Leshi Chen
Dylan C. Conklin
Jeff McHale
Duc Minh Ma
Alex Harris
Vincent Liang
Jaafar Rodgers
Ricardo Sanchez

System for Anonymous Feedback (SAFE)

Project Document

Table of Contents

1. Overview of Project

SAFE is an anonymous feedback system for students to leave questions, comments and concerns for the Portland State University Computer Science department. It allows the department chair to receive feedback given by anonymous users.

Senders are allowed to leave feedback that is delivered through internal mailing systems on the PSU network to the department chair, whereby the chair is able to view and make note of the information provided.

The Receiver (department chair) receives all feedback into their PSU gmail account. This message is sent directly from SAFE the moment a Sender submits feedback. The message contains only the information the Sender has elected to provide, and does not natively contain any potentially identifying information.

2. Project Requirements

2.1. Critical Features

System Requirements

- A webpage that must have a responsive and accessible design
- Database storing feedback provided by a Sender

Sender Requirements

- **Anonymity: Senders should have confidence that their identity will not be revealed**
    - No profiling data tracked or stored
    - Section on webpage that describes how user identity is secure
- **Low barrier to entry: UI should be lightweight, easy to use**
    - Senders access feedback form with no authentication
    - Simple and easy to use message form

Receiver Requirements

- **Notification system**
    - Email sent informing new feedback has been left

2.2. Rejected Features

- **Use of MongoDB for database**
    - PSU contracts with Cassandra and PostgreSQL for database needs.
        - Team is more familiar with PostgreSQL.
- **SSO for Senders**
    - In theory, would work well on Sender side to reduce spam and access replies with ease
    - In practice, compromises Sender identity through authentication

2.3. Technologies Used

- **Frontend | Backend**
    - TypeScript, React & MaterialUI, Node.js & Node Package Manager (npm) , Express, PM2, and JSON
- **Shell Script**
    - Python
- **Database**
    - PostgreSQL
- **Formatter**
    - Prettier, ESlint

2.4. Deliverables

- A functional webpage that facilitates anonymous feedback to intended recipient(s) (Currently the only Receiver is PSU CS Department Chair Mark Jones for this version of the system)
- Source code
- Appropriate documentation

2.5. Stretch Goals

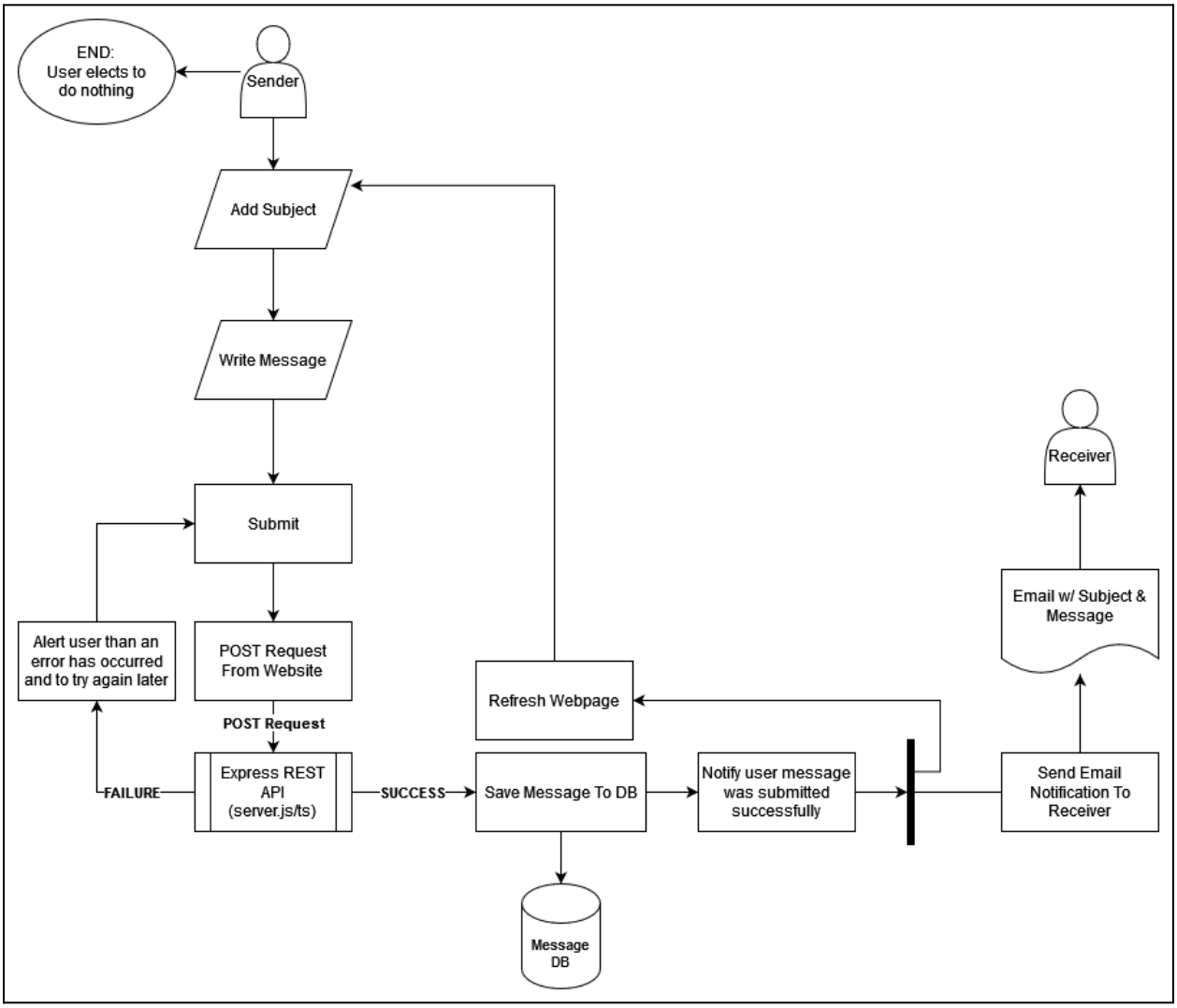
- A comprehensive dashboard that Receivers login to, viewing/interacting with feedback Senders have provided
- SSO authentication on Receiver side to access the dashboard

2.6. Completed Stretch Goals

- Deployed to the school's server (feedback.cs.pdx.edu)

3. System Architecture

3.1. System Design



The overall system architecture involves two separate actors: a Sender and a Receiver. Senders are able to write and submit feedback as an active participant with the webpage, while Receivers are slightly more passive. Senders are prompted for a subject and then are able to add a message of a certain size. Once senders are satisfied with their message, they will click a “Submit” button where a POST request is made by the REST API. If the REST API fails, the Sender is notified and asked to try again later. If the request succeeds, we save the message, notify the user and then fork where two separate actions occur simultaneously: the page refreshes for the Sender, and the REST API sends an email to the Receiver with a set of calls at the system level.

For the Receiver, they are able to patiently wait and check their emails for notifications that feedback has been submitted. From here the Receiver then takes an active role in reading the feedback, taking notes however they wish on the feedback, then addressing the feedback how best they see fit. Future versions of SAFE should implement the prototyped Receiver Dashboard mentioned in the "Backlog" section.

3.2. Sender Functionality

Sender functionality is entirely contained within the system's Sender interface. When a sender visits *feedback.cs.pdx.edu* the majority of computation will happen on the Sender's machine. When the Sender successfully visits the webpage, the required resources will be loaded locally. Once that happens, the Sender has the option to visit a number of links located on the page, or enter their feedback. Once the Sender enters their feedback and submits, the webpage will send a POST request to the REST API running in the background - if it succeeds then a few things happen simultaneously:

1. The Sender's feedback is stored in the message database.
2. The feedback subject and message text are both loaded into some variables and the PSU Linux systems "mail" function is called and takes the Receiver's email, the subject and the message as parameters to the system call. Once complete, the email is sent to the Receiver located in the systems configuration file.
3. While step 2 is happening on the backend within the REST API, the users page is refreshed and the text previously entered is cleared.

However, if the REST API server is not up and running, step 2 will be skipped, and instead, the Sender UI will display a modal notifying the user that something went wrong in our system. The modal will inform the user to try again later and the page will not refresh as to persist the feedback to be copied and saved for later or for the Sender to click submit when they would like to try again.

3.3. Receiver Functionality

The Receiver functionality is quite light in this iteration of the system. What is currently being leveraged is simply the ability to send the Receiver emails with the feedback provided by the Sender(s). This functionality requires all actions to come from the Sender, and the Receiver will simply have a number of emails to read through to analyze provided feedback and act upon it on their own discretion. View the "Backlog" section to understand what has not been started or has been prototyped in regards to the Receivers functionality.

4. Process

4.1. Software Development Process: Agile Methodology

Firstly, we began by defining the project scope and its core requirements, then constructing a backlog of features according to Mark's requested project goals. Subsequently, we organized the sprints around the concept of a minimum viable product (MVP) and made estimations regarding the necessary time and resources. Throughout each sprint, we consistently developed and tested the system. At the conclusion of each sprint, we thoroughly reviewed the implemented features and presented that sprint's current progress and functionality to Mark. We then engaged in a reflective session to assess the system and identify areas for enhancement in future sprints. This iterative process was repeated in subsequent sprints, leveraging the progress achieved in previous iterations and continuously advancing the project.

4.2. Project Management Tool: Trello & Google Document

To efficiently organize and track the progress of each task, we initially made use of Trello, and we have transitioned to utilizing Google Documents. By employing Trello, team members can conveniently visualize the completed tasks and those that are yet to be accomplished, simply by moving cards between lists. In addition, we make use of Google Documents to record comprehensive meeting notes with the development team and sponsor on a weekly basis. Prior to each meeting, we outline the agenda to ensure that we have a reference to consult whenever necessary as well as time-limits to keep our meetings concise and to the point.

4.3. Work Allocation

Our team consists of members with similar technical skills and backgrounds, which allows us to distribute tasks fairly while considering each team member's current workload. It is our priority to ensure that every team member is assigned a workload that is fair, manageable, and equitable.

4.4. Copyright and open-source license

We have decided to make this project open-source and will include the MIT license in the repository. By doing so, future capstone project teams and other developers can continue developing the SAFE project. This will allow others to modify and enhance the project, as well as address any bugs or issues that may arise and enable legal protections by providing the software and system “as-is.”

Future teams will be able to sublicense the code and therefore further restrict how future iterations of the project are handled. This also means that teams or the university can eventually take the system into a closed source distributed license if so desired. However, this does not mean that the MIT license is void - all future iterations of the project must include the original MIT open-source license provided by this capstone team, even those that go proprietary/closed-source.

4.5. Potential risks for this project

Schedule risks may arise since part of our project relies on CAT deployment and OIT to integrate SSO, potentially leading to unexpected delays or inadequate project planning. We have established a contingency plan and backup solutions to mitigate this risk.

5. Schedule

5.1. Team Formation and Project Research - January 23rd - February 8th

- Check team member availability and schedule a regular team meeting
- Establish communication channels
- Research a few projects we were interested in
- Team got the SAFE project after Project Preference Presentation

5.2. Sprint 1 - February 9th to February 22nd

- Gather and review requirement specifications
- Identity MVP and planning
- Setup SAFE GitHub repository and project management tool (Trello)
- Design system architecture
- User stories
- Sender functionality UML

5.3. Sprint 2 - February 23rd to March 19th

- Receiver functionality UML
- Mock Sender and Receiver interface pages
- Discuss allocation of project implementation
- Project document
- Midterm presentation preparation

5.4. Midterm Presentation - March 20th

5.5. Spring Break - March 24th to April 2nd

5.6. Sprint 3 - April 3rd to April 23rd

- Minimal viable product implementation
- Created test cases and code review
- SAFE project configuration
- Received a MCECS account: feedback_web
- Received SAFE domain: https://feedback.cs.pdx.edu/
- Refactored SAFE system structure to a more lightweight workflow
- Branches Merged
- Sprint 4 planning

5.7. Demo of project progress to Mark - April 24th

5.8. Sprint 4 - April 24th to May 7th

- SAFE MVP Completed
- Contacted OIT and CAT to try to integrated SSO in the Receiver side
- Fixed minor issues according to sponsor's feedbacks
- Explored implementing reCAPTCHA to see if it is viable at this stage
- Sender user interface enhancement
- Prototype of Receiver dashboard user interface and back-end functions implementation

5.9. Mark first experience with SAFE - May 8th

5.10. Sprint 5 - May 8th to May14th

- Revised the remaining sprints/schedules to accommodate the deliverable items
- Fleshed out Receiver's dashboard UI prototype, and back-end functionalities (delete, reply message)
- MVP improvement, Sender UI error handling
- Integrated additional features in the sender UI and back-end routes

5.11. Sprint 6 - May 15th to May 21st

- Incorporated minor changes in MVP based on Mark's feedback.
- Cleaned up and documented code in MVP
- Deployed MVP to SAFE domain (feedback.cs.pdx.edu)

5.12. Sprint 7 - May 22nd to May 28th

- Cleaned up and documented code in development branch (features beyond MVP)

5.13. Sprint 8 - May 29th to June 9th

- Development process reflection
- Finalized on SAFE project document revision, SAFE technical document, SAFE credential document (for Mark only)
- Preparation for Final presentation

5.14. Final Presentation - June 9th

6. Backlog

6.1. For Future Development

6.1.1. Sender Requirements

- **Capable of forwarding a message to several recipients at once**
    - The implementation time is estimated within a week depending on the potential integration issues with existing systems, the system's complexity, and the development team's expertise
- **Mechanism to reduce spam from Senders**
    - Proof of Work (PoW) system (E.g. mCaptcha) used to rate-limit requests made on Sender side
        - reCaptcha v3 was starting to be developed but ran into technical issues that prevented full implementation due to not having an available webpage to work with.

6.1.2. Receiver Requirements

- **Dashboard**
    - Access granted through SSO authentication
        - The SSO setup is in place. Contact CAT to integrate it into the receiver dashboard when it's ready.
- **Multiple channels of communication**
    - Each of these is accessible through a centralized dashboard.
    - The dashboard provides a user-friendly interface that allows recipients to switch between channels and manage their feedback responses efficiently
    - Option to create new channels
    - Development of such a system can take several days to a few weeks or even longer as it is more complex than implementing a system with a single channel

- **Mechanisms for effective feedback management**
    - To enable efficient feedback management, certain features will be considered for implementation. These features may include:
        - Categorizing and tagging messages
        - Searching messages
        - Archiving messages
        - Exporting messages
    - Implementing each of these features would require several days for each feature to come online due to:
        - User interface design
        - Backend development
        - Testing and quality assurance

6.1.3. Admin Requirements

- Some considerable functionalities that the admin can perform:
    - Ability to add/remove Receivers
    - Secure authentication for the admin account
    - Additional Receiver account configuration and permission setting
    - Monitoring usage and activity within the system
- Implementing the admin requirements features may take several days to a few weeks depending on:
    - Specific detail of the implementation(level of customization required)
    - Potential integration with existing systems

6.1.4. System Requirements

- Capability to be adopted by other organizations with relatively light setup.
    - This particular backlog item is going to be a constant item to work on. There will need to be much more documentation required on how to set this up from the start to finish for any other organization to actively support this with ease.

6.2. Unreleased Backlog

6.2.1. Sender Requirements

- Unique code generated to access specific feedback and replies from the Receiver.
- Opportunity to receive replies
    - If reply is wanted, code is generated on user screen
        - Sender uses personal code to retrieve feedback and replies
            - Option to choose to receive code via email
    - One-time message if Sender does NOT want Receiver's reply

6.2.2. Receiver Requirements

- Mostly functional prototype completed but has not been tested or implemented
    - Receiver dashboard UI
    - Ability to view/interact with feedback submissions
    - Respond, delete

6.2.3. System Requirements

- Automatic sentiment analysis to tag incoming messages as positive, neutral, negative, etc.

6.3. Released Backlog

6.3.1. System Requirements

- Ability to handle high levels of feedback without experiencing performance issues or downtime
- Tool for automated system deployment on a new server

Appendix: Glossary of Terms

Backlog: Lists of major features that the team could/would like to develop for this project but we will not finish due to time or resource constraints.

CAPTCHA: A type of challenge-response test used in computing to determine whether the user is human.

CAT: Portland State University Computer Action Team. The CAT provides IT support throughout Portland State University Maseeh College of Engineering and Computer Science.

Computer Science (CS): The study of the principles and use of computers.

ESLint: Open source tool that helps find and fix problems with code in real-time.

Express: Framework used to design and build web applications quickly and efficiently.

Feedback: A message sent by Senders to the system for Receivers to eventually read and process.

JSON (JavaScript Object Notation): A lightweight data-interchangeable format that is easy for humans to write and for computers to read.

Material UI: An open-source React component library that implements Google's Material Design.

- **Google's Material Design:** A design system made by Google to assist teams in creating high-quality digital experiences for various mobile platforms and web applications.

MCECS: Maseeh College of Engineering and Computer Science.

MIT License: A permissive free software license.

Minimum Viable Product (MVP): A product with enough features to attract an early-adopter sponsor and validate a product idea early in the product development cycle.

Node.js: A platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications while providing a large number of libraries for ready-to-use situations.

Node Package Manager (npm): A package manager for the Node.js platform that allows developers to install and use various pre-built API.

OIT: Portland State University Office of Information Technology. OIT provides direction for the optimal integration of information technology in all PSU endeavors.

PM2: A process manager that can ensure an application is always running, even when the user isn't actively logged in.

Portland State University (PSU): The location in which the SAFE application is being developed.

POST request: Method to send information from a frontend interface to a backend environment or server.

PostgreSQL: A open-source object-relational database system used to quickly store and retrieve information on systems of scale.

Prettier: An opinionated code formatter.

Proof of Work (PoW): A form of cryptographic proof in which one party proves to others that a certain amount of a specific computational effort has been expended.

React: A declarative, efficient, and flexible JavaScript library for building user interfaces.

reCAPTCHA: A Google provided service to validate a user's actions on a form for a webpage.

Receiver: Person(s) who are authorized to read, reply, and manage feedback in the SAFE system.

Receiver Dashboard: The location in which the Receiver(s) will be able to engage with the system to read and/or reply to feedback and manage all feedback messages.

REST API: An interface that allows two computing systems to communicate and exchange information across a network safely and securely.

SAFE: System for Anonymous Feedback.

Sender: People who send an anonymous message to the Receiver.

Single Sign-On (SSO): Allows users to securely authenticate with multiple applications and websites by using just one set of credentials.

Source Code: Human readable text that is compiled into a program that can be executed.

Sprint: A sprint is a short, time-boxed period when a team works to complete a set amount of work.

SSO-CAS protocol: Central Authentication Service is a Single Sign-On protocol designed to allow potentially untrusted web applications to authenticate users against a trusted central server.

Trello: Trello is the visual tool that empowers your team to manage any type of project, workflow, or task tracking.

TypeScript: A strong typed programming language that builds on the already existing language JavaScript. TypeScript fully extends JavaScript and adds additional features to ensure type safety.

Unified Modeling Language (UML): A way to visually represent the architecture, design, and implementation of a complex software system.

Uniform Resource Locator (URL): The address of a web page.

User Interface (UI): The User Interface is the point of human-computer interaction and communication in a device.