Final Report

Project - Child Care Enrollment Management System

About

Organization/Stakeholder - Child Care Group (https://childcaregroup.org/)
Client - Yan Li (yli@ccgroup.org)
Links -
GitHub Repository - CCEEMS
Link for Deployed App - Heroku
Project Management Tool
Slack channel

Presentation and Demo

https://youtu.be/q7DSW_PrOvY

Summary

The project was to build a child care enrollment tracking system for a non-profit organization, “Child Care Group,” based out of Dallas, TX. Currently, there are over 10,000 children enrolled in different programs in the organization. The users of the system will only be the internal staff of CCGroup, such as Eligibility Managers, Supervisors, Specialists, and the senior leadership team. The goal of the system is to allow the staff to keep track of existing children's enrollment requests and work on new ones as well.

As of today, all client requirements have been met successfully. Two of the most important requirements were - Analytics of current/past enrollments and requests and a Case management feature where the staff can update existing requests and work on new requests assigned to them. We have implemented the completed end-to-end flow of the system - Login, Analytics on the Homepage, Upload and Assignment of New Cases, Case management, and Logout. Additionally, there are multiple roles/categories of users so the application won’t display the modules for which the users are restricted. Technologies used are Python, Flask, HTML/CSS, Javascript (JQuery), PostgreSQL, Git, and Heroku.

Lastly, the client has suggested a few minor UI changes which we will continue to work on.

Team Members

Atharva Phand
Bharath Kumar Ravichandran

Chidambaram Ganesan
Daehee Han
Noah Pang
Tanmai Harish

Iteration wise Roles

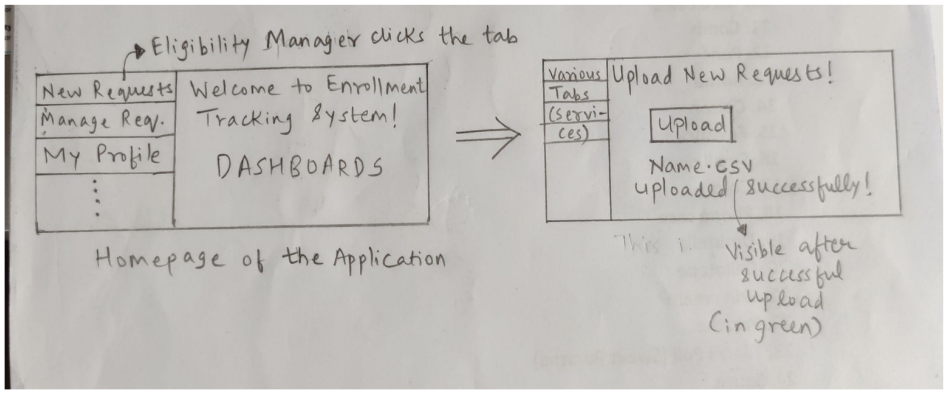
We had decided that we will try to take up the roles in round-robin fashion. That is why we had role changes every iteration.

	Iteration 0	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
Scrum Master	Bharath Kumar Ravichandran	Bharath Kumar Ravichandran	Chidambaram Ganesan	Tanmai Harish	Atharva Phand	Noah Pang
Product Owner	Atharva Phand	Atharva Phand	Bharath Kumar Ravichandran	Daehee Han	Tanmai Harish	Tanmai Harish

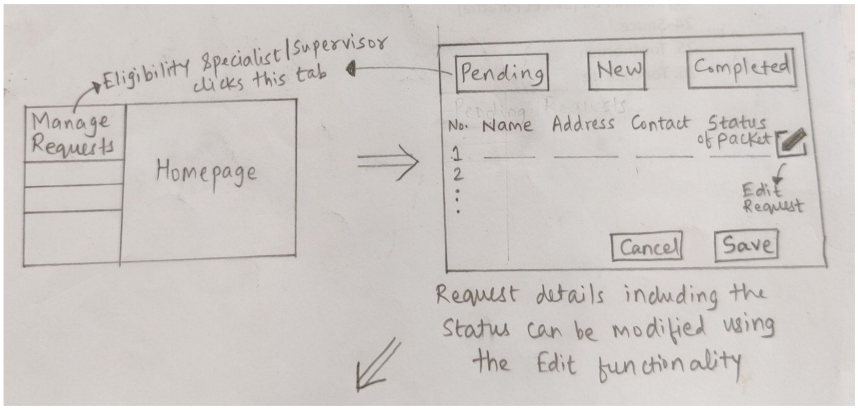
Description of User Stories and LoFi Mockups

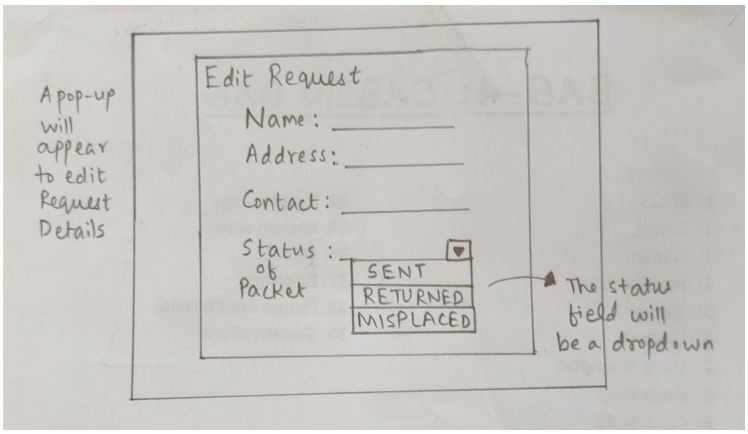
Iteration 0 mockups

New cases feature



Edit Application Packet Status UI mockup





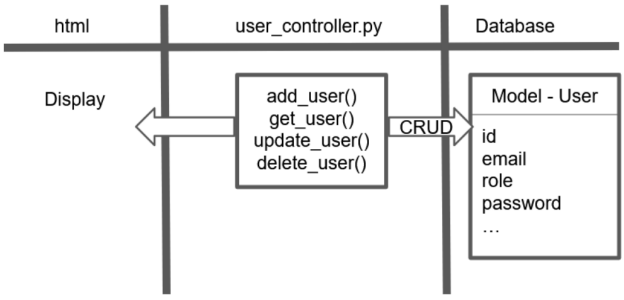
Iteration 1

ID	Story (Name & Description)	Story Points
1	Add controller to upload New Requests (CSV) and seed data in DB Created feature to upload new cases(requests) to the database. We added extra steps like validation later, but the main functionality is still the same.	2
2	Test cases and Diagram for New Requests Controller Create test cases to test upload new cases(requests) and design a diagram for how the new cases are handled in the program.	2
3	Add database models, migrations and seed data Database tables and data creation.	2
4	Controllers for manage users page	2

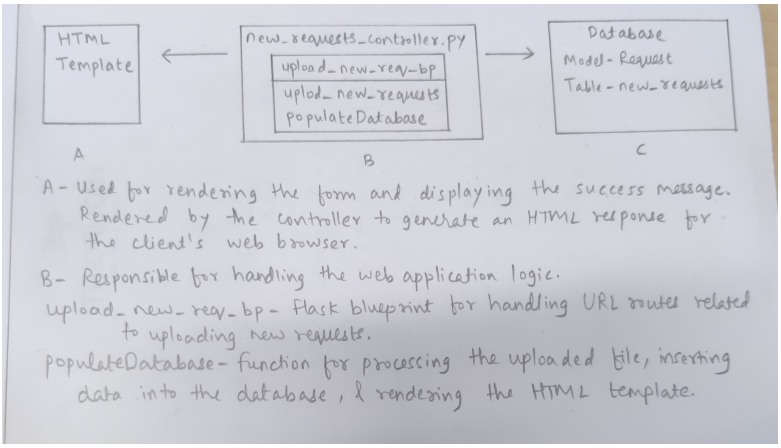
	CRUD for user information. Users are employees of our client who will be able to edit cases using our app.	
5	Add login page UI and controller Initial login feature implementation - UI, Password encryption.	**3**
6	Diagram for Login Controller	**1**
7	Design user controller flow and diagram	**2**
8	Design database schema	**2**
9	Add UI for manage users page Initial screens for add, manage and delete users.	**3**
10	Create UML diagram for Users controller	**1**
11	Setup DB and deploy application to heroku Setup Postgres installation, create a user, and database and grant privileges to the user. Setup application on heroku, create profile and deploy the app on heroku	**2**
12	Behave tests implementation and code coverage calculation Setup Postgres installation, create a user, and database and grant privileges to the user. Setup application on heroku, create profile and deploy the app on heroku.	**2**

Iteration 1 diagrams

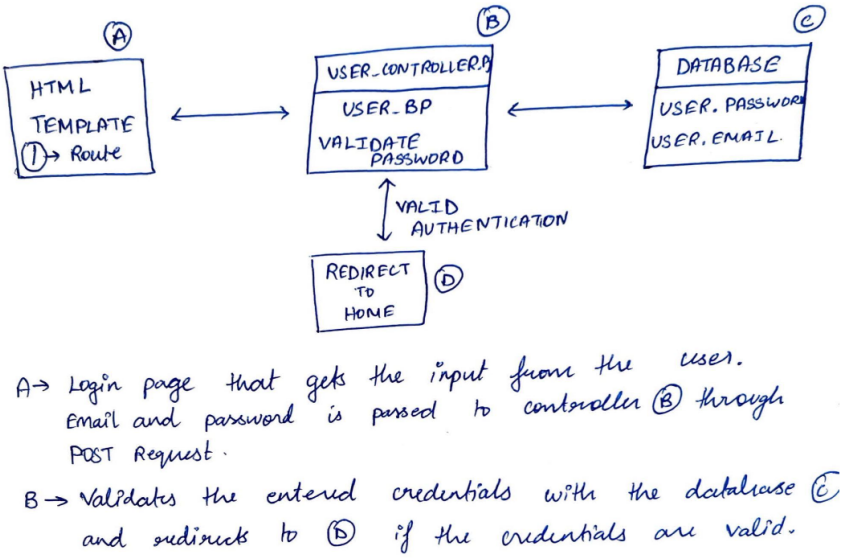
Users controller -



Upload new data to the database -



Login feature



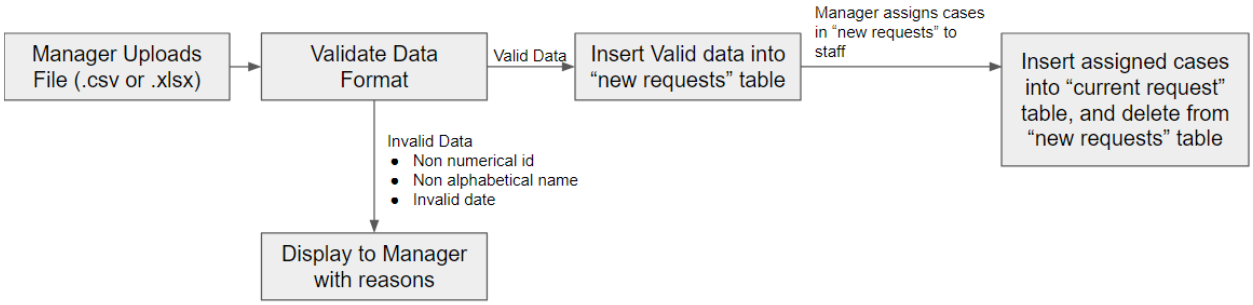
Iteration 2

ID	Story (Name & Description)	Story Points
1	Create diagram for iteration 2 Create diagram for the flow of uploading new cases to the database	1
2	Unit tests for data validation - upload new requests	1
3	Validate Uploaded Data Check if there is any invalid data in the uploaded file. Display such data.	2

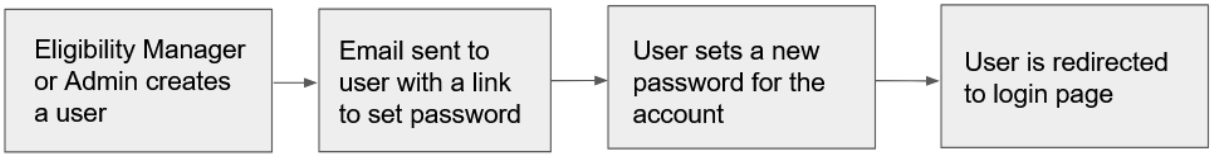
4	Ability to view and assign incoming requests as an Enrollment Manager Add Assign Cases Backend Controller View all New cases Assign a case to a user id Table to view the New cases	**4**
5	New database table - Current Requests - for managing all requests | Migrations Set up new database table in the postgres for the current cases	**1**
6	Create Password Page UI and Controller Create a UI for setting up password page. Create a corresponding controller.	**3**
7	Create diagram for setting password flow	**1**
8	Add unit tests for users controller	**2**
9	Add logging to flask application and setup sessions boilerplate Setting up the login process for the application.	**2**
10	Add homepage template and navigation menus (top and left nav)	**2**
11	Integrate Manage users UI with the flask app Integrating front end design with backend functionality for manage users.	**1**
12	Add UI for managers to upload enrollment CSV data Front end design for upload new cases page.	**2**
13	Behave test cases for Set Password Feature	**1**
14	Add pylint code linting tool	**1**
15	Create Dashboard Analytics UI Make UI design for dashboard, which will serve as default page for users	**2**
16	Update UI for manage user and user list page to have matching style Fixed html/css for manage user and user list pages to have coherent style with the rest of the application.	**2**

Iteration 2 diagrams

New Cases Flowchart



Set new user password flow:

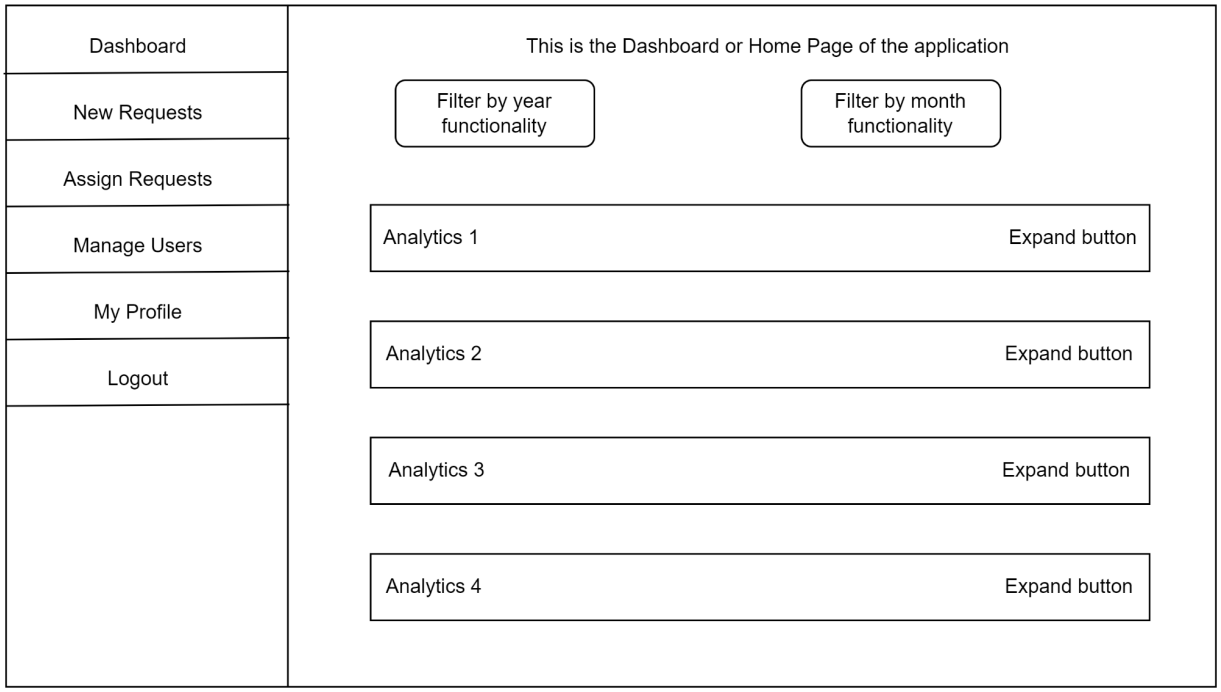


Iteration 3

ID	Story (Name & Description)	Story Points
1	Setup My Requests Page for Staff, Minor DB changes On the 'My Requests' page the staff can see the requests which are assigned only to them. They can update the case as well and save the changes.	3
2	Integrate dashboard UI with user features The dashboard UI used to link to different pages and was updated to integrate the dashboard and features together following the same format.	2
3	Add UI for users and cases Added UI for the users pages and cases pages including users list, manage users, upload new cases, and assign new cases	2
4	Fix Behave Tests for Upload Feature Since the upload file feature was enhanced and change, the behave steps had to be fixed to accommodate the change	2

5	Add Unit test cases for login addition Added unit test cases for the login controller and helper Added unit test cases for the Set password feature	**3**
6	Fix behave cases for Login and set password Added behave steps for the newly developed Set password feature.	**1**
7	Assign New Requests Popop & Rest of logic The modal popup for assigning a request to a user, populating the users in the dropdown, and sending the post request for the assignment and handling the response. Implement Unit Tests, behave tests, and the UI	**3**
8	Implement user sessions and protect auth routes Added @login_required and @admin_required decorators to prevent users from accessing protected routes without logging in.	**4**
9	Iteration Diagram update Create updated design diagram for the new homepage	**1**
10	Fix behave tests for user and admin controllers Earlier all routes were unprotected, now we have decorators to protect routes with authentication, we added behave tests to validate this.	**1**
11	Design integration Maintain uniform ui design across all pages.	**2**
12	Create flow diagram for assign case	**1**

Iteration 3 diagrams



Iteration 4

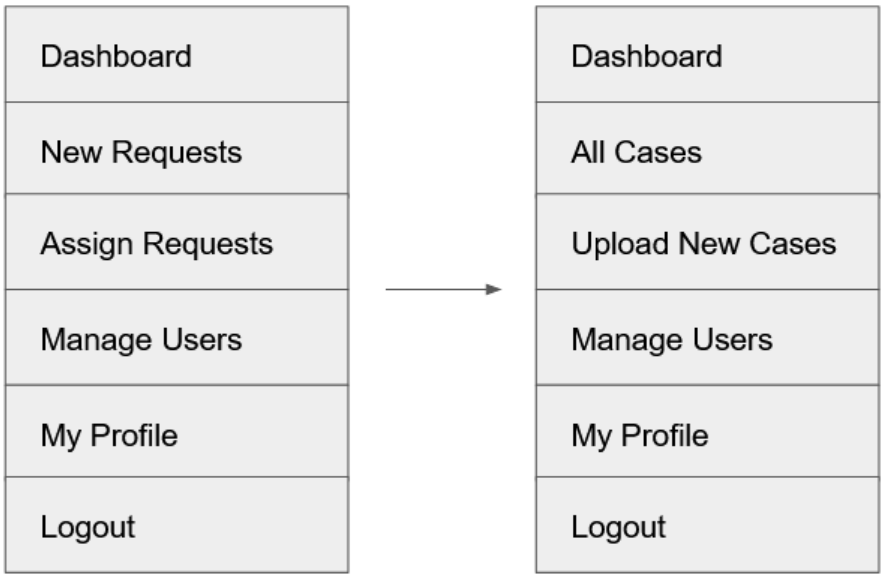
ID	Story (Name & Description)	Story Points
1	Upload cases and basic filtering Change the “new requests controller to upload” to the “cases table”. Added search and sort functionality for cases view to filter the cases in the cases template	2
2	Migrate New Requests & Current Requests to Cases Write the migration for the cases, make the cases controller to handle the new schema. Also refactor the code in the relevant places.	2
3	Design charts for data analytics page The data analytics page includes different charts to represent the six data points requested by the client. The requested data points are packets sent, packet status, family and children enrolled, family and children not enrolled, not enrolled reasons, and processing time. The charts were created using Chart.js.	3
4	Setup CI pipelines and precommit hooks	2

	Setup of CI pipelines. Everytime we push new changes, tests will be run to make sure everything works fine.	
5	Integrate backend and frontend and setup static routes - phase I Integration of some features. Setup of routes based on user roles.	**2**
6	UI changes to login pages Adjustments on Login page UI to meet client request.	**2**
7	Add verification code for new users Send email to the user added by Enrollment Manager using the email given when adding the user.	**2**
8	My Cases Page Design My Cases is where staff can see cases assigned to them after the Enrollment Manager assigns cases to them. The page displays brief information about each case, and staff can select the edit button to have full access to the information that could be edited.	**2**
9	Iteration diagram Update diagram to meet the design changes in the application navigation and new cases upload flow.	**1**
10	Develop my cases page functionality Develop backend functionality for my cases page, which is where users see cases assigned to them.	**3**
11	Make and use template html Use template inheritance in flask jinja to avoid repetition. Make a base template, and update all html files to use that.	**3**

Iteration 4 diagrams

New Cases Flowchart:

Changes in Sidebar for Enrollment Manager:



Iteration 5

ID	Story (Name & Description)	Story Points
1	My cases functionality integration Implementation of My cases feature.	4
2	Formatting my_cases and cases html files UI enhancements to my cases page.	1
3	Details feature in my_cases page My cases page is where the non-manager users can see cases assigned to them. We had designs for my cases page on the last iteration, but didn't have functionalities yet. Also, we changed the design for this iteration. We completed adding functionality.	3
4	Add Data Analytics Backend	2
5	Test cases for Data analytics Back end Client wished to see various analytics based on the current database on the homepage.	3

6	Cases Filter using Checkbox	**3**
7	Edit & Assign Functionality for Different Case Types	**3**
8	Behave & Unit Test Cases page now has a filter feature, and some fields on case cannot be edited now.	**1**
9	Fix add/edit user in frontend	**2**
10	Fix add user modal	**1**
11	Increase test code coverage for main, user and admin controllers	**3**
12	UI changes to manage users page UI changes were made to the manage users page to match the UI of the other pages.	**2**
13	CSS changes to all pages CSS changes were made to improve the website layout and design.	**3**
14	Iteration 5 my cases page and filter layout Create design diagram for my cases page and flowchart for filters in assign cases page	**1**
15	My cases - Tests Tests for my cases feature.	**3**

Understanding existing code for legacy projects

Since our project is not considered a legacy project, there was no necessity for understanding or discussing the pre-existing codebase.

Summary of each iteration

Iteration #	Points Completed	Summary
0	-	We kick-started the project by thoroughly understanding the requirements of the client. We initially created user stories and Lo-Fi mockups.

1	24	In this iteration, after discussing with our client, we determined to prioritize features to upload new requests data and users to the system/database, as they are key and central features that will work as the basis of the other features to follow. We implemented the features to some extent, along with the Login functionality.
2	24	Stories implemented - Validation of new cases, Users screen UI, Adding new user to the system
3	28	Stories implemented - Assign cases part 1, Setup My cases page, Add UI for users and cases
4	24	Tasks implemented - Setup of CI pipelines, Basic filtering of cases, UI changes to login pages, Assign cases part 2
5	38	Tasks implemented - My cases, Analytics on Homepage
Final Report	12	Slides/Presentation, Demo, and Final report, each considered as large story points

Story points completed by each member in each iteration

Iteration	Atharva Hemant Phand	Bharath Kumar Ravichandran	Chidambaram Ganesan	Daehee Han	Noah Pang	Tanmai Harish	Iteration SPs
1	4	4	4	4	4	4	24
2	5	5	5	4	5	4	28
3	4	4	4	3	4	5	24
4	4	4	4	5	3	4	24
5	7	7	7	5	5	7	38
Final	2	2	2	2	2	2	12
Member SPs	26	26	26	23	23	26	150

Customer Meetings

All meetings took place virtually on Zoom.

September 8, 2023, 1:30-2:30 PM
The meeting delved into optimizing case management by addressing challenges in the manager's role, proposing a shift to a visual dashboard, and discussing essential features like detailed rejection reasons. Manual practices, such as Excel data entry and staff interventions for payment delays, were identified for improvement. The client is ready to adjust and provide sample data, signaling a commitment to enhancing the case management system.

September 29, 2023, 1:30-2:30 PM
Positive feedback on overall progress, happy with the pace.
Demo - Uploading of new cases as a CSV file.
New requirements - Ensure that only correct cases are uploaded when new cases with incorrect information are submitted.
Priorities for iteration 2 were discussed - UI for adding a new user, Implementation of email feature for new users to set the password, integration of UI and backend for Users feature.

October 13, 2023, 1:30-2:00 PM
Positive feedback on overall progress. All committed tasks were successfully completed.
Demo - Upload New cases with validation, Login feature including email flow, Users UI
Enhancements were discussed - Protect routes according to type of user, instructions for password creation were suggested on the password initialization page, search cases functionality, dashboard analytics requirements to some extent.

October 27, 2023, 1:30-2:30 PM
Constructive feedback on the work done.
Demo - Full login flow, Routes introduced according to the user logged in, Sidebar
Enhancements - Terminology shift from “Requests” to “Cases”, Schema changes, Unified cases page requirements discussed, filters on the analytics page.

November 3, 2023, 1:30-2:30 PM
This meeting was to show progress on iteration 4.
Demo - Enhanced Dashboard, Add users, Manage users functionality.

November 10, 2023, 1:30-2:00 PM
The client provided valuable feedback and specific requests, including the suggestion to calculate processing time and enhance user experience by adding a placeholder to the search bar in the Assign Cases page. They also proposed a filter dropdown bar for better organization and outlined editable fields, emphasizing certain non-editable parameters. These refinements aim to align the system more closely with the client's operational needs and preferences, showcasing a collaborative effort to optimize functionality.

November 29, 2023, 1:30-2:00 PM
Demo - Analytics Dashboard page.
The client was happy with the implementation, as this was one of the important requirements. Some minor enhancements were suggested in the analytics

December 1, 2023, 2:00-2:30 PM
Final meeting with our client.
Demo of the entire application. The client confirmed that all requirements which were planned at the beginning of the semester were successfully met.

BDD/TDD Process

We have written our test cases using the behave framework. In our BDD process, we used Behave to define and execute behavioral tests. These tests are written in a natural language format, making them accessible to non-technical stakeholders like Eligibility Managers, Eligibility Supervisors, and Eligibility Specialists of ChildCareGroup. Generally, this process typically involves collaboration between developers, QA engineers, and product owners to create feature specifications in the form of scenarios. Behave then translates these scenarios into executable tests, ensuring the implemented features align with the desired behavior.

Benefits

The introduction of Behave tests proved instrumental in the early detection of issues during the development cycle. By adhering to Behavior-Driven Development (BDD) practices, we ensured that the features we implemented precisely aligned with the expected behavior. Concurrently, TDD played a crucial role in promptly identifying and rectifying defects as soon as the code was written. Furthermore, TDD's emphasis on writing modular and maintainable code became evident as each piece of functionality was accompanied by a comprehensive set of tests. This approach facilitated smoother modifications and enhancements, minimizing the risk of introducing regressions.

Challenges

While the benefits were significant, there were challenges encountered in the adoption of BDD and TDD. Initially, writing tests before code introduced a slowdown in the development process. There were instances where the team faced uncertainty about the outcome of certain features, making the creation of corresponding test cases a challenging task. The pressure to deliver quickly sometimes hindered the allocation of sufficient time for comprehensive testing.

Moreover, the ongoing evolution of the codebase posed a challenge in maintaining the tests themselves. As the software underwent changes, tests required updates to accurately reflect modifications in functionality. Despite these challenges, the overall advantages of early issue identification, alignment with expected behavior, and code maintainability outweighed the initial hurdles encountered in the implementation of BDD and TDD.

Configuration Management Approach

Our configuration management approach was streamlined and efficient. We did not find the need for any spikes during our development process. Our primary release branch, named 'main,' served as both the main and release branch for our repository. To facilitate feature development and testing, we maintained separate branches for development and testing, both branched off from the 'main' branch.

For each feature and iteration, dedicated branches were created. Merging code was systematically managed through the use of pull requests, providing a controlled and collaborative process. During the iterative development cycles or sprints, we established five release tags, facilitating version tracking and organization.

When it came to deployment to Heroku, we pulled directly from the 'main' branch, ensuring that the deployed version aligns with the latest stable codebase. This configuration management strategy contributed to a well-organized and systematic development lifecycle, allowing for effective collaboration and version control.

Issues in the production release process to Heroku

We didn't face any issue in production. We saw the Heroku stacks present for the python Heroku build packs on our python version requirements and decided to use the Heroku-22 Stack.

Issues when using AWS Cloud9 and GitHub and other tools

We did not utilize AWS Cloud9. We used Git and GitHub for version control in our project. While we didn't encounter any specific issues with GitHub, leveraging it for project management posed initial challenges. Notably, assigning numerical story points to stories or issues lacked a direct and straightforward method. In the early iterations, another hurdle was the absence of continuous integration (CI) configuration for our repository. This led to instances where pull requests, which potentially broke tests, were merged without adequate testing. To address this, we later streamlined our workflow by integrating CI through GitHub workflows. This enhancement automated the execution of unit and integration tests for each commit push and pull request, ensuring a more rigorous testing process before merging changes.

Testing, Static Code Analysis, and Other Tools Used

Testing

For unit testing, we employ the *pytest* package, which offers a robust framework for writing and executing tests in Python. Additionally, we use the *coverage* package to monitor test coverage, allowing us to gauge the effectiveness of our tests and identify any gaps in the testing suite.

Static Code Analysis

For static code analysis and adherence to coding standards, we rely on *Pylint*. Pylint thoroughly examines our Python codebase, providing feedback on potential errors, code smells, and adherence to best practices. This proactive approach helps maintain a high level of code quality from the outset.

Code Formatters

To ensure consistent code formatting according to the PEP 8 standards, we incorporate *autopep8* into our workflow. This tool automatically applies necessary code formatting adjustments, reducing the manual effort required for adhering to coding conventions and enhancing code readability.

Pre-Commit Hooks

We have *pre-commit hooks* to check for various code quality and formatting aspects, which automatically run before each commit. These hooks include inspections for added large files, debugging statements, proper end-of-file handling, encoding pragmas, merge conflict checks, trailing whitespace, and YAML syntax validation. Furthermore, we utilize the autopep8 hook to format code according to PEP 8 standards automatically. Additionally, we include hooks from Lucas-C's pre-commit-hooks for removing carriage return-line feed (CRLF) and tabs from the codebase. This ensures our code adheres to best practices and maintains a consistent style throughout the repository.

Continuous Integration (CI)

We leverage *GitHub Actions* as our CI tool, configured through GitHub webhooks. Our CI workflow encompasses linting, unit testing with code coverage, and integration testing.

- The lint job checks the codebase for adherence to coding standards using pylint.
- The unit-test job runs unit tests with code coverage, ensuring a minimum coverage threshold of 90% for specified files.
- Additionally, the integration test job performs behavioral testing using Behave, all within a PostgreSQL service container.

This CI setup helps maintain code quality, identify issues early in development, and ensure the reliability of our application.

Repo Contents. Scripts and Deployment

Directory Structure

Root

- /app: Contains the main Flask application structure.
- /documentation/Fall2023: Houses iteration documentation related to the project.
- /features: Holds Behave features and steps.
- /migrations: Manages database migrations.

- /tests: Contains additional tests for the application.

Flask Application (/app)

- /controllers: Controllers for different components of the application.
- /decorators: Custom decorators are used in the application.
- /exceptions: Definitions for custom exceptions.
- /helpers: Utility functions and helper classes.
- /seeds: Scripts for seeding initial data into the database.
- /static: Static files such as stylesheets, images, etc.
- /templates: HTML templates for rendering views.
- /utils: General utility functions.
- models.py: Definition of database models.

Brief Descriptions of Controllers

- admin_controller.py: Manages administrative functionalities such as user management and permissions.
- cases_controller.py: Handles operations related to cases, likely including CRUD operations.
- my_cases.py: Specific controller for actions related to individual user's cases.
- new_requests_controller.py: Manages the creation and processing of new requests.
- role_controller.py: Deals with assigning roles and managing permissions.
- user_controller.py: Handles user-related operations, including authentication and profile management.

Dependencies

- Dependencies are listed in the requirements.txt file. The major dependencies are Flask and Flask-Migrate. On the UI side, we use jQuery and Bootstrap as additional libraries loaded using a CDN
- Specific versions for Python, Flask, and other dependencies are specified in the file.

Configuration

- The application connects to the database on test/development/production based on this flag “FLASK_ENV”, the “FLASK_SECRET_KEY” is used for logging, validation, and security purposes. The “DATABASE_URL” contains the connection string (URI) for the development/production Postgres instance, and the DATABASE_URL_TEST has the string for testing.

Database Setup

- The application uses PostgreSQL as the database. Users need to set up a separate user and database with full privileges. Steps have been provided for this in the README of our repository

Deployment Process:

- A continuous integration and continuous deployment (CI/CD) pipeline is in place, for every Pull Request raised, a set of tests and lint checks happen.
- Successful pull requests merged to the main can be deployed to Heroku directly.
- The Procfile on Heroku uses gunicorn (WSGI HTTP Server, Web Server Gateway Interface) to run the application in the production environment.

Scripts

- Alembic is used internally to manage migrations, when “flask db upgrade” is called, the Alembic script is called to apply any pending migrations. The seed.py file is used to initialize the database with user and role data.

Testing

- Integration, unit, and functional tests are part of the CI/CD pipeline. To run them separately we use the unittest module to do.
- The command “python -m unittest discover -s tests” which runs all the unit tests defined in the tests directory, “behave” command runs all the features (behavioral tests) for the application.
- Running the same commands with “coverage” also provides the coverage whose report can be seen by “coverage report”

Heroku

- As mentioned above, the Procfile handles the starting of the server in Heroku after deployment.
- App Details: We use heroku-22 stack with Python for our application. Postgres is an add-on used for the database for our application in production. The build-pack heroku/python