# CSCI 370 Final Report

# Pie Insurance 3:
# Operational Data Management Application

Michelle Lieng, Nikola Jaksic, Kurtis Quant, Aidan Reaves

for

Gregory Seidman, Josh Seder, Ryan Countryman

June 14, 2022

CSCI 370 Summer 2022
Prof. Kathleen Kelly

# Table of Contents

# 1: Introduction:

Pie Insurance provides workers' compensation to small businesses. In order to accomplish this task, Pie Insurance must store data that is updated by business associates frequently. Currently, an ineffective process requiring database experience and the manual validation of a mix of spreadsheets and CSV files is used to upload data. The current process is both slow and prone to human error, which wastes the company's time and money.

To address the problem of manual data updates, our team was tasked with creating an operational data management application that allows business associates who have limited engineering experience to update data. The Operational Data Management Application (ODM App) is a web application that allows users to upload a CSV file to a GitHub database without any engineering experience. The application takes in CSV files from the user and ensures that the data uploaded is correctly formatted. If the CSV files are correctly formatted, they are uploaded to GitHub. Otherwise, the application displays a detailed list of errors found in the CSV files. By validating the data and interacting with GitHub, the ODM App maintains data accuracy and simplifies the process of updating datasets that are vital to the operations of Pie Insurance.

# 2: Requirements:

## 2.1: Functional Requirements

- Web interface
    - A front-end web interface is needed for data intake and display of the results of data validation.
- Input form that allows the user to choose a dataset to update, upload a CSV file with the updated data, and enter the Jira issue key associated with the update
    - The engineers at Pie Insurance use Jira, an issue/project tracking software application, to keep track of data updates. Each data update is associated with an issue identified by a key in Jira. Jira is integrated with GitHub and can detect GitHub branches that are created for a particular issue. The Jira issue key is needed to associate a new branch with a Jira issue.
- Data validation
    - Once the data is uploaded, the application should validate the data against the schema corresponding to the selected dataset. Validation consists of checking for correct column names and ensuring each column meets constraints such as row uniqueness.
- Indicator of successful or failed upload based on whether the data passed the validation
    - If the data passed the validation, the application should programmatically create a branch in the GitHub repository that stores the data and add the file to the newly created branch. Then, the application should programmatically create a pull request for the branch.
    - If the data failed to pass the validation, the application should display a list of errors found in the uploaded data.

### 2.2: Non-Functional Requirements

- Clean user interface
    - The web application should have a simple structure that is easy to navigate through. Users should be able to understand what to do on the web application without any help.
- Easy to use
    - A novice user should be able to navigate and use the website as intended on their first day.
- Assume a non-perfect user
    - Make sure the user can click or upload anything and the website will not crash.

### 2.3: Definition of Done:

The client wants a proof of concept that demonstrates the ability for users to input files into a system that ensures appropriate data. In this proof of concept, being able to interact with GitHub to create branches and upload validated CSV files to the branch were the most important objectives to complete. The users should be able to input CSV files into the website and receive a successfully validated data message, or a detailed rejection message.

## 3: System Architecture:

The ODM web application has three main components: the front end, the back end, and the GitHub API. These three components are connected by Microsoft's .NET Core framework, which uses HTML, CSS, and Javascript for the front end and C# for the back end. See Figure 1 below.

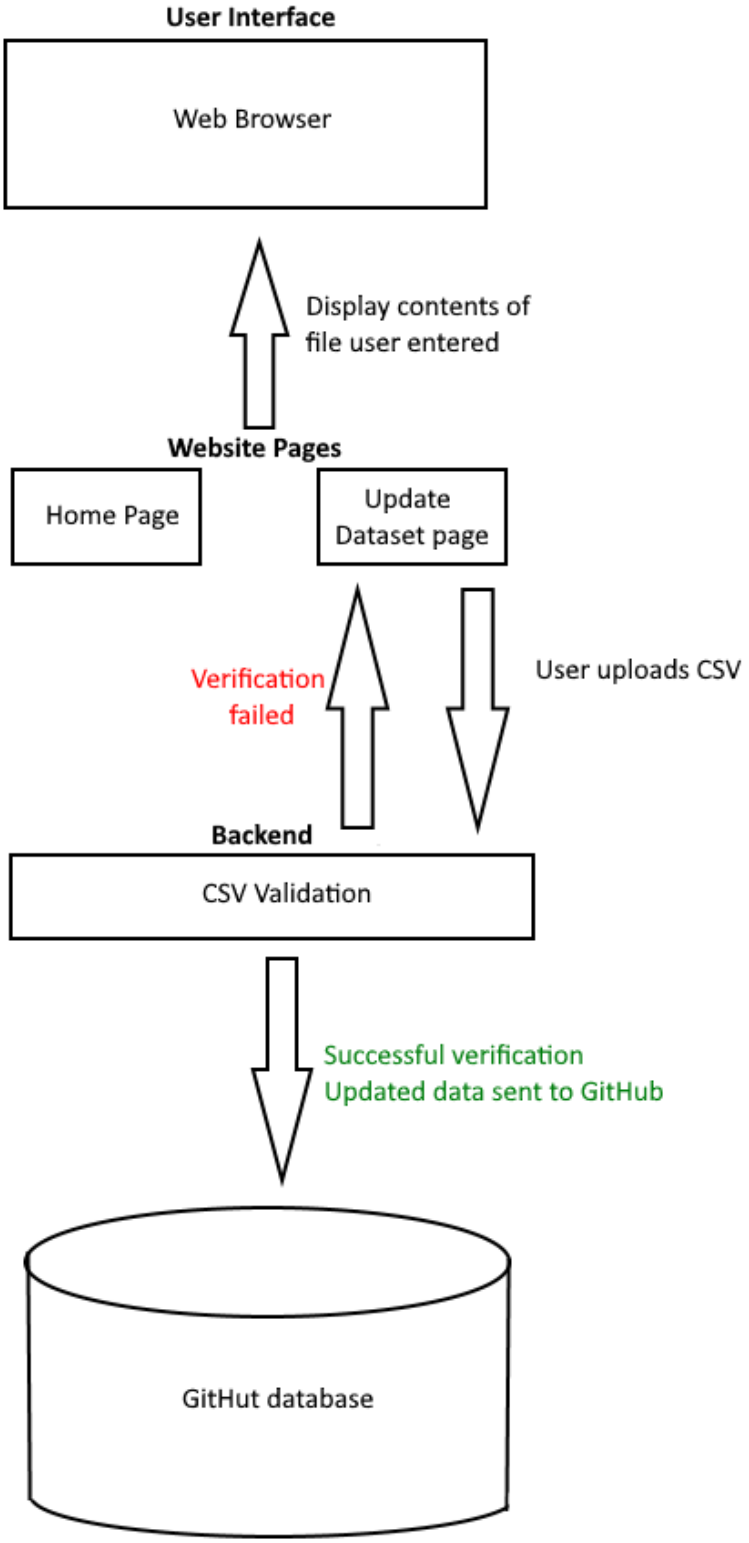



Figure 1. Architecture of the Project

**Front End:**
The Update Dataset page is the primary page of the web application. Upon accessing the Update Dataset page, the user is prompted to select a dataset from a dropdown menu to update. When the user selects a dataset, the file path of the dataset in the GitHub repository is stored. The user is also prompted to upload a CSV file containing the updated data and asked to enter the key of the Jira issue associated with the update. Then, the data from the CSV file is transferred to the back end for validation. If the data passes the validation, a message indicating a successful upload will be displayed. Otherwise, an error message and a list of errors found in the uploaded file will be displayed to inform the user of what to fix.

**Backend:**
The back end of the program is written in C# and utilizes .NET Core 6.0. The file path of the selected dataset and the data from the CSV file is passed from the front end to the back end, where the CSV file is processed. After checking the data in the CSV against the appropriate schema, the back end returns a list of errors to the front end to display. The back end also triggers the GitHub interaction if the given CSV file has no errors. The back end alone is responsible for checking and validating files to ensure that they are correct.

**GitHub Interaction:**
In order to add files to Pie Insurance's GitHub repository, the web application uses the GitHub REST API. The application uses Octokit.net, a client library for .NET, to interact with the GitHub API. Octokit is used to create a branch from the repository's main branch. The new branch is named using the Jira key entered by the user and the name of the selected dataset. Octokit is also used to create CSV files using the data uploaded by the user, add the files to the new branch, and create a pull request for the branch.

# 4: Technical Design:

**Data Validation:**
Figure 2 shows the flow of the application, starting from the CSV upload. After the user uploads a CSV file to the web application, the data from the CSV file is stored in a class that holds the data from the user. The schema corresponding to the dataset selected from the user is then retrieved from a hardcoded dataset. During data validation, each column of the CSV file is checked with the constraints of the schema. If the data follows the schema, the CSV file is uploaded to the GitHub repository. Otherwise, a list of errors is returned to the front end and displayed to the user.

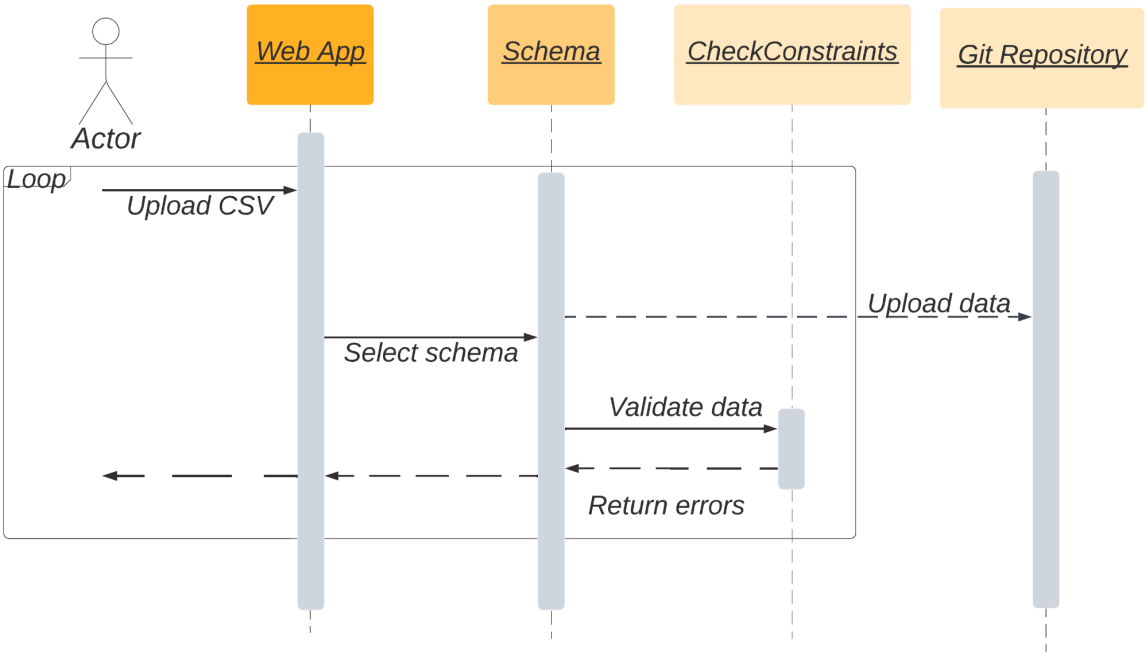



Figure 2: Sequence diagram

**User Interface:**
Figure 3 shows the wireframes of the main page of the website. Figure 4 is a screenshot of the website as it currently stands.

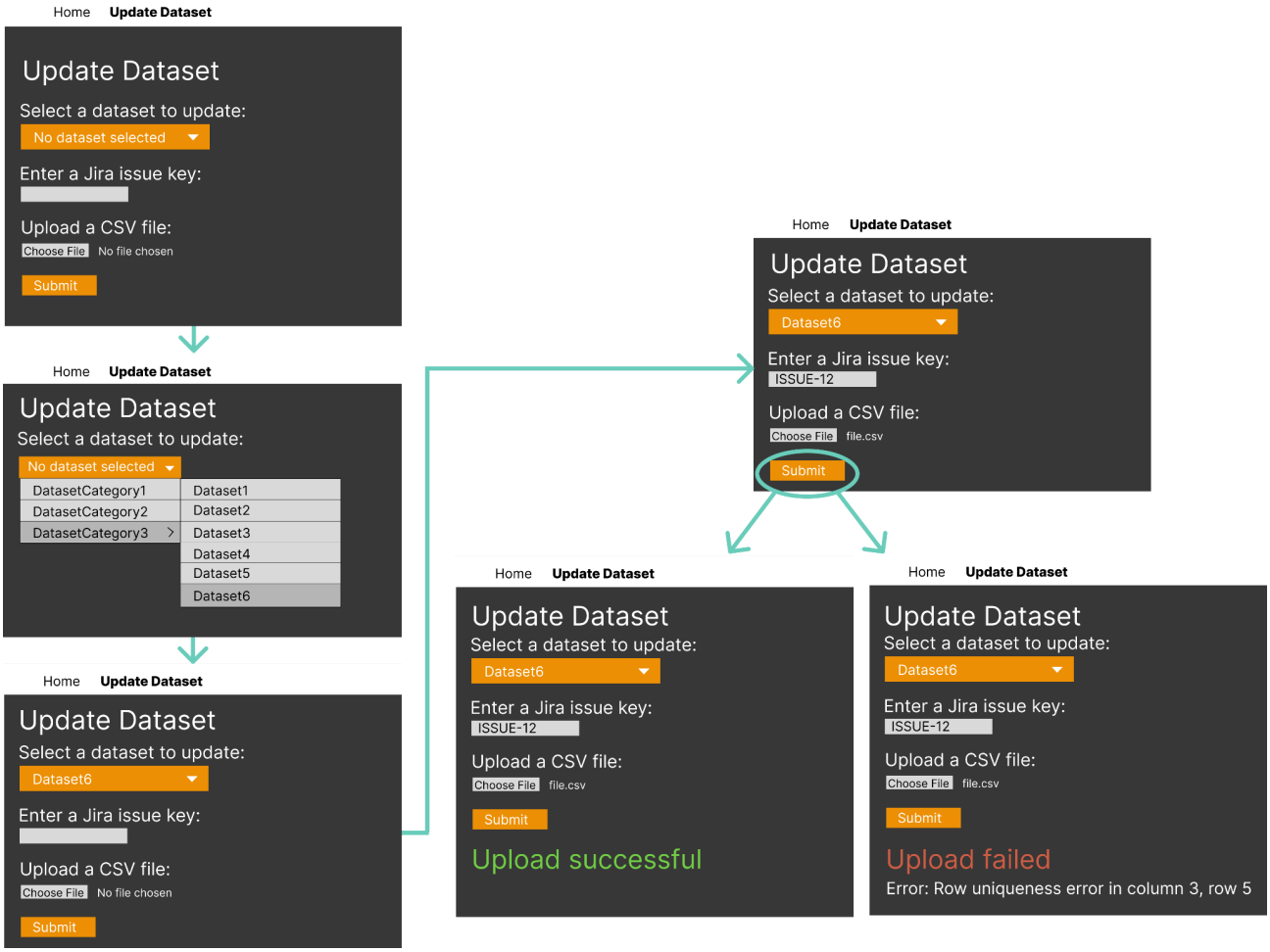



Figure 3: Wireframe of the Update Dataset page of the website

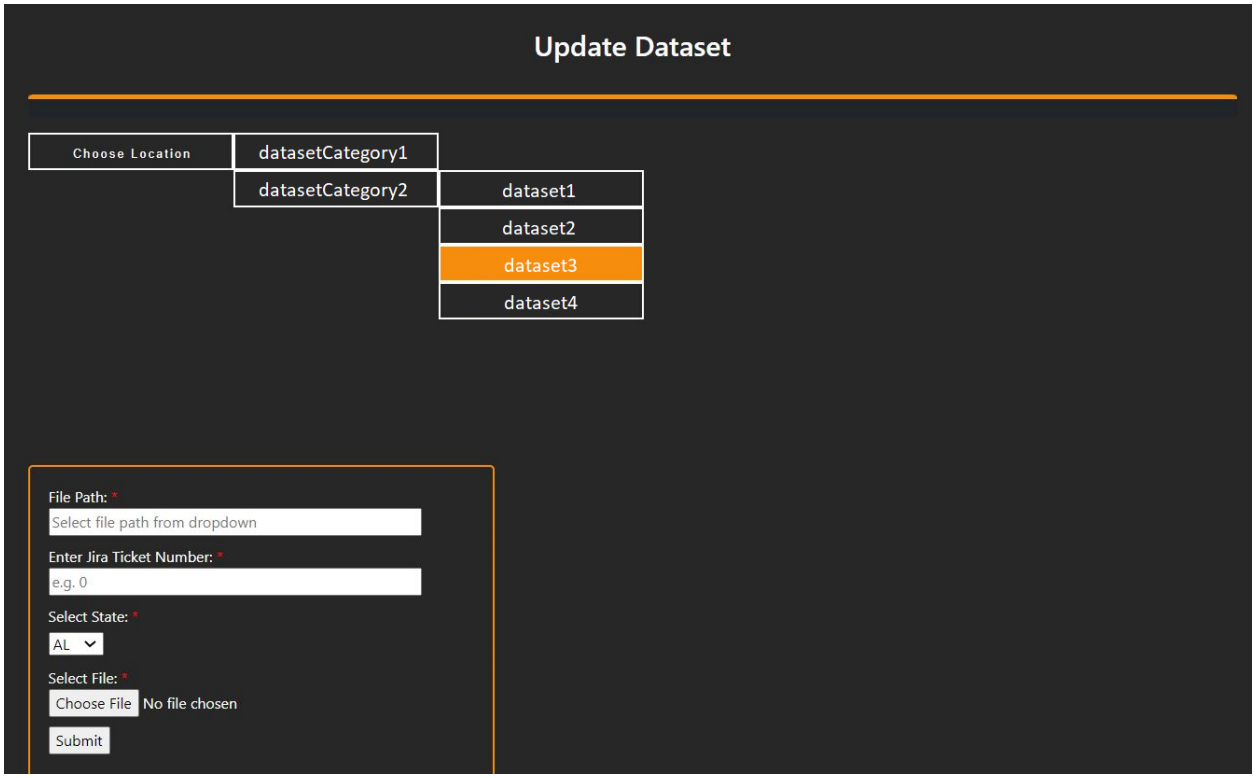



Figure 4: Screenshot of the website
(Specific data censored for privacy reasons)

# 5: Quality Assurance:

## 5.1: Software Test and Quality

- Unit testing
    - To ensure that the CSV data is properly validated, we used unit testing on the methods that check for constraints. For the unit tests, we used xUnit, a unit testing tool for the .NET framework. We wrote two sets of unit tests. The first set of unit tests was for testing the methods that take a column of values as input, check that the values meet a certain constraint, and return a list of errors in the column. The second set of unit tests was for the method that accepts a data structure containing all of the values from a CSV file and a schema to check the data against. We tested that the method produced the correct output (i.e. whether the data passed or failed the check against the schema) when given data. Unit testing helped us ensure that the methods properly handle all cases, including edge cases.
- User interface testing (automated or otherwise)
    - The user interface is not the most important part of the project; our client has told us to produce a proof-of-concept for the UI. That being said, we still needed a website that can accept data, transfer the data to the back end, and display the

results of the validation. In this aspect, we began testing the web application manually to make sure that it functions properly. We also ensured early on that the GitHub REST API works as expected in the back end when doing user interface testing.

- Integration testing
    - Since our project is small enough, we rarely had to do integration testing. However, since we split up the front end and back end, we had to do some integration testing to make sure the front and back worked together appropriately. Every time a large change was made in either the front end or back end, we manually tested it through the front end to make sure that nothing broke. This manual testing typically came in the form of uploading data to the front end to see if it could still be processed successfully in the backend without the program crashing. In this regard, unit tests were not often helpful since the most common issue would be a compiler error from the front end being unable to communicate with the back end.
- Code reviews
    - Having code reviews is an important aspect of the Agile process and software development as a whole. We had regular code reviews with our team to make sure no one was left in the dark about our progress. Our client regularly reviewed our code and requested changes that needed to be made in order to merge new code with the main branch of our project. Code reviews were also important in keeping coding standards high.
- Code metrics
    - Writing our code with good software engineering practices was important to the success of the project. This includes writing comments, breaking up code into smaller methods, etc. Following principles like YAGNI (You Ain't Gonna Need It) and DRY (Don't Repeat Yourself) helped us improve the overall quality of our code so that it flows better and can be easily understood. When creating a pull request through GitHub, CodeQL generates a quality report on our code. This quality report was helpful for identifying areas for improvement in our code.
- Static or dynamic program analysis
    - We were able to accomplish a degree of static program analysis since we were frequently pair programming and reviewing code. This allowed us to analyze how our program should run while continuing to accomplish work. We also did dynamic program analysis when we finished updating or coding a large chunk of the program and wanted to ensure that it follows the logic that we laid out beforehand.

## 5.2: Ethical Considerations

- **ACM/IEEE Principles that are pertinent to the development of our product:**
    - 2.05. Keep private any confidential information gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law.

        - It is important that private company information is not released to the public because vulnerable information is stored in the Pie Insurance database. The data that we work with is private data between customers of Pie Insurance and Pie themselves, so we must keep this private and not violate this principle. This is being addressed by simply keeping the data withheld.
    - 3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.
        - Every project should have achievable goals. Throughout this project, we planned out each sprint which helped us stay organized and efficient. Although we struggled to estimate task difficulty, we knew the overall project was achievable. The way that we conveyed this ethical principle was by having standups and communicating if it was too much work for the week.

- **ACM/IEEE Principles that are most in danger of being violated:**
    - 2.01. Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education
        - This section is important for all of us in our project. Although nothing serious has occurred, at times, team members seemed to be uncomfortable explaining their lack of experience with certain problems.
    - 3.03. Identify, define and address ethical, economic, cultural, legal, and environmental issues related to work projects.
        - Although we have discussed ethical issues surrounding our project (such as leaking private data), we have not gone into a lot of depth about these issues. This does not violate this principle, but it brings up something that the team should discuss more.

- **Michael Davis Tests that apply to our project:**
    - Legality test: Would this choice violate a law or a policy of my employer?
        - In the beginning of the semester, we signed an NDA that had a section that spoke about data confidentiality. If the data is not kept private, we could violate our NDA and Pie Insurance policies regarding the data confidentiality, which would fail this test. However, we are successful in passing this test since we have kept all of the data private.
    - Common practice test: What if everyone behaved this way?
        - Although most of our code is rather clean, there are some aspects that are a bit messy. If everyone coded that way, it would surely cause problems with understanding each other's code. For example, there are some parts of our projects that are hardcoded in. If everyone relied on hardcoding, there would be a lot more issues with software today. However, our client has expected that some of our code will be hardcoded due to the time constraint and our lack of experience. If everybody behaved this way, then

software would be less respected, but since our client gave us the green light, it is fine for this situation.

- **Ethical considerations for the project if the software quality plan is not implemented properly:**
    - Our client would not be able to learn from or use our work if they wanted to reference it in the future. Since we are making this as a proof of concept for them, it would be unethical to make code that does not fulfill its purpose.
    - Also, if we do not follow our software quality plan, we could submit code that has numerous bugs that can easily be detected with appropriate software testing. This is unethical as the quality of the code would be decreased unnecessarily and would negatively affect anyone trying to use our code as a model.
    - Our software would be essentially unusable. If implemented poorly, it would be easier for Pie Insurance to work from the ground up (wasting more time) than expanding on what we have worked towards. It would be unethical to waste their time like that.

# 6: Results:

The goal of this project was to make a web application that can take in a CSV file, validate it against a schema, and add the file to a Git branch in a repository if the data is valid or display errors if the data is invalid. In addition to implementing these functionalities, our web application displays the uploaded data in a table and highlights the cells to indicate which values in the CSV file do not meet the constraints of the schema.

**Testing Summary:**
We used two sets of unit tests: one for the methods that check that an input column meets a certain constraint and another for a method that checks that a collection of columns meets the constraints of a schema. Some errors in the code were found through testing. Through creating tests for the schemas to pass, we found that sometimes the schemas would have incorrect information, such as allowing a type of data to be null when it should always have a value. After creating a massive amount of tests, all of these errors in the schemas have been fixed, and the program now passes all of the unit tests.

In addition to unit tests, we did user interface testing. We manually tested that each button in the drop-down menu works and the program requires the user to enter inputs into all fields in order to upload a CSV file for validation. We also uploaded a mix of valid and invalid CSV files to the web application and checked that the web application showed either a successful or failed upload message and all of the errors if there were any.

In conclusion, since our product meets the goals, has gone through adequate testing, and passes all the functional and non-functional requirements, we have met our definition of done.

# 7: Future Work:

There are a lot of ways that this product could be improved upon in the future. However, there are two major features that we were not able to complete in this project. One main feature that could be implemented is linking our project up to an application called Jira using the Jira REST API in order for the process of checking CSV files to be more organized. Rather than entering the Jira issue key, the application could display a list of Jira tickets retrieved from the Jira API for the user to select from. As Pie Insurance becomes a larger company, they will have to upload or update files more often, and keeping track of which files have already been updated will become more difficult.

Another way that this project could improve in the future is by adding a better front end to the web application. Since this project was mainly a proof of concept, we were told not to worry about the front end look and mainly focus on the back end of the project. The application could be more intuitive to use for new users as a lot of the inputs only give vague information on what to do. Data could also be saved in order to predict the user's next input which would save a lot of time for the user.

In addition, the application could check if the file the user uploaded is intended for a dataset different from the one that the user selected. It is inevitable that a user will attempt to upload data in the wrong spot. Depending on how much data is incorrect, displaying something that tells the user what dataset they seem to be trying to upload to could be helpful in keeping user-interaction time low.

# 8: Lessons Learned:

- There are very useful resources for communicating with GitHub. The original idea was that it would be a nightmare to communicate with GitHub to access data for the client. However, we found the toolkit Octokit that let us access the client's GitHub data in a simple manner.

- We learned about the Model-View-Controller design pattern for web applications. The APS.Net Core MVC framework that was used in our application uses CSHTML files, which are powerful for creating websites that deal with back-end calculations. With CSHTML, it is easy to take in data via a website and manipulate it in some way.

- The Scrum process works effectively. Using Scrum every day has helped with team communication, fixing blockers, and resolving any issues that came up. The ability to meet every day and bring issues to a short meeting gives valuable insight into what goals have been met and what needs to change. It helped us understand what our client wanted from us, and made it easy to adapt to any changes.

- Demonstrations of a project for the client is important. In one of the first weeks, the project was shown to our clients and they gave us helpful feedback. However, we did not show another demo until two weeks later. This meant that we had less time to fix any issues that our client would have with our code. During a demonstration, we learned that

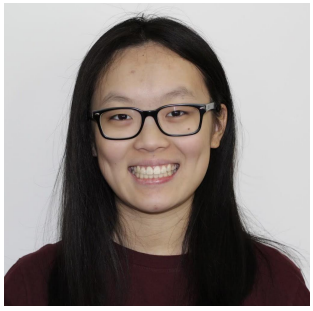
we were approaching problems differently than our client expected and were able to schedule a meeting with our client to refactor our code.

- Refactoring is crucial to code quality. Refactoring can involve deleting a significant amount of code, but it will help in the long run by making the code easier to understand and extend.

- One of the overarching themes of this was how ethics are involved with software engineering. In every project, there are layers of ethics involved. In this one, we understood the importance of things such as keeping data secure since you don't want people's private information being leaked. Furthermore, we learned how transparency is the key to a successful project and why it is unethical to be secretive.

# 9: Team Profile:

- Nikola Jaksic
    - Sophomore
    - Computer Science
    - Pueblo, CO
    - Hobbies: Music, biking, video games



- Michelle Lieng
    - Sophomore
    - Computer Science

    - Lafayette, CO
    - Hobbies: Piano, music

- Kurtis Quant
    - Sophomore
    - Computer Science
    - Danville, CA
    - Hobbies: Music, swimming, computer graphics, video games

- Aidan Reaves
    - Sophomore
    - Computer Science
    - Greeley, CO
    - Hobbies: Biking, karaoke, trivia, video games

# Appendices:

Appendix A – Key Terms

| Term | Definition |
|---|---|
| Epic | A series of user stories that share broader objective |
| Blocker | An obstacle that you cannot pass - call this out at stand up |
| Schema | Describes the columns that exist and the type of data expected |
| REST API | The protocol used to communicate with the GitHub servers |
| Octokit | The toolkit used to easily interact with the GitHub servers |
| Jira | Specifically for our project, the site used to hold updated data requests for approval |
| CSHTML | A file format that integrates C# with HTML |