Software Requirement Document

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)

Version 1.0

Feb 24, 2025

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this document is to outline the requirements for the features of Form Buster, a web-based application designed to host the storing, completion, and distribution of registration forms at Florida Tech. This document describes the general overview of the project, features, functional requirements, and non-functional requirements.

1.2. Scope

The scope of the Form Buster project is to replace the current registration process at Florida Tech. The idea is to create a registration portal similar to the Financial Aid portal that is in place at Florida Tech, which houses all registration forms and allows students, staff, and administrators to complete them.

1.3. Definitions

- GUI: Graphical User Interface
- MERN stack: Full stack web development tools/languages (MongoDB, Express, React, and NodeJS)

1.4. References

This document follows the “IEEE Recommended Practice for Software Requirements Specifications.”

2. General Description

2.1. Project Perspective

Form Buster is a web application that simplifies and streamlines the registration process at Florida Tech. The system will allow users of all types to fill out, approve, and track a multitude of registration forms that Florida Tech utilizes. The system is also completely located in one centralized web application to ensure ease of access for all users.

2.2. Product Functions

Form Buster, in true web application fashion, has multiple functions. The main goal, as described above, is to simplify the registration process through these functions:

1. Form Completion: Users utilizing HTML Forms, can fill out registration forms online.
2. Form Tracking: Users can track the progress of a form after completing said form.
3. Form Layout Editing: Administrators can modify forms in the form database.
4. User Registration and Login: Users can register/log in to save their data and active forms.
5. Email Notification System: Users can enable/disable email notifications for when their forms are updated.
6. Dashboard: The users can view their dashboard, which has a centralized and concise list of active forms for the specific user.
7. Inbox. The users can view updates and notifications through their inbox in the Form Buster application

2.3. User Classes

There are three user classes in the Form Buster system. The purpose of Form Buster is to simplify the registration process, which is mainly for students; however, all users, when it comes to registering, will be greatly and positively affected by the changes. All user types will log in using the same login module and be shown slightly differing dashboards based on their user type.

2.3.1. Student

The student user class will make up most of the user pool since there are about 10-15 students per advising faculty member. The students will be able to start any available registration forms and track those same forms. They will also have access to a personal dashboard and inbox as well.

2.3.2. Faculty

Faculty users will have the same functions as students, where they can start forms and track those same forms. Faculty also have the ability to deny any form that is sent their way. Since the faculty are advisors for some students, they can deny a class registration form for a specific reason and send it back to the student for them to update.

2.3.3. Administrator

Administrators can use the same functions as students and faculty members alike. The main difference for administrators is that they will have access to the Form Layout Editor. Most administrator users will be employees at the registrar, where they have the final say on the registration status for students.

2.4. Operating Environment

The web application will be created through the MERN stack and hosted on Amazon Web Services (AWS) to ensure the website is public and accessible.

2.4.1. Portability

For a web application to be fully portable, it must be able to work on all different browsers. The main browsers we will focus on ensuring compatibility/portability are Google Chrome, Opera, Safari, Firefox, and Microsoft Edge. We will work to ensure that the web app is mobile-responsive and functions properly on devices of all sizes and form factors.

2.5. Design and Implementation Constraints

Due to a lack of funding, the project will be run on a free server through AWS which has a limited number of requests per month, along with other limitations.

2.6. User Documentation

A user guide and documentation on how to use each module will be provided when the project is completed.

3. External Interfaces

3.1. Graphical User Interface (GUI)

The graphical user interface (GUI) is where the system displays the information it contains in a way that is pleasing to the users. It is the primary user interface for the application. It initially presents the user with a Dashboard that provides a summary of available information to the user on what forms they have that are still in progress or that they need to approve. It also provides access to the Inbox so that the user can check their notifications and the configuration menu. The Dashboard for Administrator users will also grant access to the form editor and template manager.

3.2. System Interfaces

Our database will interface with the web application to pull form layouts, active form data, and user information. This is our main interface to another system and the most important one to keep secure. The database will keep the user data, which will be encrypted, and send the data to the Form Buster system depending on the needs of each user.

3.3. Hardware Interfaces

Both the web app and the backend databases require server hardware to run on. The web app must be able to run in any modern web browser for any modern device, be it a desktop PC, laptop, or smartphone.

3.4. Software Interfaces

The GUI will be presented in a web app that can run in a modern web browser. The web application will also interface with AWS and the EC2 server on which it will run.

4. Functional Requirements

4.1. Form Completion

The system shall allow all user types (student, staff, administration) to begin a new form and fill out the information on the form. All forms will be displayed as HTML form objects, with the initial inputs to the application being the form that was selected by the user. The system will then access the form database, pull the form fields required, and display them on the GUI. The user then fills out the form with the necessary information

4.2. Form Tracking

The system shall allow a user to track a form that requires their input. When a form becomes active, a list of users will be created and attached to our form data structure. This list of users will be able to view the form throughout its lifecycle and check its progress.

4.2.1. Example

When a student fills out a class registration form, they will automatically have the form added to their tracked forms list. Since it is a class registration form, the student's advisor and the Dean of the College that the student is a part of also get the form added to their tracked forms list since their input is needed for the form to complete its life cycle.

4.3. Form Layout Editing

The system shall allow the Administrator users to create, modify, publish, unpublish, and delete forms. Due to the ever-changing registration environment, our system needs to allow the modification of forms.

4.3.1. Form Creation

The system shall allow the administrators to create a new school form. When creating a new form, they can pick from any form template in the database or start with a blank form.

4.3.2. Security

The system shall only allow users under the type Administrator to use the Layout Editor.

4.3.3. Example

Say the Administrators feel that the student's school identification number is no longer necessary when registering for classes. An administrator shall log in and navigate to the list of all forms in the database, pick the class registration form, and click the edit button. The administrator will now view an editable view of the HTML form and can click delete on the student identification number input. The administrator can then save the updated form, which will create a new form layout in the database for later review. Once all checks are made, the administrator can publish the new form in place of the old class registration form. Alternatively, they can immediately publish the change to all users, which will allow the new form layout to replace the existing layout in the database.

4.4. User Registration and Login

The system shall allow new users to register for an account and allow registered users to log into the system with their unique credentials.

4.4.1. Registration

The system shall allow new users to register when first using the site. The users will be asked to pick their user type (student, faculty, administrator) and then create a password along with their email address. During registration, the users will also be asked to fill in important information such as name, major, student identification number, and advisor. This information can be used to autofill form inputs to simplify the form completion process.

4.4.2. Login

The system shall allow a user who has already created an account to navigate to the login module where they can input their email and password to log in.

4.4.3. Form Autofill

The system shall autofill certain form inputs based on the user information connected to a user's account. This information, which is given by the user during the registration process, can be modified by the user at any time on their profile page.

4.5. Email Notification System

When enabled, the system shall send email notifications to users when a form has received a signature, been denied by an interested party, been completed, or when any other changes have occurred in the completion status of said form. Notifications shall inform the user of the status change and what action (if any) should be taken to further complete the aforementioned form. The email notification system shall be optional and may be completely disabled at the discretion of the user. This information which is sent through email, is a replica of the Inbox module described below.

4.6. Dashboard

The system shall have a centralized dashboard, where a user, upon logging in, will be able to see a summarized list of all the forms that currently need their attention, along with the status of their completion. This list shall be categorized by whether the current user initiated the form or whether they are a receiving party. The dashboard shall also allow the user to initiate a new form or manage the list of form templates should their permissions allow. The dashboard shall allow the user to access, whether directly or indirectly, all other functionality available in the app.

4.7. Inbox

The system shall list all the user's notifications through an inbox module. These notifications shall be identical to the notifications distributed by the Email Notification System, but the Inbox shall not be disabled by the user. The Inbox shall notify the user when a form has received a signature, been denied by an interested party, been completed, or when any other changes have occurred in the completion status of said form. Notifications shall inform the user of the status change and what action (if any) should be taken to further complete said form.

5. Nonfunctional Requirements

5.1. Security Requirements

There are many security requirements that must be taken into account. The main concern is that users who do not have an account are able to access user data, or that users are able to access other user data that they should not be able to access. The first problem will be solved by creating a landing page for all users that are not currently logged into an account where they can view the details about the system, but not directly access the system. The second problem can be solved by ensuring that the user data is encrypted and not accessible for any users, through an API endpoint for example. Students and faculty will have access to the name and email of all user accounts for correspondence about forms or other important information.

Users must also be only allowed to complete tasks that their user types are able to complete. The users are in a hierarchical structure of available functions, where administrators can use all functions, faculty can use most functions, and students can only use a select few functions. It needs to be ensured that students and faculty cannot be allowed access to the form layout editor.

Another security concern is that a student signs up under the student user type and not the faculty or administrator user type. This will be tackled by adding a checkpoint when creating a faculty or administrator account. Any administrator can look at the list of faculty members and administrators who have requested to register and can accept or decline the account based on whether the user is actually a faculty member or an administrator and not a student or other user trying to hack into the system. This would require there to be a main administrator that will be added to the system to release to allow for other administrators to be added.

5.2. Performance Requirements

The system must perform under all test cases, including edge cases, and be completely reliable, following these requirements as well:

- The system shall process and send forms in under 5 seconds
- The system shall process forms and updates with 100% reliability and accuracy
- The system shall register users in under 5 seconds
- The system shall register users with 100% reliability and accuracy

These performance requirements are similar to the requirements of corporate projects where reliability is of the utmost importance.