Notified

Goose River Technologies
Sponsored by: Duck Creek Technologies

Manager: Sameen Shaik

Developers: Din Masic, Hassan Shah, James Piñeiro, Conor Deely, Andre Gilbertson, Nhat Minh Le, and Raymond Huang

Table Of Contents

1. What is Notified?

1.1. Description:

1.1.1. Notified is the home of all of your Duck Creek Technology insurance claims, policies, and news communications.

1.1.2. It consolidates messages from customers, Duck Creek employees, and third parties (such as claims, examiners, and their companies) into one unified interface.

1.1.3. With Notified, users no longer need to check multiple platforms or pages to track important updates—it brings all relevant notifications into a single, organized feed that is managed by the company and not by third-party software.

1.2. Types of Users:

1.2.1. Customer: The customer is a user who needs to check their insurance claims/policies/ and news. They should be able to reply and send messages to employees based on what inquiries they have about the insurance.
1.2.2. Employee: The employee is a user who sends and manages messages to the customers and answers questions/concerns.

1.3. Security:

1.3.1. Envelope: Used to mask our database login information. This keeps our admin login private and inaccessible to anyone who isn't us.
1.3.2. Encryption: Used encryption to encrypt the JWT for the current user.
1.3.3. ID Generation: Upon account creation, a very long unique ID is generated for each user, making it nearly impossible to guess.
1.3.4. SQL Injection: Since we query for everything, it is impossible to SQL Inject into our text boxes, which we have checked.

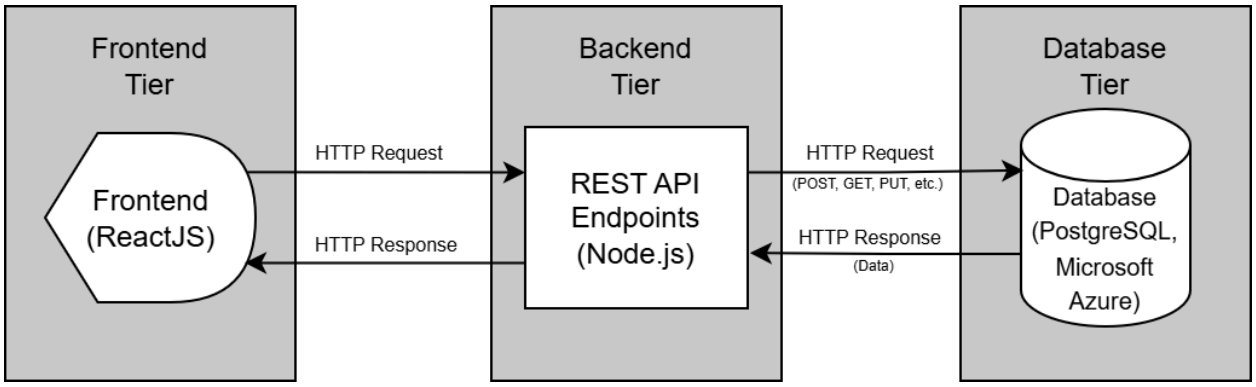
1.4. Use Case:

1.4.1. Employee: An employee needs to send an insurance claim message to a customer filing a claim by attaching a file for the claim.
1.4.2. Customer: The customer receives the message and finds it in the inbox through the claims tab, in which he notices a specific detail in the file in which he needs to send a reply so he can ask about the section in the file..
1.4.3. Employee: The employee then replies back to that reply and answers the customer's misunderstanding.

2. Third Party Software:

Software Title	Description	Purpose
Microsoft Azure	A cloud computing platform by Microsoft that provides services for creating, deploying, and managing applications and infrastructure.	Database cloud hosting for frontend access
pgAdmin	A graphical administration and development tool designed to manage PostgreSQL databases.	Creation, modification, and visualization
Postman	An API development environment that lets users design, test, and document HTTP endpoints.	Testing API endpoints and requests
Node.js	A JavaScript runtime enabling event-driven, non-blocking I/O for scalable network applications.	Hosting backend server, endpoint connection
ReactJS	A JavaScript library for building component-based frontend user interfaces for web apps.	Frontend interaction and visualization of data in app
Figma	Cloud-based graphics editor and prototyping tool that brings parties together on a unified, shareable canvas.	Facilitated a collaborative design, prototype, and iteration on the frontend user interface.

3. Architecture Diagram:



4. Project Difficulties:

4.1. Understanding Client Expectations:

One of the first and most significant challenges was achieving alignment with the client on what the product should accomplish. Early in the project, there was some confusion regarding user roles, such as who should initiate messages, what types of notifications should be visible to different users, and how the system should be managed overall. To address this, we communicated with the client to clarify requirements and ensure our design decisions matched their expectations.

4.2. Learning New Technologies:

Choosing the right technologies introduced another set of challenges. One of our biggest technical hurdles was selecting and setting up the database. After exploring several options, we chose PostgreSQL because it is an open-source database system capable of handling complex data. However, due to our team's limited experience, setting up and integrating PostgreSQL took longer than expected.

Deployment also posed challenges, as many cloud platforms had limitations on student accounts. After researching various solutions, we chose Microsoft Azure, which met our hosting needs but required additional setup and learning time. Additionally, since not all team members were familiar with Node.js, extra effort was needed to onboard and support those with less experience.

4.3. Role Designation:

We also faced difficulties with team role distribution. Initially, the database development team was understaffed, which delayed progress on setting up the database. To resolve this, we temporarily reassigned team members from other areas to help with database development. This restructuring allowed us to catch up on lost time and align the database team's progress with the rest of the project.

4.4. Balancing Time Commitments:

Finally, balancing academic and personal responsibilities alongside project deadlines was a constant challenge. As a team of seven students, we had to manage coursework, exams, and other commitments. To stay on track, we organized additional meetings outside of scheduled sessions to collaborate when team members were available. This flexibility and extra effort were critical to keeping the project moving forward.

5. What Worked Well:

5.1. Communication

To maximize our output as both individuals and a group, we had to ensure spotless communication. By knowing the ins and outs of each other's availability, we were able to host multiple extra meetings throughout the week in order to maximize productivity. In order to keep Notified as close to the desired product as possible, we also made sure to reach out to our client, Jeff, for any lingering questions. When we were having trouble on the database side of things, our extraordinary communication skills allowed us to schedule three extra meetings throughout the week to get our project back on track.

5.2. Organization

In order to prevent a messy codebase and unsustainable code, we made sure to create plenty of separate branches for each section of our project. We created a frontend branch, backend branch, and separate branches within those branches for each team member. As a result, splitting up our tasks into different branches allowed us to create our assigned portions of the project without merge conflicts.

5.3. Integration

Once we had finished our individual tasks and were ready to move onto integration, our organization and communication allowed us to integrate seamlessly. Because of our extra meeting times and efficiently-ordered GitHub, it only took us a few commits and two meetings to merge our codebase together.

6. Technical Issues and Testing:

6.1. Choosing and learning new technologies

One of the first issues we encountered was choosing what database management system we should use. We originally used postgreSQL, debated between switching to Firebase, and then eventually decided to stay with PostgreSQL. Our team's primary issue here was analysis paralysis, as we spent about 1-2 weeks deciding which program we should go with instead of sticking with .

6.2. Integration

After the Database and backend were created, we had an issue with integration. Due to the lack of standard naming conventions, there were a lot of errors and extra time spent standardizing the naming convention. This could have been avoided if we planned it out beforehand, but it ultimately improved our communication skills.

6.3. White Screen issues

Another issue we ran into was a white screen appearing when running PGAdmin when we were trying to integrate everything together. When opening the object manager, it would white screen and we were not able to see any of the data. To solve this issue we changed computers, but it had unfortunately already slowed down two of our group members.

6.4 Testing

For testing we used Postman, which we used to see if our API returned the correct object. For the front end, we ran it multiple times to ensure it looked professional from a user perspective. We also changed the window size to make sure everything was functioning correctly. As for other types of user testing, we also performed bulletproofing. In this bulletproofing, we ensured that each password had to have a certain number of characters, and ensured that each account had an associated email address.

7. How To Run Notified:

To run the Notified application locally, follow the steps below. The application consists of a backend server and a frontend client, and requires access to a PostgreSQL database.
Link to GitHub: https://github.com/sshaik03/GooseRiverTechnologies

7.1. Prerequisites

Ensure you have the following installed on your system:

- Node.js (v14 or later)
- npm
- A PostgreSQL-compatible environment or connection to our hosted database

7.2. Clone the R=epository

```
git clone <repository-url>
cd notified
```

7.3. Set Up Environment Variables

Create a `.env` file in the root directory of the project and add the following database credentials.

```
PGHOST=notified-datahub-server.postgres.database.azure.com
PGUSER=notified_db_admin
PGPASSWORD=Duckcreek2025
PGDATABASE=notified_db
PGPORT=5432
```

Note: These credentials allow the backend to connect to the hosted PostgreSQL database on Azure.

7.4. Start the Application

You'll need two terminals for running the backend and frontend concurrently:

Terminal 1: Start the Backend

```
node app.js
```

This command runs the Express.js backend server.

Terminal 2: Start the Frontend

```
cd client
npm install
npm start
```

This starts the React frontend and opens it in your default browser.