Chemical and Inventory Management System
SoFab Inks
Team 3

Hilton Benson, Blake Hourigan, CJ Johnstone, Maggie Jackey

November 29, 2024

Contents

1 Introduction/Executive Summary

1.1 Introducing SoFab Inks

SoFab Inks is a chemical manufacturing startup that was spun-out from the University of Louisville, Conn Center for Renewable Energy Research with support from the US DoE. SoFab inks focuses on accelerating the commercialization of Perovskite Solar Cells through the development and manufacturing of functionalized inks that improve cell efficiency, reduce module cost, and enable scalable manufacturing. [1]

1.2 SoFab Inks Inventory Management Problem

SoFab Inks is doing important work in the field of solar cell technology, helping to drive humanity towards a cleaner, more energy abundant future. The problem, however, is that the team currently faces issues with managing inventory. These challenges prevent SoFab's talented team from working on the most important aspects of their work. These challenges include expending valuable energy on menial tasks like locating inventory, managing a growing number of shipments manually, scouring inventory entries found across several Google Sheets or handwritten labels to pinpoint important product information, and more.

To aid SoFab in these challenges, this semester Team 3 was tasked with developing a more efficient method of managing inventory items. To accomplish this, the team employed various existing software solutions, including database software, CRUD[1] user interface software, and solutions to containerize this software together into one package. The team also developed custom software to provide features not available in the existing software solutions.

CAC 3. Communicate effectively in a variety of professional contexts

2 System Description

The following sections describe the specifications that were formulated as a result of communication between Team 3 and the SoFab Inks team.

[1]Database term for Create, Read, Update, Delete.

2.1 Needs Assessment/System Requirements

Following assignment to this project, the team assembled and met virtually with the SoFab Inks team for brief self-introductions, an overview of the current issues facing SoFab, and to gain an initial insight of what the SoFab team was looking for in a solution to these issues.

The SoFab team described the current state of inventory management at the company which included problems such as shipments arriving to incorrect customers, an inability to pinpoint important information about products as they progressed through the manufacturing process, and difficult to track remaining volumes of chemical products. In these discussions, SoFab also emphasized the importance of a solution begin *simple and easy-to-use*. As the team is primarily constructed of chemical engineers or business people, technology was not a strength.

The company also made it clear that the ability to generate item labels for internal tracking of should be a priority. These labels would allow members of their team to easily scan a barcode to view the details of an item. Finally, it was also made clear that the company also desired the ability to generate shipment labels for their products as well. These labels would differ slightly from their internal tracking label counterparts with the inclusion of chemical hazard information. This information would be included as a way to reduce harm for customers handling SoFab's chemicals upon receipt.

After this initial discussion, Team 3 began to brainstorm potential solutions to the problems presented. Immediately, 3 main pieces of software jumped out at the team as critical pieces to what would be the final product.

1. **A Database System** - In order to move the company away from the usage of Google Sheets and toward a more efficient, safe and redundant, user-friendly solution, Team 3 knew it would be necessary to select an existing database software solution. While the team was unsure of what specific solution would be chosen, it was sure that one of these solutions would be required.
2. **A User-Friendly Database Interface** - While a database solution would be an incredible improvement on its own, it would be useless to the SoFab team if there were not a simple and easy way to interact with the underlying data. Again, the team was unsure of what specific solution would be chosen, but a few requirements from discussions with SoFab were clear.
    - **Clean, Simple, Easy-to-Use** - As previously discussed, SoFab was clear that a simple and easy-to-use solution was of paramount importance for the day-to-day usage of the product.

- **Free and Open-Source** - While this requirement was not mentioned in the initial discussions with SoFab, this requirement jumped out as important to Team 3 because this would help to avoid incurring additional costs beyond the development cost that SoFab had already paid.

3. **Software to Generate Internal and Shipment Labels** - After discussing the need for barcode and label generation software, the team found that it would likely be necessary to build custom software to meet the needs of the client. The requirements were quite specific, and would not be available in any existing commercial product. These requirements certainly would not be made available in any *free and open-source* software.

2.2 Initial System Specification

Following the identification of the 3 main categories of software that would comprise the solution to SoFab's inventory management problems, Team 3 dove into research of each individual component to identify optimal selections.

2.2.1 Selecting a Database

Firstly, it was important for Team 3 to become familiar with existing solutions that were **free and open-source**. This required conducting research on computer-science related websites and forums, and more general forums like Reddit. While a site like Reddit may not always be the most reliable source of information, it could provide a general sense of what people feel about particular software and how easy it is work with.

Many databases fulfilling the free and open-source requirement were identified, including: PostgreSQL, MySQL, MariaDB, and SQLite. As many options were available to select from, it was important to look beyond this requirement and investigate techinical specificaitions of these solutions. During this time it was discovered that **PostgreSQL** had several technical benefits over more popular rivals. PostgreSQL boasts of being a fully **ACID** compliant database system. With features such as *Write-Ahead Logging*, which writes transactions to a log file to avoid pushing a full table(s) every time a transaction is committed, *Multi-version Concurrency Control* which the PostgreSQL documentation describes as

> "This means that while querying a database each transaction sees a snapshot of data (a database version) as it was some time ago, regardless of the current state of the underlying data. This protects the transaction from viewing inconsistent data that could be caused

> by (other) concurrent transaction updates on the same data rows, providing transaction isolation for each database session."[2]

Furthermore, PostgreSQL utilizes its *Write-Ahead Log* to implement *Point-in-Time Recovery* without the need of complete backup. Additionally, since the Write-Ahead Log contains all transactions since the previous system backup, one can return to the exact state of **any** *point in time* between the most recent backup version and the current version. Team 3 felt that these benefits would be incredibly beneficial for SoFab's new database, which would be the central hub containing the whereabouts and remaining inventory of their lab. This database would also contain important product information which if lost or damaged could have severe consequences for their business.

Beyond technical considerations, ease of use was of great consideration when selecting a database, and luckily, PostgreSQL benefits from being very easy-to-use. PostgreSQL has an additional software called **PgAdmin4**, a project led by a core developer of PostgreSQL! This software integrates very well with PSQL[2] This software provides a GUI that allows developers to interact with the database, run queries, add and edit tables, view ERDs [3] for tables, and much more. This would simplify team members development tasks significantly and enable faster production.

For the reasons stated above, PostgreSQL was selected as the database software of choice for this project.

2.2.2 Selecting an Interface

While PostgreSQL would be a great choice to build a database for SoFab inks, this software alone - as stated previously - would be useless to the SoFab team by itself. Team 3 needed to produce a user interface that was simple, user-friendly, and would be capable of fulfilling all of SoFab's functional requirements. Team 3 scoured the internet, searching for a solution that would fulfil these requirements. Eventually, the open-source software Budibase was found. Upon investigation into Budibase, it was discovered that this software could be self-hosted, and if self-hosted, was *free-to-use*. This immediately fulfilled two requirements, so the team installed the software locally and began to interact with it to determine if it could fill the user-friendliness and functional requirements discussed previously.

It was found that Budibase functions in a manner similar to that of website builders like "Wix" or "Squarespace". An architect has the ability to connect to an existing database, then can select from existing templates based on the

[2]PSQL is short for PostgreSQL.

[3]ERD stands for entity relationship diagrams.

type of project at hand, and even use templates for types of pages to use. The architect can select between form-type templates where the user can edit or insert information about a row in a database table, or table-type templates where the user can view large amounts of data at a glance.

Additionally, one can drag and drop pre-made components from the sidebar, and arrange them as they see fit. These components can connect to any data source in the database. Data sources include standard database tables and even custom queries to provide maximum customizability.

While using a builder software like Budibase reduces the complexity in creating CRUD apps, they can pose a challenge in that all documentation and instructions for using the software provided strictly by Budibase themselves. The self-hosted Budibase community is not very large, so if an issue arises, an architect may be left to themselves to resolve it if that issue is not covered in the existing documentation.

Thankfully, however, the Budibase team has prioritized documentation, with instructions on how to use their many available components in various ways. They have created a page dedicated to the use of their platform, self-hosting instructions and guidance, component use, connecting the service to the database software in use, and much more. [3] The team also provides many instructional videos and how-tos on their website and on their company YouTube page. [4]

2.2.3 A Language for Custom Software

After selections were made for database and user-interface software, discussions were held on the topic of how the final major piece of the software stack - the custom label printing interface - would be constructed. Team 3 understood SoFab's requirement for this deliverable - that the software create printable labels to place on their inventory - but Team 3 imposed additional requirements that would support the team to fulfill the company's requirement. These requirements were identified as follows.

1. **Familiarity** - The most important requirement that was identified by Team 3 during these discussions was familiarity among team members with various programming languages. Due to semester imposed time constraints, it would be important to hit the ground running, without additional challenges of learning an unfamiliar programming language. A consensus was reached that team members all had prior experience with *Python*. This prior experience, paired with the languages intuitive syntax made Python an attractive choice with regard to this requirement.
2. **Community Support/Libraries** - Another important requirement concerning the selection of an optimal programming language to develop

this new software was community support and library availability. Once again, *Python* was identified as a language with great strength in this aspect. Python is known for its extensive library support, with over 589,000 projects in the 'pypi' Python package repositories. [5] Packages were identified that would allow Team 3 to easily generate barcodes, generate PDF outputs, and read from PostgreSQL databases. These features attributed to the appeal of Python for the development of this deliverable.

3. **Reliability/Durability** - The two requirements previously discussed were found to be great strengths of the Python language. However, an additional consideration for selection for this custom software was the durability of the language. A reasonable argument could be made that the responsibility of ensuring reliability and durability of code lies with the programmer, and not the language that a programmer may employ. However, humans have proven to be much less reliable than machines in many domains with defined rules.

   Recent developments in the programming domain such as the Rust language have built this fact of reality into the fabric of the language. These such languages provide memory safety by default, meaning that the programmer must have advanced knowledge of the language before performing potentially dangerous programmatic actions. Rust also provides an excellent ecosystem of tools such as 'Cargo' which both manages project dependencies and provides an easy method to run and test written code. Rust offers many benefits that Team 3 believed could be incredibly beneficial for a project that would need to be consistent and reliable day-to-day at SoFab Inks.

4. **Speed/Efficiency** - A final major consideration in choosing a programming language was the speed that the chosen language would provide after development. Performance is *always* an important consideration, but this was especially true with Team 3's plan to install many pieces of software on one machine on final deployment. On this point, Rust again was found to have great advantages. Compared to Python, Rust is far more efficient, as it is a compiled language. This means that at program run-time, no 'interpretation' of the code is required. This essentially shifts much of the heavy computational lifting to the code 'compilation' process, allowing the program to execute more efficiently at run-time.

Upon reviewing the major points of consideration, Team 3 decided that Familiarity and Comunity Support/Library availability were critical attributes in the lanuage selection process. While a language such as Rust provided significany boosts in execution time and Reliability, both of which would be beneficial to the end user, it was decided that the team had too little familiarity with this language. Learning a new language as a team would increase the complexity of the project too greatly, and endanger Team 3's ability to produce a satisfactory product on-time. It was for these reasons that Team 3 decided to select Python as the language of choice for the development of this additional software, pending approval by the SoFab team during the next meeting.

(External design document) EAC 1. Identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics and CAC 1. Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions)

2.3 Final Specifications

Little changed with regard to the major required specifications of chosen software over the course of this project. While minor changes were implemented in requirements with regard to specific data tracking points or tracking methods, these could all be accomodated within the existing framework. For this reason the majority of the final technical specifications were set during the meeting that followed Team 3's discussion of initial specifications for the system.

Team 3 met with the SoFab team to discuss the specifications that Team 3 had arrived at as results of the prior discussions. The team wanted to ensure that these specifications aligned with the needs and vision of the company. Team 3 proposed the previously discussed database and user-interface solutions, with brief technical demonstrations showcasing the potential of these pieces of software as solutions to the inventory management problem. SoFab responded positively, with few notes with regard to the technical decisions reached by Team 3.

For these reasons the following selections were made final as software solutions for this project.

- **Database System** - PostgreSQL
- **User-Interface/CRUD Frontend** - Budibase
- **Programming Language for Custom Software** - Python

(finalized internal design document) EAC 1. Identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics and CAC 1. Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions and CAC 6. Apply computer science theory and software development fundamentals to produce computing-based solutions)

2.4 System Diagrams

Detail all interfaces between the environment and the components EAC 2. Apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors)

2.5 Hardware Overview Diagram

CAC 2. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline)

2.6 Software Overview Diagram

CAC 2. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline)

2.7 Economical, Technical, and Time Constraints

3 Detailed Implementation

3.1 Hardware Detailed Implementation

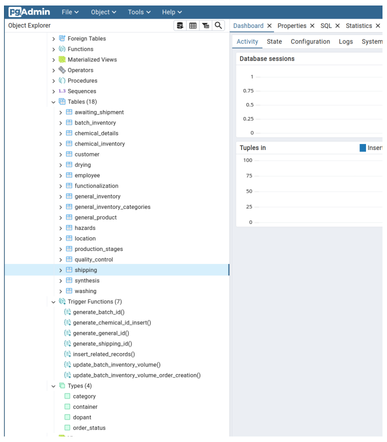
3.2 Software Detailed Implementation

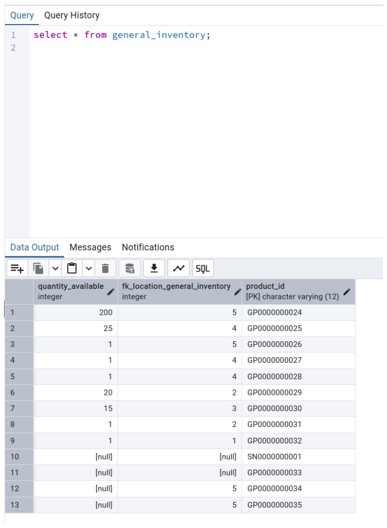
3.2.1 Docker-Compose

3.2.2 PostgreSQL

3.2.3 PgAdmin4

3.2.4 Budibase





```sql
select * from general_inventory;
```

	quantity_available integer	fk_location_general_inventory integer	product_id [PK] character varying (12)
1	200	5	GP0000000024
2	25	4	GP0000000025
3	1	5	GP0000000026
4	1	4	GP0000000027
5	1	4	GP0000000028
6	20	2	GP0000000029
7	15	3	GP0000000030
8	1	2	GP0000000031
9	1	1	GP0000000032
10	[null]	[null]	SN0000000001
11	[null]	[null]	GP0000000033
12	[null]	5	GP0000000034
13	[null]	5	GP0000000035

(a) Viewing existing tables in PgAdmin interface.

(b) Running PostgreSQL queries inside of PgAdmin4.

Figure 1

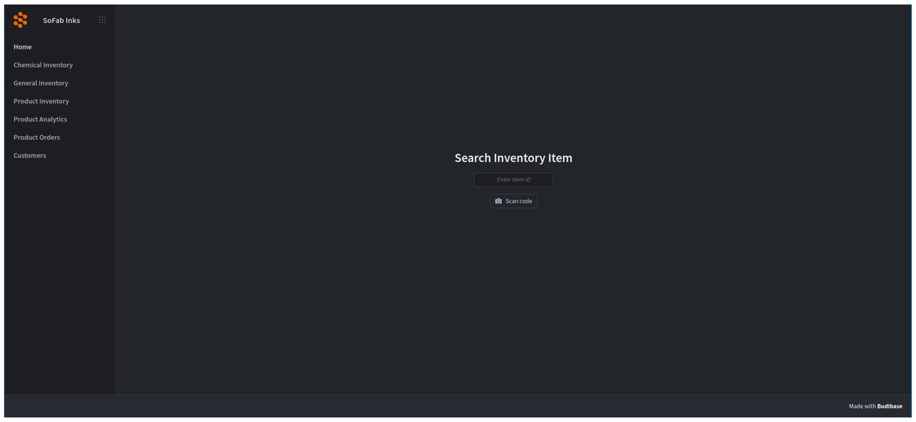
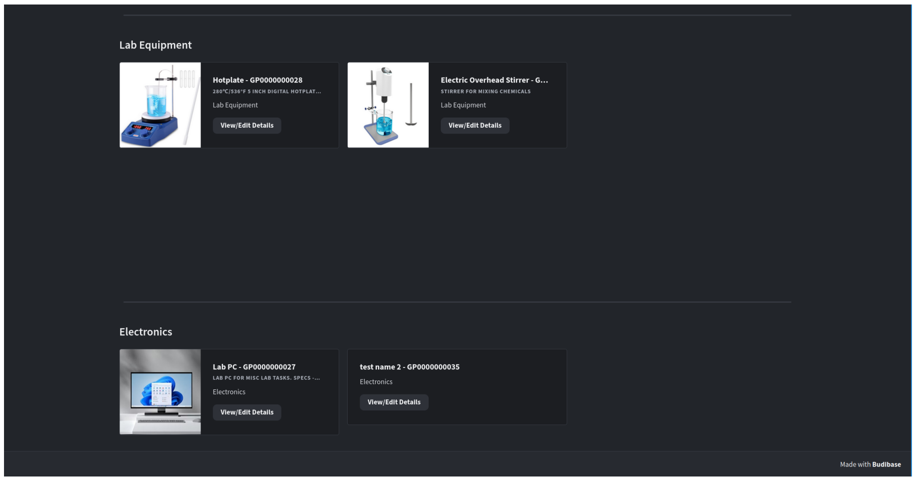


Figure 2: Budibase home page. Global search that searches all types of inventory via QR code or direct ID entry.


Lab Equipment

Hotplate - GP0000000028
280°C/536°F 5 INCH DIGITAL HOTPLAT...
Lab Equipment

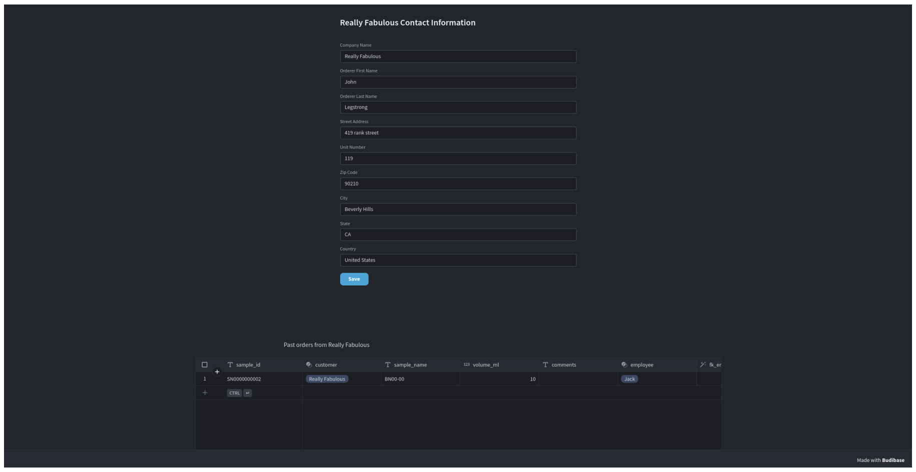
Electric Overhead Stirrer - G...
STIRRER FOR MIXING CHEMICALS
Lab Equipment

Electronics

Lab PC - GP0000000027
LAB PC FOR MISC LAB TASKS. SPECS -...
Electronics

test name 2 - GP0000000035
Electronics

Figure 3: Budibase general inventory page. Keep track of general lab items.


Really Fabulous Contact Information

Company Name	Really Fabulous
Orderer First Name	John
Orderer Last Name	Legstrong
Street Address	419 rank street
Unit Number	119
Zip Code	90210
City	Beverly Hills
State	CA
Country	United States

Past orders from Really Fabulous

	sample_id	customer	sample_name	volume_ml	comments	employee	fk_er
1	SN0000000002	Really Fabulous	BN00-00	10		Jack	

Figure 4: Keep track of customer information with Budibase. View customer information or view all orders originating from a customer

3.2.5 Python Database Insertion/Label Generation

```python
class App(Tk):
    def __init__(self, controller):
        # initializing the Tk class instance
        super().__init__()

        self.title("SoFab Inventory Managment System")

        self.style = Style(theme="darkly")

        self.controller = controller
        self.controller.set_view(self)

        self.notebook = ttk.Notebook()
        self.notebook.pack(fill="both", expand=True)
        self.notebook.bind("<<NotebookTabChanged>>", self.
            on_tab_selection)

        # filling the notebook (top tabs) with frames

        item_type_tables = self.controller.
            get_item_type_tables()
```

```python
class HazardPrecautionFrame(tk.Frame):
    def __init__(self, parent, controller, warning_dict,
        images=False):
        super().__init__(parent)
```

```python
def generate_checkboxes(self, images=False):
    self.checkboxes_frame = tk.Frame(self)
    self.checkboxes_frame.grid(row=0, column=0)

    for item in self.warning_items:
        if images:
            image = Image.open(item[1])
            resized_image = image.resize((100, 100), Image.
                LANCZOS)
            image = ImageTk.PhotoImage(resized_image)
            item = item[0]
        else:
            image = None
        var = tk.BooleanVar()
        checkbox = ttk.Checkbutton(
            self.checkboxes_frame,
            text=item,
            image=image,
            compound="left",
            variable=var,
            command=lambda var=var: self.parent.
                update_text_box(),
        )
        checkbox.image = image

        checkbox.pack(anchor="w", fill="x")
```

EAC 1. Identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics and EAC 2. Apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors, and CAC 6. Apply computer science theory and software development fundamentals to produce computing-based solutions)

4 Test/Evaluation Experimental Procedure and Analysis of Results

EAC 6. Develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions)

5 Societal Impact of Project/Legal and Ethical Considerations

include legal and ethical considerations

CAC 4. Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles)

6 Contribution of Project to Society/Expected Effects

CAC 4. Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles)

7 Engineering Standards, Constraints, and Security

EAC 1. Apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors)

8 Conclusions

9 Recommendations for Future Work

References

[1] SofaBinks. (n.d.). About SofaBinks. Retrieved November 29, 2024, from `https://www.sofabinks.com/about`

[2] PostgreSQL Global Development Group, *MVCC: Multi-Version Concurrency Control*, PostgreSQL 7.1 Documentation, `https://www.postgresql.org/docs/7.1/mvcc.html`, Accessed on: November 27, 2024.

[3] Budibase Team, *What is Budibase?*, Budibase Documentation, `https://docs.budibase.com/docs/what-is-budibase`, accessed on: November 28, 2024.

[4] Budibase, *Budibase YouTube Channel*, YouTube, `https://www.youtube.com/@Budibase`, accessed on: November 28, 2024.

[5] Python Software Foundation, *The Python Package Index (PyPI)*, `https://pypi.org/`, accessed on: November 29, 2024.

Appendices

A Customer Contact Information

B Data Sheets

C Additional Drawings and Diagrams

D Source Code

E Experimental and/or Simulation Test Results

F Software Installation Instructions

G User Manual

H Quotes, Including Ordering Information

I White Papers