GROUP 1 PORTFOLIO

HRDC Inventory Management System

ESOF 423 – Spring 2025

Teddy Bagley, Emma Bergman, Luke Breum, Quinlin Gregg, Cody Hager, Matthew Parker, Alan Zetzer

---

Section One: Program

For our project, we built an inventory management system (IMS) for HRDC, a local nonprofit. Specifically, we built the IMS for HRDC's warming center to help them keep track of their inventory levels and analyze their usage of inventory items. Previously, the warming center did not have an IMS in place. Instead, they used things like spreadsheets to manage their inventory. This posed a few issues. They were running low, or even running out, of inventory items without knowing it. The system wasn't very organized, as there wasn't one central place that they were tracking their inventory. Jenna, our point of contact from HRDC, wanted a seamless way to track inventory to replace this system. The goal was for there to be one application where HRDC staff and volunteers could manage all of their inventory needs. The system needed to be intuitive and work on desktop and mobile devices.

Our inventory management system provides a solution to this problem. Jenna and her team can manage their inventory items, be notified when they run low, track data about the price and usage of their inventory items, and keep them organized in different categories. This application provides a seamless way to manage their inventory system, which can help the staff and volunteers at HRDC make sure the warming center has everything it needs.

The code for this project is stored on this github repository: https://github.com/423S25/repo1
The publicly hosted site is located at https://hrdc-ims-staging.fly.dev

Section Two: Teamwork

Team Member 1

Team Member 1 primarily focused on front end development of the UI and inventory management system. In the early stages group member 1 worked closely with the design team to help out together a plan that was visually appealing and workable for the engineering team. Group Member 1 greatly assisted in communicating and helping other members get on track and maintain a level head when they experienced blockers and issues. During development Group Member 1 worked on multiple parts of the system including, but not limited to: the main table page, report page, filtering of main page content. Group Member 1 researched different uses of charts.js since the client requested multiple different forms of data reporting from the system. Additionally Group Member 1 tested the functionality of the sign in page, product page, and other reports within the project and fixed several bugs throughout development. Overall, the focus of Group Member 1 was to make sure the system was usable so that every member of the HRDC organization can use the system effectively and easily.

Team Member 2

Team Member 2 worked on various different parts of the software. They were responsible for the system that takes snapshots of the inventory to be used for future analysis and reports. They were also responsible for some aspects of styling on the desktop version, notably on the individual product page and the main inventory table. The entire mobile version of the site was team member 2's responsibility, which will primarily be used by the volunteers. The mobile site

has the ability to set the current stock and to search, filter, and sort the items being tracked. This team member added the product-specific reports page to give the HRDC more insight into their inventory usage. They added stock unit types, so the HRDC can separately track individual units and various bulk units, each with different prices. They implemented the designers' 404 and 401 page designs, as well as adding form validation to most forms on the site. This also included validating every product image being uploaded to the site. Finally, they did the styling for the user and developer documentation to make it more appealing.

Team Member 3

Team member 3 contributed to the project in a wide variety of ways. Primarily, they were responsible for lots of the backend development as well as maintaining and setting up the publicly hosted application. Some of the major application features they worked on were access control, email notifications, and writing end-to-end and unit tests to run in a github action. Team member 3 was the primary maintainer of anything related to these three areas, and was responsible for making sure they stayed functional throughout the semester. They were also responsible for making sure that the publicly hosted deployment of the application was up to date and ready to be used by the client. This included researching and using the tools provided by fly.io, our hosting provider, to ensure that the user's data wasn't lost in the case of a bug or outage. To ensure that the publicly hosted application reflected the current state of the application, team member 3 set up a github action that updates the application from every contribution to the repository.

Team Member 4

Team member 4 put their efforts in implementing Team member 6 and Team member 7's designs onto the project's front-end codebase. This mainly consists of modifying the HTML files added by the other team members and giving them a CSS file to the corresponding pages. Their

major contributions include the looks of: the navigation bar, the login page, new product and new category UIs, the search bar, and the filter inputs. Team member 4 had to be on top of many of the changes Team members 1-3 and 5 were making to the product due to the changes heavily modifying the look and feel of the website. This also saves the trouble of each team member having to do the styling, focusing their work on the features of the product. When changes were made, designs were implemented to the features added, with some improvising having to be done whilst trying to replicate the feel and vision of Team members 6 and 7's designs. Their number one priority through the entire project was to ensure the product was usable and accessible to the users using the product.

Team Member 5

Team Member 5 was actively involved in both the backend and frontend development of the project. Their primary focus was on implementing CRUD (Create, Read, Update, Delete) operations for products and categories. This work involved setting up and configuring the necessary database structures, writing backend logic to handle data manipulation, and ensuring smooth interaction between the application and the database tables. On the frontend side, Team Member 5 utilized HTMX to build modals and forms that allowed users to perform CRUD operations directly from the interface without requiring full page reloads. This improved the user experience by making the application more responsive and interactive. In addition to CRUD functionalities, Team Member 5 contributed to the product page features, expanding the customizability of the app. They also implemented various other components, including category icons for better visual navigation, a CSV export feature to allow users to download data, filtering capabilities to help users search and sort through content more efficiently, and email lists.

Team Member 6

Team member 6 focused on the user experience and user interface design of the inventory management system. They conducted UX research and utilized the design process. This included the creation of a low and high fidelity wireframe in collaboration with Team member 7. As part of this process, Team member 6 pushed for an onsite visit to the shelter where they took detailed notes and implemented feedback from future users of the software. Team member 6 worked on creating intuitive, accessible interfaces with Team member 7, this included creating visual systems for different elements of interaction including buttons, input boxes, drop down menus, and more. Beyond interface design, Team member 6 helped ensure our team remained user-focused throughout the development process, advocating for features that prioritized clarity, ease of use, and adaptability for a wide range of users.

Team Member 7

Member 7 also focused on user experience and user interface design for the inventory management system. They researched other inventory systems and UX solutions, then created low-fidelity wireframes in collaboration with team member 6. After solidifying the layout and functionality of each page, they designed high fidelity pages implementing user interface design principles. Team member 7 focused specifically on the mobile pages of the inventory system, designed to be used by employees and volunteers to count and keep track of the current inventory of each item. They also created a cohesive icon set, and designed various other pages that will be used throughout the inventory management system. Additionally, they helped team member 6 keep the project aligned with the client's needs, ensuring that the design was easy to use, accessible to tech-illiterate users, and functions as intended.

Section Three: Design Pattern

Our project naturally followed the Model-View-Controller (MVC) pattern. The MVC pattern is a design pattern that separates the application into three separate components: the model, the view, and the controller. The model component is responsible for handling the logic for persisted data. We have the model component of our project **here**. The view component is responsible for presenting the data and creating a user-friendly interface. The view component for our project is located **here**. Finally, the controller is what handles the flow of requests and acts as an intermediary between the model and view components. The controller component in our project is located **here**. Here is one very simple route from our controller that demonstrates the use of the MVC pattern, with comments added for clarity:

```
@app.get("/settings") # CONTROLLER
@admin_required
def get_settings():
    accounts = User.all() # MODEL
    emails = Email.get_all_emails() # MODEL
    return render_template("settings.html", user=current_user, accounts=accounts, emails = emails) # VIEW
```

As you can see, the controller component is what handles the logic between the view and model components by using a handler on the /settings route. The controller gets persisted data from the model component, emails and accounts in this case, and sends that data to the view component. The controller is the bond between the model and view. Without the

controller, the data from the model would never reach the user and the UI would have nothing to display.

The Model-View-Controller pattern is a go-to for many web applications and is well-suited for a hypermedia-based framework, which is what we used. Using the MVC pattern to split up the full application into distinct parts comes with many benefits. One of the main benefits is that it helps make the codebase more readable and maintainable. Another benefit of using the MVC pattern is that it helps support CI (continuous integration) by making it easy to test different parts of the application independently. For example, we have a unit test suite for the model component of our application **here**. Overall, the MVC pattern provided a clean and logical way to split up this application while making it easily extensible and understandable.

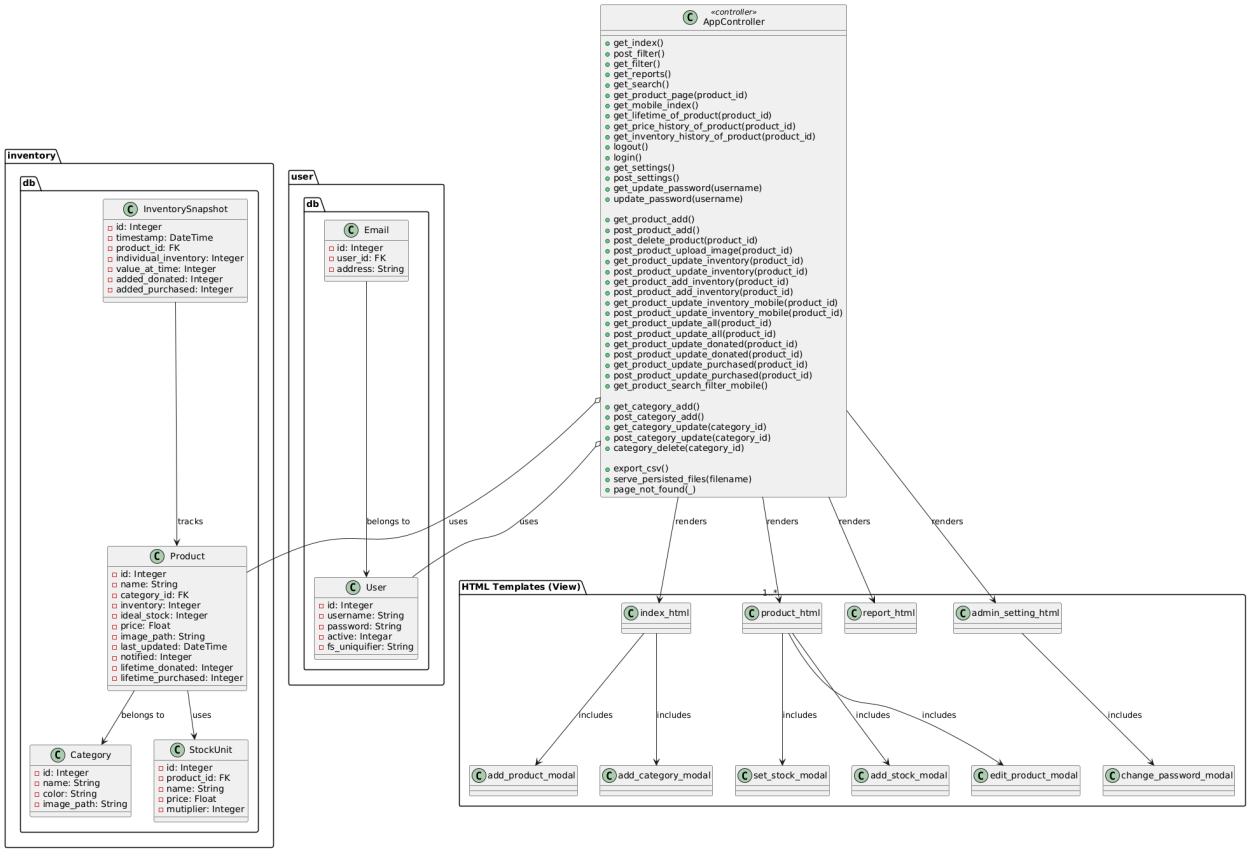
Section Four: Technical Writing

User documentation can be found **here**.

Developer documentation can be found **here**.

Section Five: UML

This is a UML Class diagram showing the desktop version of the inventory management system. This diagram shows the MVC design pattern with the model represented by the two databases, the view by the html templates, and controller that connects the two. There are some missing templates as well as a mobile view; however, for simplicity and size constraints only the desktop version is shown in this diagram.



Click **here** for an expanded view.

Section Six: Design Trade-Offs

The main design trade off that our group encountered was the decision to use a hypermedia-style technology stack instead of more popular JavaScript frameworks (such as React). Our inventory management system prioritized simplicity and scalability by using hypermedia-driven development rather than relying heavily on JavaScript. To achieve this, we used HTMX to power dynamic front-end interactions, keeping the user interface simple with primarily basic HTML and CSS. Chart.js was also incorporated for data visualization where necessary.

While this approach minimized front-end complexity, it shifted more operational logic into the back-end, resulting in a more complex and sometimes jumbled routing structure within the Flask controller and helper files. Another trade-off we encountered with the hypermedia-based approach is that while it kept our UI very simple and readable at first, it became harder to implement more complex front-end features. While our approach let us get the application running quickly with much less setup compared to something like React, building and polishing the UI was slower than it would have been with prebuilt component libraries.

Although setting up the initial routing was straightforward, the reliance on server-side handling for dynamic behavior made future changes and expansions to the system potentially more challenging. We chose this architecture understanding that the system would likely require minimal ongoing development, and provided thorough documentation to assist any future developers with maintaining and updating the system as needed.

Overall, the hypermedia-driven approach was a good fit for our project. The inventory management system doesn't require a very complicated user interface, or one that needs to be updated extremely frequently. It is also a CRUD (Create, Read, Update, Delete) application, which is another good use case for hypermedia. The simplicity offered by this approach made it easy for everyone on the team to contribute, regardless of past experience. While some things might have been smoother with more popular front-end frameworks, the team agreed on using the hypermedia-driven approach for this project and the finished product turned out well.

Section Seven: Software Development Life Cycle Model

We used the scrum framework as our development life cycle model throughout this project. The scrum framework is one of the most popular agile methodologies. There are a few scrum strategies in particular that we consistently practiced. First of all, we had a few “standup” or “scrum” meetings every week. Normally, these meetings are supposed to happen daily, but since we were contributing less hours per week than what scrum was designed for, 2-3 meetings a week was the most realistic option. These scrum meetings were concise, and each team member discussed what they were currently working on, if there were any issues, etc. We also completed our work in 2-week sprints, which is a popular agile/scrum strategy. At the beginning of each sprint, we would have a planning meeting where we set our goals and assigned our work for the sprint. At the end of each sprint, we would fill out a retrospective document and assess whether or not we completed our goals, making adjustments where necessary.

The agile-scrum framework provided many benefits for our project. The 2-week sprints helped break the daunting task of building the application into more manageable parts. They also provided us with opportunities to demo our latest work to the client and get feedback after each sprint, which was crucial to the success of this product. The scrum meetings helped keep the team on track, encouraging collaboration when applicable. The retrospectives were useful for reflecting on any issues we had and what we could do better.

One downside of using scrum for our project is that it requires some time and effort to implement correctly, which sometimes cuts into our development time. Doing tasks like creating burndown charts, keeping an organized retrospective, etc. can be time-consuming. Since we are all full-time students and only have so many hours to devote to this project, we had to decrease our development time in order to correctly use the scrum framework.

However, using scrum as the development life cycle for our project was a good choice and enabled us to efficiently work as a team. In addition to the benefits mentioned above, the scrum strategies that we practiced naturally created a collaborative environment, which improved the team's morale. As scrum is one of the most widely used development models in today's software industry, it is very likely that the skills we learned while using scrum for this project will be helpful in the future.

Inventory Management System

Developer Documentation

v1.0.0

Teddy Bagley, Quinlin Gregg, Cody Hager, Matthew Parker, AJ Zetzer

Getting Started

1. Make sure you have Python 3 installed by running `python --version` or `python3 --version`. If not, download it here. This project is known to work with Python 3.12 and 3.13, but other versions should work too.
2. Clone the repository to your local machine with `git clone https://github.com/423S25/repo1.git`.
3. Navigate to the directory where the repository was cloned. Run `pip install -r requirements.txt` or `pip3 install -r requirements.txt` to make sure you have all of the needed dependencies installed.
4. Create a file named .env in the root directory of the project with the following fields. Note that each field in your file should follow a format similar to this: `NAME="value"`
    a. `ADMIN_PASSWORD`
    b. `STAFF_PASSWORD`
    c. `VOLUNTEER_PASSWORD`
    d. `SENDGRID_KEY` - this is used for email notifications. Reach out to a developer if you need this and they can get you the correct key. We can't store this on github because SendGrid won't allow it to be stored on a public repository.
    e. `APP_SECRET_KEY` - flask uses this for session management. You can view this for information on how to generate a key.
5. Run `python app.py` in the root project directory to start the site.
6. Go to `http://127.0.0.1:5000` to view the current site.

Contributing

1. Make a branch on the GitHub repository.
2. Run `git pull` on the main branch.
3. Run `git switch <branch-name>`.
4. Do your work, add your features, and save your changes to your branch.
5. Push your changes. **Make sure you are on your branch, not main**!
6. On GitHub, make a pull request to merge your changes into the main branch and resolve any merge conflicts.
7. Tag someone for a review.
8. Once two other contributors approve, merge your pull request.

Access Control

Three users are available:

- *Administrator*: Has access to everything.
- *Staff:* Can update inventory, but cannot add or delete items
- *Volunteer:* Currently the same as **staff**.

On local deployments, the passwords are pulled from your .env file (see the "Getting Started" section). If you are developing and sick of re-signing in after every server restart, consider temporarily changing `app.config['SECRET_KEY']` to a constant value.

Project Structure

The current structure is as follows:

- `app.py:` the main web server
- `/templates:` the HTML pages to be rendered
- `/src/model:` the database code, containing the Product, InventorySnapshot and User classes.
- `/src/common:` the Flask forms
- `/static:` static resources like images, fonts, and JS scripts
- `/docs:` the documentation pages (similar to what you are seeing now.)

Testing

Developers should be sure to test the core functionality of the application with every contribution. At least one approving reviewer should do a code review and test each contribution.

Automated Building and Testing

There is a GitHub action that will run the app and then test it with selenium. Right now, since the UI is going through a lot of changes, the tests are:

- logging in
- logging out

- adding a product

If you make a PR and the tests fail, you should investigate and see whether you made a breaking change or if the tests just need to be rewritten because of DOM changes.

There are also unit tests in the automated-testing/unit directory. Developers are responsible for fixing/updating these tests as they add features.

How to release a new version

To release a new version of this software, update the code and send the updated code to the server.

Additional considerations

The code for the project is stored here. Developers should provide meaningful comments when contributing source code.

Inventory Management System

User Documentation

v1.0.0

Teddy Bagley, Quinlin Gregg, Cody Hager, Matthew Parker, AJ Zetzer

Starting the software

The software can be run by entering the URL into a browser. For a local deployment, refer to the developer documentation.

To access our current deployment located here, you can sign in with the "admin", "staff", or "volunteer" usernames. To make our demo environment easily accessible, we set the password for each to be "password". The admin can change the passwords for all 3 accounts in the "ADMIN SETTINGS" page.

Using the software

There is an intuitive user interface that will allow users to create, read, update, and delete entries in the inventory. A tutorial for using the software can be found here.

The Table

The table shows all the current products in the inventory

- By clicking the arrow on the right side of a product, you can view more info about the product.
- The table can be filtered by the menu on the left of the page as well as the banner along the top of the page.

Creating New Products and Categories

In the upper left corner, you are able to create new products and categories.

Categories

- A new category is created by pressing the "NEW CATEGORY" button.
- NOTE: In order to make a product you must first have a category to put the product in.
- Once in the new category menu, name your category and select a color and icon.

Products

- A new Product is created by pressing the "NEW PRODUCT" button
- Once in the new product menu, fill out the form. Here you can create units. You can create as many units as you like. For Example, there is an individual unit that contains 1 individual unit. You can then create a "Box" unit that contains 20 individual units.

Product Page

The product page is where you can edit and interact with a specific product.

- Click on the camera icon in the upper left corner to add or change the product image.
- "SET INVENTORY" button can be used to change the total inventory of the product.
    - This can be used while taking stock to input the **total** amount of stock.
- "ADD INVENTORY" button is for in-taking a new product into the system.
    - In this menu the input amount is added to the total stock.
    - This is where you specify if a product was purchased or donated.
- The drop down ("...") next to ADD INVENTORY is where you can update information about a product and permanently delete the product.

Logging in

You can log in as an admin, staff, or volunteer. Reach out to your administrator to obtain the proper credentials.

Email Notifications

In order to receive email notifications, the admin should click the "ADMIN SETTINGS" button in the top right corner of their screen and type in their email address. Then, email notifications will be sent whenever a product's inventory reaches 50% and 25% of the ideal stock.

Exporting the inventory data to a CSV file

The application allows users to transform the inventory data into a CSV file. The button to perform this action is located on the REPORTS page.

Reporting a bug

Problems can be reported by making a GitHub issue here.