SalesForce One

AC Draft App

Team Members:

Parker Egan, Curry Gardner, Michael DeVries, Kayden von Grosse

Client Name:

Matt Buland

Date:

June 18, 2021

Introduction

Our client for this project is Matt Buland, an internal developer for SalesForce. SalesForce is a multi-billion dollar company that specializes in software applications for the public and private sectors. As such, Matt proposed that user stories should be somewhat standardized regarding the wording of acceptance criteria. Streamlining the story creation process would help increase the efficiency of agile dev ops, as user stories are the beginning and basis of agile development.

The project goal is to create a web application that encourages users to create more thorough and informative user stories. The app encourages the user to write with 'harder' or more definitive language so that their stories are more concise and easier to understand. 'Harder' language describes the content of internet standard RFC 2119, which outlines what language such as 'should' and 'should not' should be used instead of 'could' to definitively state what a program must do and what must be done for the criteria to result in a feature rather than an idea.

Requirements

Minimum Viable Product

- Draft story saved to database with subject and description fields
- Includes language suggestions/language linting
- Open/edit existing story drafts from database
- Utilizes Lightning Web Components (LWC)
- Abides by Salesforce UI/UX standards

Delivery Method

Source code uploaded to client's github

Functional Requirements

- Language linter for real-time suggestions

- User management: different users can login and access their drafts or comments
- Merge similar story criteria together
- Peer review and commenting, users able to view, edit, delete their comments and resolve comments of others (akin to google docs)
- Integrates with SalesForce Agile Accelerator
- Customizable rules for creating stories

Non-Functional Requirements

- Is easy to use
- Lightweight
- Intuitive
- Promotes more specific user story creation with hard language
- Streamlines user story creation process
- Integrates visually and functionally with existing salesforce LWCs

System Architecture

Design Diagram(Figure 1):

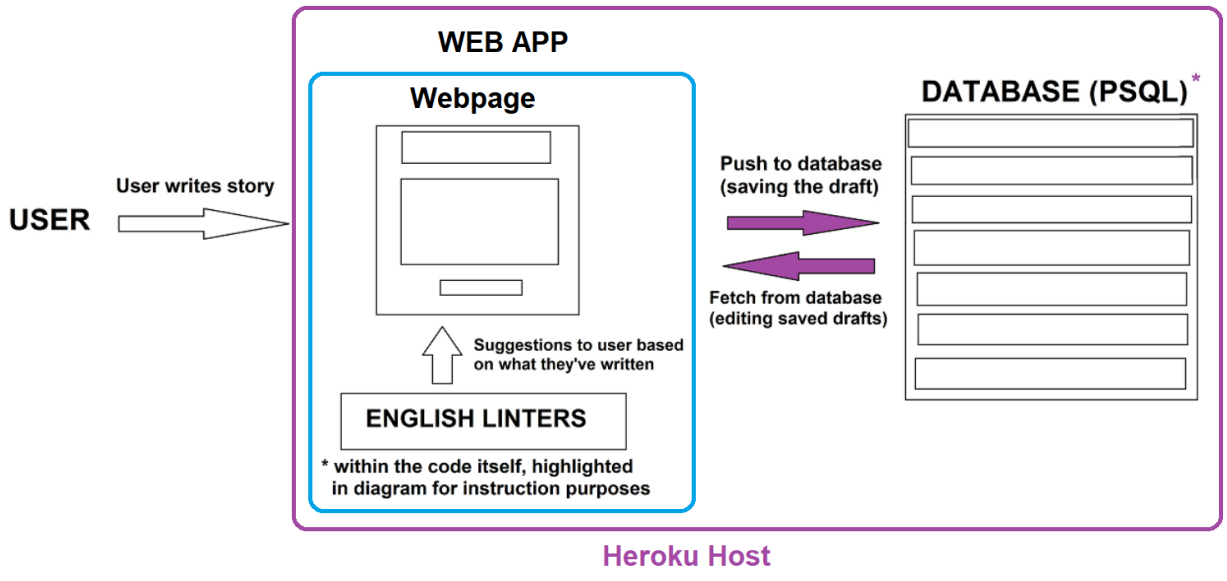

Figure 1

Design Descriptions:

Users

Users of our product are those who interact with the web application in order to write a user story, likely a SalesForce developer.

Web Application

The front end of the app consists of a basic login page followed by a menu page that allows the user to navigate the web application, from this navigation page the user may either access the new story interface or the edit story interface. The new story interface allows the user to type a new subject and story description with their own rules. Additionally, the user will receive recommendations throughout their writing on how they can make their story more thorough.

The back end handles html updates for the front end page such as communication between components and navigation between the different pages The 'Language Linter' checks contents of text field and applies filters for soft language. An API server is started to show the app's html content and for connection to the Postgres database so that the web application can remotely save, edit, and delete user stories.

Technical Design

By far the most interesting and unique aspect of this project is the English linter. The English linter takes in user input word by word as the user types into the text box and in real time recommends replacement words that would make the user story more official and easier to understand. The words that are replaced are soft words such as might, could, ought to, etc. and they are changed into more definitive words such as must, should and may (Table 1). This forces the user to use a standard language throughout all user stories that is simple and concise. Additionally, the linter also checks for user acceptance criteria, which is the basis for how the web application establishes customizable writing rules.

Table 1: English Linter Recommendations

User inputs	Recommended changes
Might	Consider rewording into 'must' or 'may'
Try	Consider rewording into 'do'
Ought to	Consider rewording into ‘should’
Could	Consider rewording into ‘should
Possible	More definitive word choice
Possibly	More definitive word choice
Required	Consider rewording into ‘should’ or ‘must’

In figure 2, there is an example of a poorly written user story. In the orange notifications, there are a few problems that exist. One problem a few times throughout the user story is that soft language is used where definitive language should be used. In addition, there are no listed acceptance criteria. In Figure 3, there is a better example for a good user story with strong language and acceptance criteria. Because of the english linter implemented, the user was able to write a stronger, more understandable user story for whoever is going to be working on it.

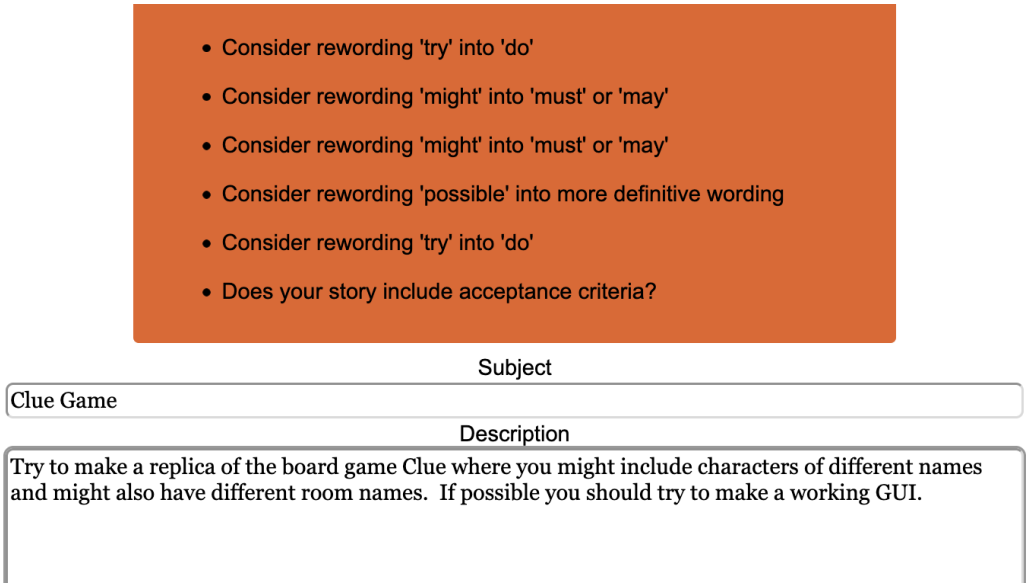

Figure 2: Bad User Story

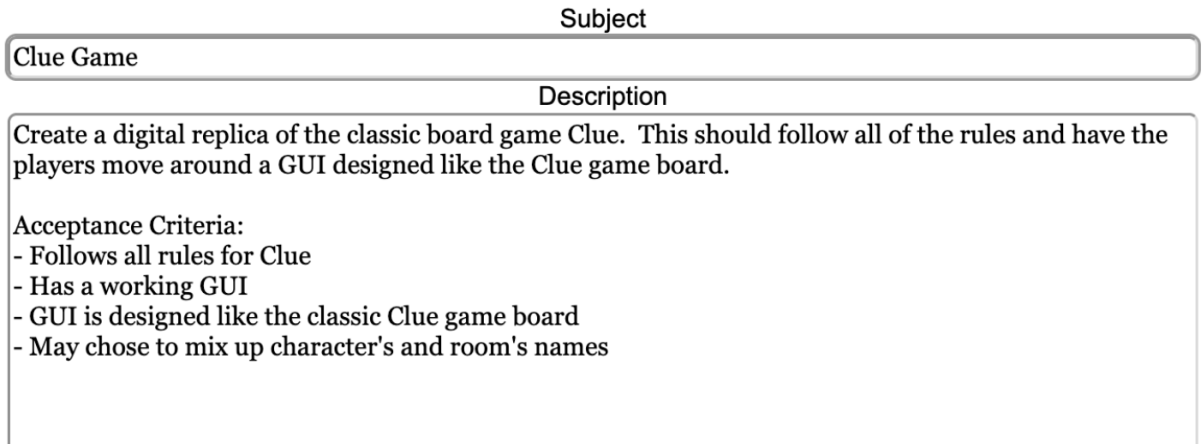

Figure 3: Good User Story

Each front-end element is a separate html file with an associated javascript file to control its functionality. Each set of these files is considered a component, which can be easily included in a parent component or app. These components are modular, and use SalesForce proprietary architecture called Lightning Web Components. While all of our components are custom built, their framework is the Lightning system.

As an additional part of our technical design, we also have included and implemented a PSQL database into the back end of our system. The database is hosted and maintained by Heroku, a cloud platform owned by SalesForce. The database works with our product by allowing users to save the drafts of their stories and then pull them from the database at a later time, whether that be in days, months, or even years, and edit it if they think of improvements or additional content for their story.

The database is accessed through an API server written in javascript. When the app is launched, this API server starts and connects to the database. There are a number of endpoints that allow users to interact with the app to push, pull, and delete from the database, using INSERT, SELECT, and DELETE queries.

When it comes to the creation and maintenance of the schema that we use, we had a simple design with a single table titled drafts. The creation of this table itself is not built into the code and is instead done manually through a PSQL shell. This was done in this way as our application does not alter the database itself but instead pushes and fetches data to and from the database.

Quality Assurance

The team's quality assurance plan is derived from good agile development practices. These practices include pair programming, daily scrum meetings, client input meetings and test-driven development.

Pair Programming:

The team has been split into two subgroups. Parker and Curry are the first subgroup and Michael and Kayden the second subgroup. In the groups, different elements of the project are developed simultaneously with constant peer reviews during development. The first group works on and develops the front end, including but not limited to, organization on the web application using HTML, Styling using CSS, and handling logic and inputs with JavaScript. The second group works on and develops the backend, including, but not limited to, PSQL database management using Heroku, and linking it to the web application using JavaScript.

Daily Scrum Meetings:

Every weekday at 10am, the team got together, in-person and over zoom/discord and discussed what was achieved since the previous scrum and what the plan is for the day. This worked well for the team and made sure that progress was being made at a decent and expected rate. After the short scrum meeting was complete, the discussion moved to talking about any difficulties or resources the team found to aid in each other's efforts. Since this project was a learning experience, we aimed to pool resources and time in order to maximize learning and working efficiency.

Client Input Meetings:

The team meets with the client every Monday, Wednesday and Friday. At the client meetings, the team then presents on what work was done since the previous client meeting. This includes a live demo with the client and a walkthrough of the coding done. The client would then give input for

amendments to the design or functionality along with any new features. This allowed us to pivot quickly and

Test-Driven Development:

For test-driven development the team used a variety of different testing methods to test the code. The different testing methods are as follows:

Unit Testing:

The unit testing for this project was very limited. The nature project did not warrant many unit tests. One major test that was made and used is testing pushing and pulling to the PSQL database. This can be done by manually sending an item to the database and then retrieving the item and checking to see if the items are identical.

User Testing:

User testing was the main source of testing done for this project. Since the project is heavily dependent on user inputs that vary based on what the user wants. This consists of tests such as inputting poorly written user stories into the description box along with copying and pasting poorly written user stories into the text box and in both cases, checking if all of the errors show up. A different test included saving stories and reloading them into the text box.

End-to-End Testing:

This testing was another frequently used test, since the project deals directly with front-end interfacing to back-end with a database. This was done by writing a new story in the front end, HTML code and saving it to the database. This test can be concluded by checking the data in the database to see if the story was saved. Similarly, the test can also be done by pulling a story from the database in the edit menu.

Results

We, as a team, are satisfied with the quality and completeness of our project. We achieved the minimum viable product and have created a framework

for testing the ideas of standardized story creation with acceptance criteria and future expansion.

Unimplemented Features:

- User management and SalesForce integration
- Merging two similar stories
- Peer review and in line commenting
- Creating a unit test from user stories

Summary of Testing:

Although Unit testing was not practical for the project, end-to-end testing works very well and thoroughly encapsulates any errors we may come across. Throughout the build process and testing the team has come across a multitude of errors that we have been able to remedy because of end-to-end testing. A lot of errors we encountered were design errors. For example, after we had the program working end to end we ran through it and realized that once the user had made a selection on what page the user wanted to travel to, the user could not actually get back to the navigation page, leading to the creation of the cancel buttons.

Lessons Learned:

- Set realistic goals for each day of work via scrums or other methods
- If the team gets stuck on a problem for an unreasonable amount of time, either get help from someone that is well informed in the area or move on to a different issue and come back to it later
- Communication is key, constantly communicate with the client and the team members to ensure a quality product

Future Work

This project was built to replicate lightning web components, meaning that every single thing that can be seen throughout the web app is a series of self-enclosed components. This allows for extremely quick and simple component

swapping throughout the application so future groups will have little to no difficulty when replacing components.

Furthermore, the team left a large variety of comments throughout the entirety of the project to ensure that future teams could clearly understand what is happening in the code.

This project has a lot of potential for new features, a specific example would be the ability to have peer review sessions with live commenting and editing similar to google docs. Another good feature to implement would be higher level user management, the current user management system is just a framework for future development that can be expanded upon very easily. Finally an interesting feature implementation would be to be able to create basic unit test from detailed enough user story, this would be difficult to implement in concept however the actual integration with the current web application would be quite simple as all the future team would have to do is call a pull request from the database for the user story that they would like to get unit tests for.

Appendix

Dependencies:

For the web application to load properly both lightning-base-components and the Salesforce Design System are required.

Dependency Installation:

- npm install lighting-base-components
- npm install @salesforce-ux/design-system @salesforce/design-system-react