any other information than that given explicitly by the user. Using that data, the input fields that require the account information are automatically filled in, saving time for users. The date is also automatically filled in and cannot be changed. A user then must fill out all the other required information and type their signature themselves before clicking *"Submit"* which submits the form and then sends the user back to the ***Dashboard***.

2.1.8. Form Signature

The ***Form Signature*** page can be accessed from many different pages on the website, but it is also our most secure page of the website. Users only see a link to the ***Form Signature*** if they are able to currently sign the form. If a user has already signed a form they do not have access to ***Form Signature***.
On the ***Form Signature*** page the ***Form View*** is presented as described in ***2.1.4***. The only difference is that instead of a list of signatures, there are two buttons to click at the bottom of the page; *"Sign"* and *"Decline"*. When clicking on *"Sign"* a signature input will appear at the bottom of the page, allowing for the user to sign. When the *"Decline"* button is clicked a signature input will still be displayed but a text box will also appear requiring the user to write a comment on why they are declining the form.

If the inputs are not filled out, the signature will not submit, but once they are filled out a user can hit the *"Submit"* button and submit their signature.

2.1.9. Account

The ***Account*** page displays account information, which is given by the user when creating their account. There is an option to show their password, which will allow the user to see their password. On the left side of the account information there is a small sidebar which allows the user to switch between the ***Account*** and ***Settings*** pages.

2.1.10. Settings

The ***Settings*** page displays the different settings that can be modified by users. At the current implementation of the website the only modifiable settings is whether or not the user would like to receive Email Notifications. Email notifications are actually not currently implemented so the button does nothing substantial when clicked.

3. Frontend Developer Manual

The Frontend manual provides information about the Javascript and CSS files which were created and maintained for the Form Buster project. Each Subsection is split by the file directory each component is in. Each ***.jsx*** file is separated into two main parts; the rendering (HTML heavy) and the preprocessing logic (Javascript heavy).
Text formatting for terms:
- *#id-name*
- *.class-name*
- ***Page.jsx / Page***
- *ReactComponent*
- **function/ statement**

3.1. Account

The ***Account.jsx*** file is a container file which is used to solely hold the subcomponents needed for the account page. Each component is listed below in **3.1.1 - 3.1.3**. The *Outlet* tag is a native React tag which allows for different components to be displayed based what page a user is on. The *Outlet* here changes between ***Information*** and ***Setting*** pages.

3.1.1. Account Side *(ACSide)*

ACSide.jsx contains the side bar on the ***Information*** and ***Setting*** pages. This is the source code for the side bar where based on which page is active in a simple **if** statement. The only thing rendered in this component is the two links in the sidebar.

3.1.2. Information

In the ***Information.jsx*** file the account information for the logged in user is displayed. In the render there is just a list of the account information that is gathered from the React function *useContext* which is how the system can access the information (explained in **3.14**). There is a conditional state in the render which displays *account.advisor* information if the account is a student account.

There is a function called **switchPasswordVisibiilty** which switches the visibility of the user's password from asterisks to the actual password. The function switches when the #*show-password* button is pressed.

3.1.3. Settings

The ***Settings.jsx*** file contains the list of items for the ***Settings*** page, which is only two items at current implementation. There is a function, **switchEmail**, which would switch the database boolean for email notifications for the user, however since email notifications are not implemented the function does not do as such. The function also changes the color of the #*notif-bl* button rendered in the component as well, which is the same button that triggers **switchEmail**.

3.2. App

The ***App.jsx*** file is the main driver for the web application, hence its name. This file contains the most amount of imports in the project since it needs to collect each component so they can be displayed. The way that components are displayed is through the *BrowserRouter* which displays a specific component based on the route (website path). Within the *BrowserRouter* is the *Provider* tag, which allows for React to use a centralized storage system for user information. Each *Route* inside the *BrowserRouter* has a path and element attribute which describe the element that should be displayed at the specified website path.
There is a *useEffect* call in this file which calls the backend to get any possible cookies from the web browser. If there are cookies then the user data will be passed on to each component.

3.3. Authentication

3.3.1. LogIn

The ***LogIn.jsx*** file contains the source code for the sign in page which, when rendered contains two inputs for email, a submit button, and a link to the ***Register*** page. There is a function, **handleSubmit**, which sends the inputted email and password to the backend (**4.1.3**) and navigates the application to the home page if the combination is correct or prompts the user to try again in the case of an incorrect combination.

3.3.2. Register

The ***Register.jsx*** file contains conditional rendering based on the enumerated state variable, **disp**, created at the beginning of the file. **disp** starts as an empty string, which will display the choosing part of the ***Register*** page. Based on which user type is chosen, the conditional render will change to that specific type. Each type has a different amount of inputs that are needed to be filled out. A user variable is created which will hold the eventual data which will be submitted to the database. When inputs are modified, the **handleChange** function is called which just updates the user variable. When the *#reg-sub* button is pressed, the **handleSubmit** function is called which sends the user information to the database (**4.1.2**) and upon return, navigates to the sign in page if data is stored, or requests the user to input information again if there is an error.

3.4. Form

The ***Form.jsx*** file contains the HTML required to format all possible form templates in a uniform way, which covers the rendering in this file. There are many parts to the preprocessing logic, starting with *useEffect* which fetches the form template based on the url path of the page. There is a **handleSubmit** function which sends the inputted form data to the backend and navigates the user back to the ***Home*** page. There is another function, **autofill**, which will fill in certain form inputs on page load. The last function, **getDate**, returns a specific date-time format which will be used in the **autofill** function to autofill the date input.

3.4.1. Display

The ***Display.jsx*** file is very similar to the ***Form.jsx*** file since both display a form template. The difference with the ***Display*** page is that the form is already filled out. Within the *useEffect*, there are two functions, the first is **getFormData** which gets the data for an active form based on the url path. The second, **getFormTemplate**, uses the response from **getFormData** and retrieves the form template. The last function that sets up the page is **inputAnswers** which goes through each input in the form template and checks to see if there is an answer for that input, if so the answer is inserted.

3.4.2. SignBar (*Display/Signbar.jsx)*

The ***SignBar.jsx*** file renders a singular input with the associated user's signature for a form. There is a *useEffect*, which calls the **fetchSig** function that accesses the backend to get a signature and the user for that signature. Then the signature is recreated from that data and inserted into the input.

3.5. FormBuilder

3.5.1. Structure

The form builder component can be found in ***FormBuilder.jsx***. It is split into three sub-components, namely the **Editor**, **Toolbar**, and **Util** components. The main **FormBuilder** component handles putting together the form editing interface and calling the appropriate methods from each of its sub-components.

3.5.2 Editor

The **Editor** component can be found in ***Editor.jsx***. It creates a rich text editing interface with a custom Insert menu to ensure that the user can insert custom form fields. The rich text editor is based on the React Quill package, and in-depth documentation about the standard parts of the Quill interface can be found at https://github.com/zenoamaro/react-quill.

3.5.3 Toolbar

3.6. FormList

The ***FormList.jsx*** file renders a list of links to start a form. There is a *useEffect* function with the **fetchForms** function nested inside. **fetchForms** accesses the backend and gathers all form templates from the database. There is also a smaller function, **formsAndLinks**, which goes through all forms collected from **fetchForms** and replaces all the dashes with spaces for readability in the GUI.

3.7. Home

The ***Home.jsx*** file is mainly a container file which simply holds the ***FormTracker*** component. There is also a function, **updateType**, which will update which type of form will be displayed based on the type of form clicked in the list under the Form Tracker title.

3.7.1. FormTracker

The ***FormTracker.jsx*** file renders the list of active forms for the logged in user. Using a conditional statement before rendering, different forms are rendered based on if the user is an Administrator. Both renders go through each list of forms and creates a ***TrackedItem*** component. For the Administrator, the forms used come from **getForms** in the *useEffect* call. The *useEffect* is only called when the user is an Administrator. **getForms** retrieves every form from the backend and stores the data. The other users get the forms from their user accounts. Administrators are likely not to have any forms sent directly to them.

There is an **if** statement which changes the *ftype* variable to a different name to render the title. A future improvement to this file is making it so that not every form is turned into a ***TrackedItem*** and then checking if the form should be rendered or not in that component as it would reduce overhead and processing/loading time.

3.7.1.1. TrackedItem

TrackedItem.jsx is one of the most involved source code files of the project at around 200 lines, even though each component is modularized. A ***TrackedItem*** is a bar across the screen with up to four separate parts when rendered. The first part is the information inside the *.form-info* div, then the ***ShowSig*** component, then possibly the ***Warning*** component, followed by the component signified by *.end-bar*.

Within the *useEffect* call there are two backend calls, **fetchForm** and **getSignStatus**. **fetchForm** gets the data for an active form and **getSignStatus** gets a signature from the backend database based on the formID of the form and the userID of the user. After the *useEffect* call ends, the render begins which calls the **getFormName** function. Using the data from the active form, the **getFormName** function gets the name of the form from the backend. There is an **if** statement which checks if the user is of the Administrator user type, which if true will store HTML data for the delete option. The delete option, if clicked by an Administrator

will call the **deleteForm** function and upon refresh the form will not be visible in the frontend.

3.7.1.2. Warning

The ***Warning.jsx*** file renders a warning sign and some text if the current user has not signed a form which has been accessible to them for over 2 days. The amount of days is calculated based on the start date of the form and the **if** statement is how we check if it has been more than 2 days.

3.7.1.3. ShowSig *(Home/FormTracker/TrackedItem/ShowSig)*

The ***ShowSig.jsx*** file renders a circle of a color based on the status of the signature. When the circle is hovered a textbox appears of the same color with information about the signature. In the *useEffect* call the **fetchSig** function gets the signature and the user tied to the signature.

3.7.2. FullForm *(Home/FullForm)*

The ***FullForm.jsx*** file contains the rendering for the full form container which includes a title, some links, and the Signature Graph. In the *useEffect* call there are two functions, **fetchSig** which collects the signature of the current user that is connected to this form and **fetchForm**, which gets the data for the active form from the url path. Similarly to ***TrackedItem*** the **getFormName** function gets the name of the from as the component is rendering.

3.7.2.1. SigTree *(Home/FullForm/SigTree)*

The ***SigTree.jsx*** file contains the logic for setting up the ***Signature Graph*** which includes calling the ***SigLink*** and ***SigNode*** components. The preprocessing in the file creates the nodes and links and connects them according to the form template data. Then in the render the **goNode** and **goLink** functions are called to create a ***SigNode*** and ***SigLink*** respectively.

3.7.2.2. SigLink *(Home/FullForm/SigTree)*

The ***SigLink.jsx*** file renders a line between two ***SigNodes***. The *useEffect* call runs the **fetchSig** function which will check the signature connected to both nodes that the link connects. Using the information from the signatures the color and line decoration of the line is determined.

3.7.2.3. SigNode *(Home/FullForm/SigTree)*

The ***SigNode.jsx*** file renders a svg circle which represents a node in the ***Signature Graph***. The *useEffect* call runs the **fetchSig** function which returns the signature and user attached to the signature. The other function in the file, **changeDisp**, creates a ***SigSide*** element on the side of the graph.

3.7.2.4. SigSide *(Home/FullForm/SigTree)*

The ***SigSide.jsx*** file renders a text box on the side of the ***Signature***

Graph which displays some information about the user that is tied to the signature.

3.8. Inbox

The ***Inbox.jsx*** file contains the source code for the notification inbox of the Form Buster website. The rendering includes an ***InboxInteractionBar*** component, a dynamic list of ***InboxMessage*** components, and a dynamic popup menu for displaying additional details about messages in the inbox. The inbox messages are updated and refreshed by calling the **fetchInbox** function. Notification popup logic is controlled by the **openNotifPopup** and **closeNotifPopup** functions.

3.8.1. InboxInteractionBar

The ***InboxInteractionBar.jsx*** file renders a bar at the top of the inbox that contains buttons for executing various passed-in functions, such as *Refresh*, *Mark All Read*, and *Delete All Read*. These buttons execute the functions passed in as arguments to the constructor of the InboxInteractionBar.

3.8.2. InboxMessage

The ***InboxMessage.jsx*** file renders an inbox message based on some details about the form, including its *formID,* whether the form was rejected or not, a reason for that rejection, whether the message has been read or not, the *formName* of the form, as well as the message type. The inbox messages are split into two types, namely approval/rejection and signature requests.

3.9. LandingPage

The ***LandingPage.jsx*** file contains the source code for the landing page of the Form Buster website. There is only just a rendering of HTML for the page with the ***SignBar*** at the top.

3.9.1. SignBar

The ***SignBar.jsx*** file renders the top bar across the landing page with the ***Register*** and ***SignIn*** links.

3.10. Menu

The ***Menu.jsx*** file is the source code for the ***Menu*** which is on the left side of every page. The ***Menu*** provides links to other pages of the website including ***Settings, FormList, Home,*** and ***Inbox***. For administrator users there is an extra link to ***Form Builder***. When the ***Menu*** is not open, there is a small square in the top left corner. The render changes when the square is clicked to the full menu, then changed back when the back arrow in the menu or anywhere off the menu is clicked. This is kept track of by the **changeDisp** function.

3.11. SignForm

The ***SignForm.jsx*** file is the source code for the ***Signing*** page on the website. The rendering contains a ***Form*** component along with a section to sign the form in the *.choicePicks* div. In the *useEffect* call, there are two functions. The first, **getFormData**, gets the data for the current form from the database. **getFormTemp** retrieves the template of the current form based on the response

from **getFormData**. Outside of the *useEffect* there are two functions, **inputAnswers** and **displaySig**. **inputAnswers** goes through the form data and the form template and fills in all of the data into the template while **displaySig** changes the signature display based on whether sign or decline was picked.

3.11.1. SignBox

The ***SignBox.jsx*** file holds the HTML to conditionally render the input(s) for a user when they are signing a form. The function **submitSig** runs when the submit button is clicked in the render. If the signature is accepting the form then the **updateSig** function is called which updates the signature in the database as a signed signature. If it is a decline signature **decSig** is called which will update the signature to 'rejected' status. At the end of both signatures **lastStep** is called which navigates the website to the home page. When *.choicePicks* buttons are clicked the **moveWind** function is called which moves the window to show the signature inputs.

3.12. TopBar

The ***TopBar.jsx*** file renders the ***TopBar*** on every page of the website, which shows a greeting and the user's name. When the name is clicked it will navigate the website to the ***Account*** page.

3.13. User

The ***User.jsx*** file solely creates a React object which will store data throughout the website. The *createContext* function is used which allows for the entire website to have context for some specific piece of data. The *User.Provider* tag in the ***App.jsx*** file wrapping around all *Routes* allows each component to receive the ***User*** context.

4. Backend Developer Manual

4.1. User Authentication

4.1.1. Component Location

The components that handle user authentication are located in the ***/backend/controllers/userFunctions.jsx*** file. This file contains two main functions. The ***register*** function creates a new user account and stores it in the database. The ***logIn*** function locates an existing user account (if present) in the database and authenticates the session.

4.1.2. User Registration

When a new user attempts to register, the ***register*** function is called. It first checks to see if a user with the same ID number has already been registered. If so, the new registration is rejected. If not, the new user account is created and saved in the database.

4.1.3. User Authentication

When a user attempts to log in to their account, the ***logIn*** function is called. It first attempts to locate an account associated with the given email address in the database. If this lookup fails, it returns an error

message stating “Email is not registered.” Otherwise, it attempts to compare the given password hash to the one stored in the database for the given user account. If they do not match, an error message is returned stating “Incorrect Password.” Otherwise, the user is then signed in to their account.

4.1.4. Developer Recommendation

It is the recommendation of the developers that the user authentication and registration components be replaced with the institutional Single Sign-On authentication system in any production deployment of this system. Not only will this increase the security of user accounts, but it will make for a smoother experience for student and faculty users alike, and will reduce the possibility of fraudulent user accounts being registered by unauthorized users.

4.2. Database Schema

4.2.1. Form Template Format

Form templates are stored in the database as strings of HTML that define the appearance of the form. Input fields must be located in a div class entitled “custom” in order to be picked up by the saving function. The Template Editor takes care of this automatically, but this rule must be followed when creating templates by hand or using external software. Form templates each have their own unique identifiers given in the “id” field. The schema definition is located in ***/backend/schemas/FormTemplate.js***.

4.2.2. Active Form Format

Active forms, like form templates, are identified by a unique identifier given in the “id” field. Active forms store the identifier of the form template they are associated with in the “formType” field along with the form status, creation date, comments and feedback, and signatory information. The schema definition is located in ***/backend/schemas/CurrentForm.js***.

4.3. Form Tracking

The signature graph is generated from the signatory information stored with the active form. Inbox notifications are generated when signatures are added to forms by signatory parties. There are methods to query the database for forms that require attention from a given user. These methods can be found in ***/backend/controllers/CurrentFormFunctions.js*** and ***/backend/controllers/inboxFunctions.js***.