

## **Insurance Claim Management Hub**

### **(The Claim Hub)**

Tejas Kulkarni ([tejas.kulkarni@sjsu.edu](mailto:tejas.kulkarni@sjsu.edu), SJSU ID: 015528539)

Anubhav Sawhney ([anubhav.sawhney@sjsu.edu](mailto:anubhav.sawhney@sjsu.edu), SJSU ID: 012032137)

Alfred Koh ([alfred.kohsenyi@sjsu.edu](mailto:alfred.kohsenyi@sjsu.edu), SJSU ID: 016702881)

Aung Bo Bo ([aungbo.bo@sjsu.edu](mailto:aungbo.bo@sjsu.edu), SJSU ID: 017368013)

Kaung Sithu Hein ([kaung.s.hein@sjsu.edu](mailto:kaung.s.hein@sjsu.edu), SJSU ID: 017376619)

CS 157A-Sec-01

Dr Ramin Moazeni

05/10/2024

**Goals and Description of the application**

Hospitals around the country receive numerous patients, and consequently have to send out numerous insurance claims for each appointment and procedure done. This process is often done efficiently and in an automated manner for larger hospitals. However, smaller practitioners have to manually send out these claims to each individual insurance provider on a regular basis. To tackle this issue, we propose to build an Insurance Claim Portal where these smaller clinics can easily upload claim information. Additionally, the insurance companies will be able to view their claims directly on our portal as well. This solution offers a centralized platform, providing a one-stop access point for all clinics, significantly streamlining the claim submittal process for them.

**Objectives and Scope:**

The primary objective of this project is to enhance the claim submission process for small to medium sized clinics. As well as create a hub for insurance providers to view claims that are submitted to them. The scope of this project encompasses the development of a “web database” that is equipped with proper access control so that clinics can view/add only their patients, and insurance providers can view only patients submitting a claim under their program. Additionally, an intuitive interface will be created to help users interact with our database of claims.

**Application/Functional Requirements**

The application will have several functionalities that help users accomplish the objectives mentioned above.

**Functional Requirements:**

- User Registration and Authentication:
  - Users can select from a number of different hospital and insurance company
  - Users can create a new hospital and insurance company if their provider is not listed
  - Users can add new patients and medical procedures

- Users are able to see their username listed on the homepage and users will be able to log in and out of the system, change their password, delete profile and provide feedback
  - Users will be able to only view claims associated to their hospital or insurance company
  - Admin account is provided so that the user can see all the claims, patients, and hospital and insurance companies
  - Admin account can also view search history, edit log and feedback of all users
- Claim Submission:
  - Users can create and submit new claims
- Claim Management:
  - Insurance providers can view and delete claims
  - Insurance providers can change the status of the claim from Pending, Approved, Rejected
- Claim Tracking and Status Updates:
  - Patients can track the status of their claims, check how much of their treatment is covered, and any comments the insurance claims have made on their claim
  - Hospitals can monitor the status of claims they have submitted.

### **Non-Functional Requirements:**

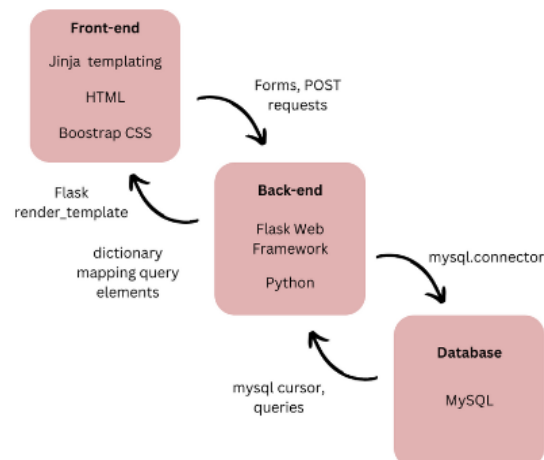
- Performance:
  - The application should respond quickly to user requests, with minimal latency.
  - The application should be able to handle a large number of concurrent users without performance degradation.
- Security:
  - Data encryption for sensitive information.
  - Compliance with relevant data protection regulations.
- Scalability:

- The application architecture should support growth in users and data volume.

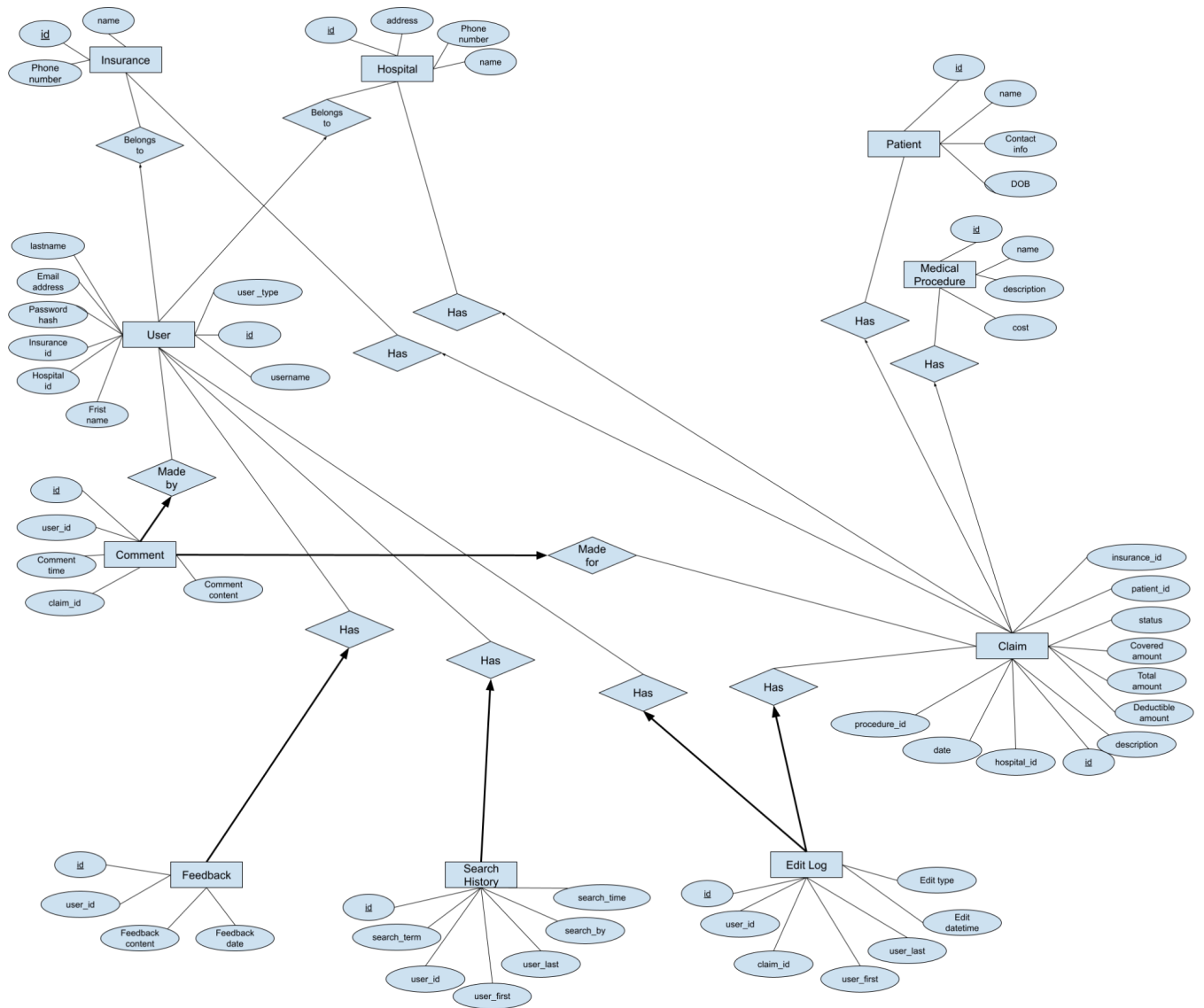
### Architecture:

This application was developed using a 3-tier architecture.

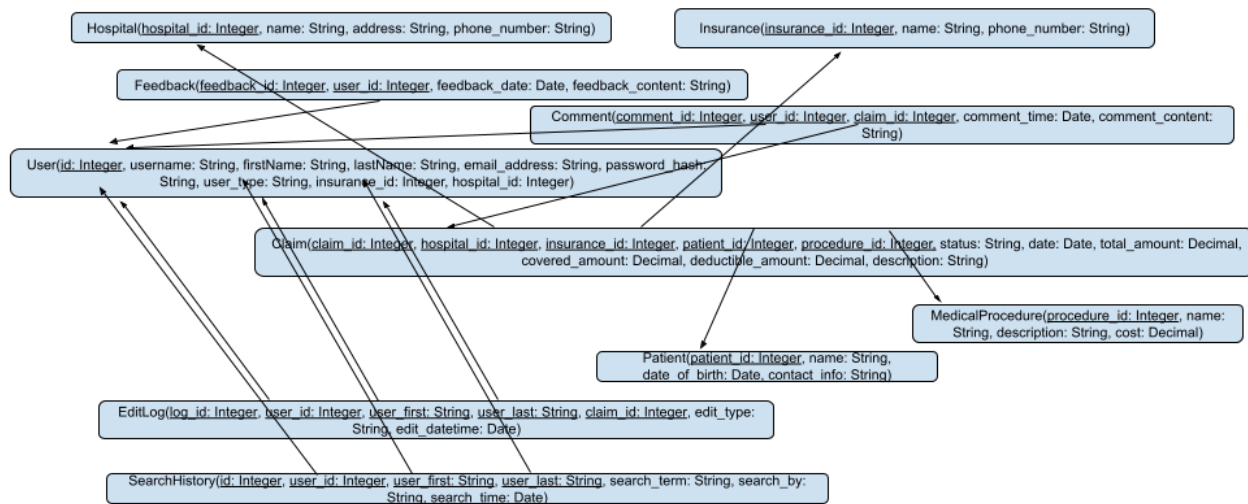
- Front-end: HTML, Bootstrap CSS and Jinja
  - This was where all the logic was written, how we queried the database, and how the result set was passed to the front-end.
- Back-end: Flask, Python
  - Queries were used by the backend to query the database and get the required data.



## Entity Relationship Diagram



## Database Design



## Normalization

Insurance(insurance\_id: Integer, name: String, phone\_number: String)

→ I(A, B, C) and FD's: (A → BC)

Hospital(hospital\_id: Integer, name: String, address: String, phone\_number: String)

→ H(A, B, C, D) and FD's: (A → BCD)

User(id: Integer, username: String, firstName: String, lastName: String, email\_address: String, password\_hash: String, user\_type: String, insurance\_id: Integer, hospital\_id: Integer)

→ U(A, B, C, D, E, F, G, H, I) and FD's: (A → BCDEFGHI, B → ACDEFGHI, E → ABCDEFGHI)

Modify User table to split since superkeys are id, username, email\_address

UserInsurance Table (Primary Key: user\_id, insurance\_id)

user\_id (Foreign Key referencing User(id))

insurance\_id (Foreign Key referencing Insurance(insurance\_id))

UserHospital Table (Primary Key: user\_id, hospital\_id)

user\_id (Foreign Key referencing User(id))

hospital\_id (Foreign Key referencing Hospital(hospital\_id))

Patient(patient\_id: Integer, name: String, date\_of\_birth: Date, contact\_info: String)

→ P(A, B, C, D) and FD's: (A → BCD)

MedicalProcedure(procedure\_id: Integer, name: String, description: String, cost: Decimal)

→ M(A, B, C, D)

Claim(claim\_id: Integer, hospital\_id: Integer, insurance\_id: Integer, patient\_id: Integer, procedure\_id: Integer, status: String, date: Date, total\_amount: Decimal, covered\_amount: Decimal, deductible\_amount: Decimal, description: String)

C(A, B, C, D, E, F, G, H, I, J)

SearchHistory(id: Integer, user\_id: Integer, user\_first: String, user\_last: String, search\_term: String, search\_by: String, search\_time: Date)

Comment(comment\_id: Integer, user\_id: Integer, claim\_id: Integer, comment\_time: Date, comment\_content: String)

Feedback(feedback\_id: Integer, user\_id: Integer, feedback\_date: Date, feedback\_content: String)

EditLog(log\_id: Integer, user\_id: Integer, user\_first: String, user\_last: String, claim\_id: Integer, edit\_type: String, edit\_datetime: Date)

The four above tables were fairly simple and had no functional dependencies other than the primary key being able to determine the others. They followed BCNF format and did not require any changes

### Verification of BCNF

**1NF** - each attribute value is a single atomic value from its domain.

This is verified in the above schema because no column in any of the tables has composite or multiple values.

**2NF** - 1NF is valid, and all non-candidate key attributes are fully functionally dependent on a candidate key if there is any. Most of our tables do not depend on any candidate key.

**3NF** - No non-candidate-key attribute is transitively dependent on a candidate key.

Initially, under the "claim" table, we had total\_amount, covered\_amount, and a derived attribute: deductible\_amount. This was later changed and the derived attribute was derived in the routes rather than stored in the database.

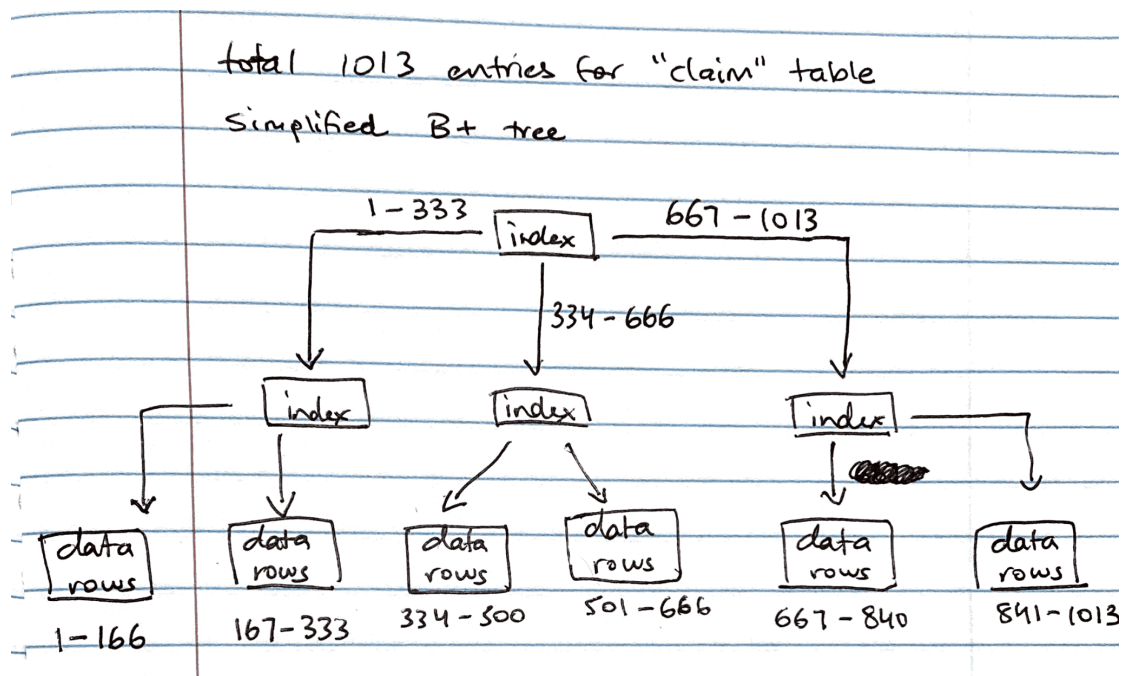
**BCNF** - The closure for each functional dependency was checked to ensure it produced

the entire relation. This verified BCNF. Finally, we made sure no functional dependencies were lost. After making the appropriate changes, we confirmed our schema followed BCNF.

## Indexing

While indexing is automatically done by MySQL for primary and foreign keys; we implement an additional index for certain attributes in the “Claim” table. Our Claim table contains over 1000 entries, and therefore a fast and efficient querying is required. To implement this, we added the following attributes in index:

```
INDEX (hospital_id),  
INDEX (insurance_id),  
INDEX (patient_id),  
INDEX (procedure_id),  
INDEX (status),  
INDEX (date)
```





Above a simplified B+ tree can be seen that shows how the data entries are split up into indexes. This tree is the same case for all above-mentioned attributes since they all have the same (1013) entries.

## **Major design decisions**

### Database

Logs should be maintained even after a user mentioned in the log deletes their profile. To accomplish this, we had to remove the foreign key relation.

Additionally, we implemented ACID properties and transactions in many areas. When creating an insurance/hospital only one could be written at a time since we used: “LAST\_INSERT\_ID()”. We did this by turning off auto commits, and placing the logic in a try block. This ensured that no other commits would take place during that timeframe, and also ensured that if anything were to go wrong during that block, the transaction would revert back to its original state.

### Security and Confidentiality

View and access control is one of our primary design features. Hospitals will only be able to see and create claims for their hospital. Insurance providers will only be able to see claims created for them.

Additionally, we have features in place to capture and log any edits made to the claims by users. We also have a log of searches conducted by users. This is visible only to the admin user.

## Implementation details

As mentioned in our architecture above, we used a 3-tier architecture with MySQL for the database, Flask and Python for the backend, and HTML/CSS for the frontend. To dive deeper into this implementation, let's talk about the specific structure of the code. Our code was broken up into 6 smaller sections: html templates (frontend), forms (connection between frontend and backend), initialization and run code, model (current user session), routes (all logic and functionality for each route), and finally, the database which included create, insert, and drop statements.

While there are many pages and routes, we can focus on the claim page route primarily as it was the largest and most difficult to implement. This route starts by fetching all the claims for the user depending on their user\_type (insurance or hospital). Then there are additional queries in place to collect information about the claim that isn't explicitly given in the claim schema. For example, the claim table has all the ids, patient\_id, procedure\_id etc, however it does not have the actual names and other information about those ids since that would violate BCNF. Therefore, additional queries were used to get that information. Following that, the id's would be mapped to the respective information using a dictionary and that would be passed to the frontend. Searching was also troublesome because it involved passing the search results through a session, and overriding the claim dictionary with the search result set. Finally, this page also handled all the POST requests, such as delete, update status, add comment, and more.

The dictionary that was passed to the frontend could be accessed by using Jinja. For example, to get the hospital name of a particular claim, the hospital names dictionary was accessed with the hospital\_id of the claim (which is found at the 2nd column of the claim table (claim[1])). This was done as such:

```
Hospital: {{ hospital_names_dict[claim[1]] }}
```

## Demonstration of example system run

### 1. Login Page

- Old users can log in or new users can choose to sign up.

The screenshot shows the 'Log In' page of 'TheClaimHub'. The header includes the site name, a search bar, a 'Hospital' dropdown menu, and links for 'Login' and 'Sign Up'. The main content area features a dark grey login form with the title 'Log In'. It contains two input fields labeled 'Username' and 'Password', followed by a 'Log In' button. A link for 'Sign up' is provided for users who do not have an account.

## 2. SignUp Page

- Users are supposed to choose First Name, Last Name, Username, Email, Password and their account type (Insurance Provider or Hospital).

The screenshot shows the 'Sign Up' page of 'TheClaimHub'. The header is identical to the login page. The main content area features a dark grey sign-up form with the title 'Sign Up'. It includes input fields for 'First Name', 'Last Name', 'Username', 'Email', 'Password', and 'Confirm Password'. Below these fields, there is a section titled 'Select one of the following:' with two radio button options: 'Insurance Provider' and 'Hospital'. A 'Submit' button is located at the bottom of the form.

- Insurance Provider users got to choose the company they represent.

The screenshot shows a web application interface for 'TheClaimHub'. At the top, there is a dark header bar containing the site name 'TheClaimHub', a search input field, a 'Hospital' dropdown menu, and 'Login' and 'Sign Up' links. The main content area has a light blue background. In the center, a dark gray rounded rectangle contains the form titled 'Select Your Provider'. The form asks 'Which insurance company do you represent?' and features a dropdown menu with 'Kaiser' selected. Below the dropdown is a 'Submit' button. At the bottom of the form, there is a link that says 'Don't see your insurance company? [Add a new one!](#)'.

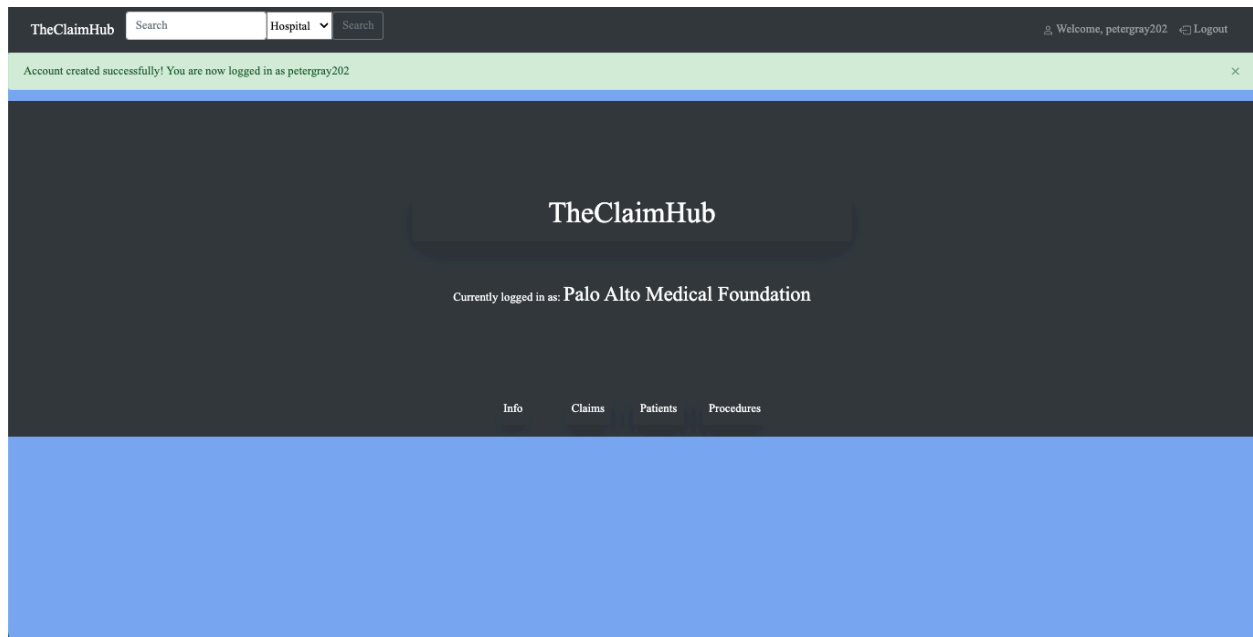
- Hospital users got to choose the hospital they represent.

This screenshot shows the same 'Select Your Provider' form as above, but with the dropdown menu set to 'Palo Alto Medical Foundation'. The rest of the interface, including the header and footer, remains identical.

### 3. Home Page

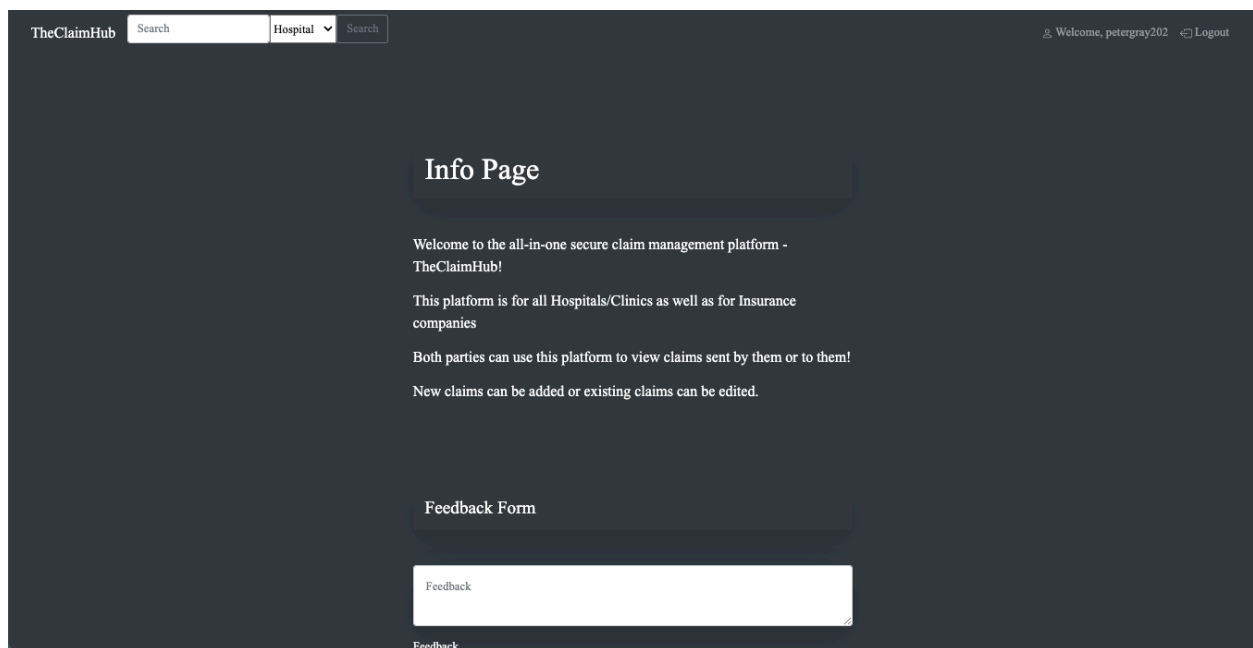
- The first page users will see once they log in.

- Every essential function can be redirected from this page.



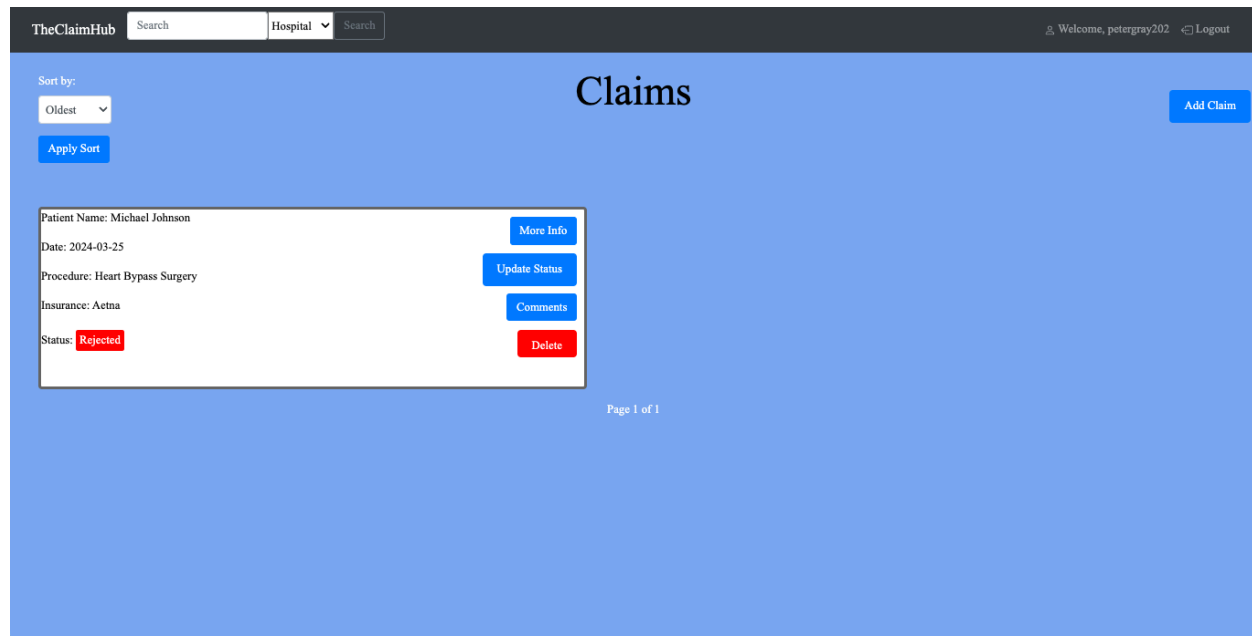
#### 4. Info Page

- The information about the application can be read here.
- Users can send feedback by filling up the feedback form.

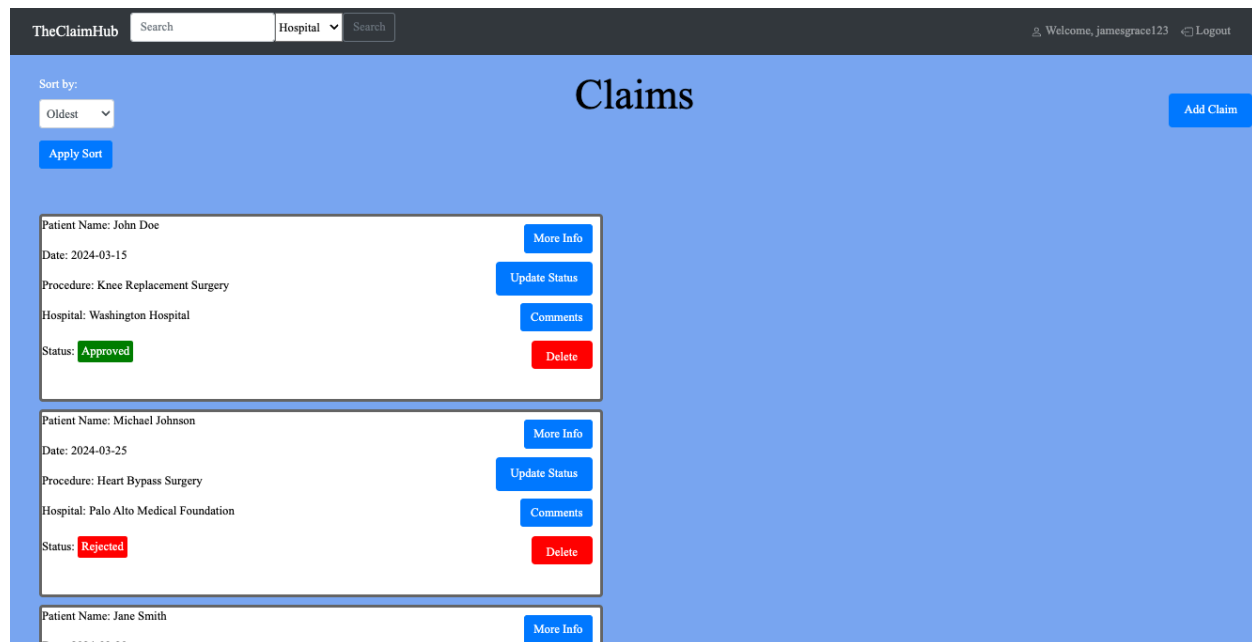


## 5. Claims Page

- Every claim can be found on this page.
- Hospital users can create new claims and edit the old ones.



- Insurance Provider users can update the status of claims created by hospitals.



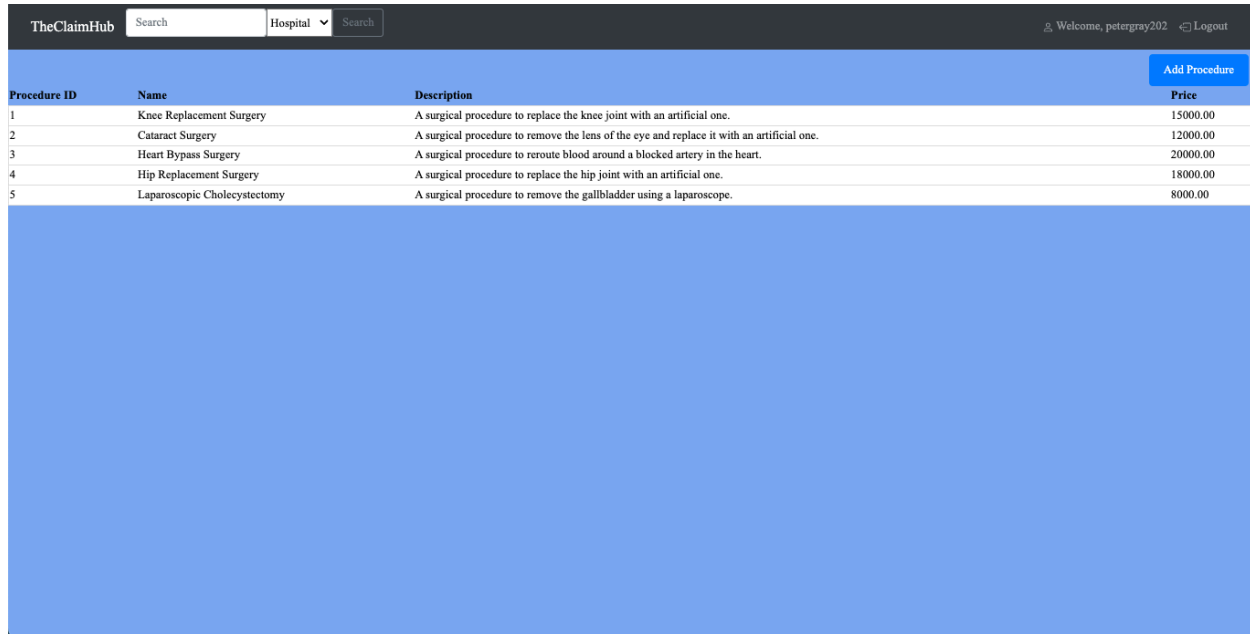
## 6. Patients Page

- All patient information can be found here.
- New patients can be created.

TheClaimHub			
Search		Hospital	Search
			Welcome, petergray202 Logout
Patient ID	Patient Name	Date of Birth	Contact
1	John Doe	1990-05-15	john.doe@example.com
2	Jane Smith	1985-08-20	jane.smith@example.com
3	Michael Johnson	1978-12-10	michael.johnson@example.com

## 7. Procedure Page

- Procedures information are displayed here.
- New procedures can be created.



The screenshot displays the 'TheClaimHub' web application. At the top, there is a dark navigation bar with the site name, a search input field, a 'Hospital' dropdown menu, and a user login area showing 'Welcome, petergray202' and a 'Logout' link. Below the navigation bar is a table with four columns: 'Procedure ID', 'Name', 'Description', and 'Price'. The table lists five surgical procedures. A blue 'Add Procedure' button is located in the top right corner of the table area. Below the table, there is a large blue rectangular area, likely a placeholder for more content or a full-screen view.

Procedure ID	Name	Description	Price
1	Knee Replacement Surgery	A surgical procedure to replace the knee joint with an artificial one.	15000.00
2	Cataract Surgery	A surgical procedure to remove the lens of the eye and replace it with an artificial one.	12000.00
3	Heart Bypass Surgery	A surgical procedure to reroute blood around a blocked artery in the heart.	20000.00
4	Hip Replacement Surgery	A surgical procedure to replace the hip joint with an artificial one.	18000.00
5	Laparoscopic Cholecystectomy	A surgical procedure to remove the gallbladder using a laparoscope.	8000.00

## Challenges

The primary challenges we came across were designing the application to work for both hospitals and insurance companies, and maintaining user sessions. In order for the application to be suitable for both user types, we had to ensure we had proper view control, as well as design an appropriate database that worked regardless of the user type. Maintaining user sessions was also difficult, especially the signup page which contained several pages of getting the insurance/hospital name etc. before the user was actually signed up. Passing all the information from one page to another using sessions was difficult.

## Conclusion

To conclude, this project aimed to address the inefficiencies in insurance claim management for small to medium-size clinics by developing a centralized Insurance Claim Management Hub. Through a web-based platform with user authentication, claim submission and management functionalities, we created a robust system that streamlines the claim submission process for clinics and provides insurance providers with easy access to view process claims.



## **Lessons learned**

Designing the database was definitely the biggest lesson learned. It involved a significant amount of refining and testing. It was challenging because we did not have a clear vision of our platform, and therefore as we went on adding new features, we had to change the database structure accordingly. The lesson learned was to have a thorough plan of the platform and build the database first. Once that is done, stick to the plan and avoid implementing features that could break the database structure.

Another lesson learned that kind of ties into the previous one, is to account for scalability when designing the project. Initially, we had very few claims in our database, therefore the claim page did not cause many problems. But once we populated our database with 1000+ claims, it made the claim page super long. To combat this we had to implement pagination which caused even more problems later down the line.

## **Possible Improvements**

Claims offer a much greater selection of options on these existing claim submittal platforms. One possible and necessary improvement would be to account for these and let insurance company users interact more with their claims. This could include paying the claim off, or passing it to a third party auditor. Currently we only allow for changing status, adding, deleting, and commenting claims, but more should definitely be considered.

Another possible improvement would be considering our platform more like a database than a portal/hub. This would mean allowing for mass creations/changes in data. For example, if all claims for a particular hospital need to be changed a certain way, it would be difficult to do so on our current platform. This would also bring things such as exporting all particular data to an excel spreadsheet or some other document.

The biggest possible improvement would be to add yet another user type: Patients. Patients tend to have no say or knowledge of their medical claims. This is something typically handled

between the hospital and the insurance. However, we could possibly add them as a user type to let them view their claims. This could be helpful as they could catch any errors or mistakes in their claims. We could also add a feature allowing patients to directly contact their clinics or insurance companies.

Finally, a significant change to UI/UX is also a necessary improvement that needs to be made to ensure the success and likeability of our platform.

**Future Challenges:**

Challenges we may come across are primarily data security and data scalability. Working with sensitive healthcare data requires us to be attentive to any potential security related issues that may arise. We will follow up-to-date security protocols and ensure that there is never a data leakage or attack.

Data scalability is also our top priority. Health insurance claims even for one provider can be in the millions. Transactions, and updates to our database happening every second is something we must accommodate for. Additionally, our data querying methods must be highly efficient and fast as the large number of claims can exhaust our resources.