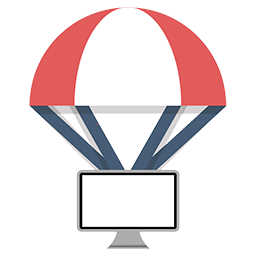
Dropzone HQ

****

**Design Document**

# 

[**Team**](#_xbdmgrmcoqqh) **2**

[Members](#_28qwvvhw072c) 2

[Resource Links](#_nhyo0yrjysz) 2

[**Project Scope**](#_nsdr9kfjxhb2) **2**

[Currently Available Services](#_dzc6pnw7etua) 2

[Dropzone HQ’s Key Differences from Other Services](#_h71ijns74lmw) 3

[**Key Project Technologies & Frameworks**](#_34s8v9tqxxnk) **3**

[Frontend](#_w4on6awx78ok) 3

[Backend](#_17cy5ei39sos) 3

[**User Interface**](#_cw4bg0mzp6ye) **3**

[Login Screen](#_csf1un2u5mjb) 4

[Main Screen](#_c33170lrrzt0) 5

[Employee Screen](#_11w45zhdltcy) 6

[Inventory Screen](#_iwtmqcuukhq6) 7

[Tandem/Student Rig Sheets Screen](#_jx78qycb8st1) 8

[Loft Screen](#_vk4ig71fcd9x) 10

[Rental Screen](#_thvfyxemw0rz) 11

[**Database**](#_v571txzbmzwo) **13**

[EER Diagram](#_9suao3r99vov) 13

[Relationships](#_mwvcrm8yurjf) 13

[Items](#_tm2mi29e3f7x) 13

[Employees](#_fq4xi694v0q2) 13

[Rigs](#_5f1entqs6sid) 14

[Triggers](#_d36mw4fnjaa) 14

[Views of the Data](#_7owd9nzbewz9) 14

[Possible Future Views](#_utrukpybms4t) 15

[Design Choices](#_106mrsuhlhj7) 15

[**API Endpoints**](#_ctlch1byd5x3) **17**

[**Glossary**](#_c4n80hw2tx0d) **19**

# Team

## Members

Paul Bayruns

Jatin Bhakta

Andres Blotta

Brian Krick

Jon Spratt

Paul Turner

## Resource Links

[**paulbayruns.com/dropzone-HQ.html**](http://paulbayruns.com/dropzone-HQ.html)

[**GitHub**](https://github.com/sprattj/Senior-Project)

[**ER Diagram**](https://drive.google.com/file/d/1BAyWk755B8WrV6QJuz7GqGFF8hgF3DDj/view?usp=sharing)

# Project Scope

## Currently Available Services

Jobs at skydiving dropzones can be split into a few basic groups.

On the customer side, employees called Manifesters deal with payment processing and scheduling for recreational or instructional skydives. This scheduling process involves organizing which skydivers are going on which planes throughout the day. The record of these jumps is called the manifest, hence their job title.

Recently, a company named Burble built a web app to organize and streamline the manifesting process. One section of Burble’s service has functionality for creating/organizing plane loads via simple dragging and dropping, after which the plane loads are then visible to customers via a webpage or Burble’s mobile app. This mobile app also offers customers the ability to sign up for sky dives and maintain a user profile, allowing them to sign up for sky dives more easily. This signup process also handles payment for the dropzones, offering integration with various payment and bookkeeping services.

## Dropzone HQ’s Key Differences from Other Services

Despite handling the front end of the business effectively, Burble does not manage any of the other employee roles at a dropzone. A large percentage of dropzone employees do not directly deal with payments or scheduling. Instructors, riggers, and packers handle most of the processes that go into making a jump happen. When an instructor takes a parachute rig out for a jump with a student, they need to record which rig they used. Packers, who repack the parachutes after people come back in from a jump, need to know what parachute they are packing so they can verify that they packed it. When gear gets damaged or needs maintenance, riggers are the one who are responsible for repairing it, as well as making sure all gear is up to government defined standards. Jumps, repacking, rentals, and repairs all need to be managed, and Burble does not offer solutions for any of them.

Dropzone HQ fills in what is missing from Burble by handling the internals of the dropzone that customers don’t directly interact with. By managing the employee related jump records, providing a rental system, and logging information on all the gear of the dropzone, DZHQ can modernize many aspects of several jobs that Burble does not support.

# Key Project Technologies & Frameworks

## Frontend

* ***ReactJS*** - A javascript library made by Facebook, serving as our primary frontend library. React was chosen because it enables the creation of modular, widely reusable code, and can quickly update HTML DOM components, even when dealing with large datasets. Also, supports easy transfer to mobile apps via ReactNative, without the need to write native code (Swift or Java).
  + **Reactstrap** - An implementation of Bootstrap for ReactJS. Allows for easier creation of clean, responsive ReactJS components and pages.
  + **React Router** - A router library for ReactJS that manages component loading/changing based on the URL. In React, pages are not switched in the same way as in HTML, rather the same HTML file has its content changed by React. React Router manages the mapping of urls to changes of page content.

## Backend

* ***Django*** - A python web framework that was built to power complex data-driven websites. Chosen for speed, modularity, and dev team experience/interest in python.
* ***MySQL -*** Widely used database language chosen for speed, reliability, and team familiarity.
* ***AWS*** - Webserver and database hosting
* ***Rest API*** - Standard web practice, build an API that can be called to handle communication between the frontend and backend/db

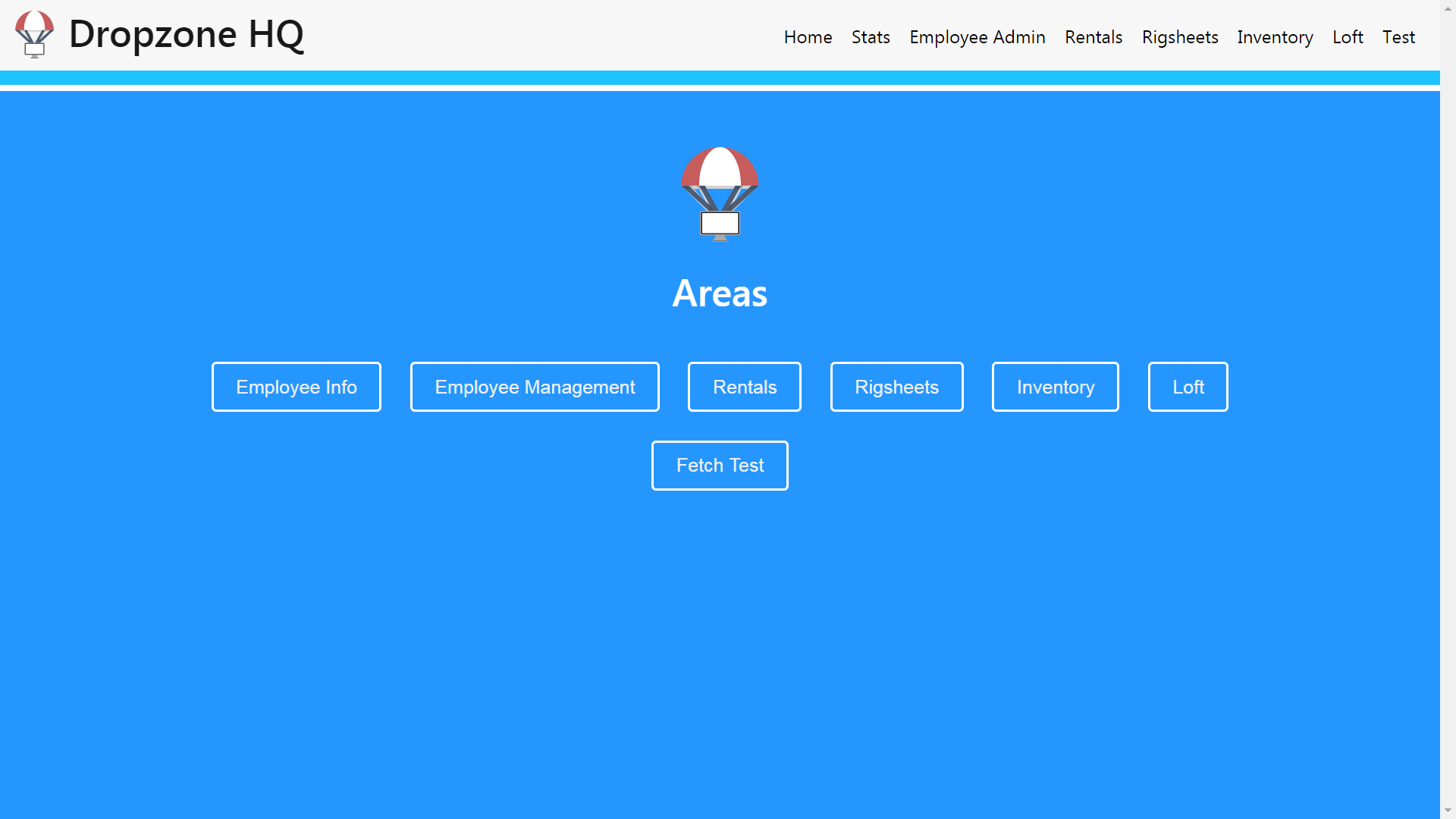
# User Interface

## Login Screen

The Login screen is the outer layer of the website. Each dropzone will be required to enter a unique email and password for their dropzone. This login information will be stored in the Dropzone table. When a user attempts to log into the system with an email and a password, it will authenticate and allow users access to the system for their dropzone if they entered valid information.

*Login Screen DB Communication - submit button to check dropzone email and password*

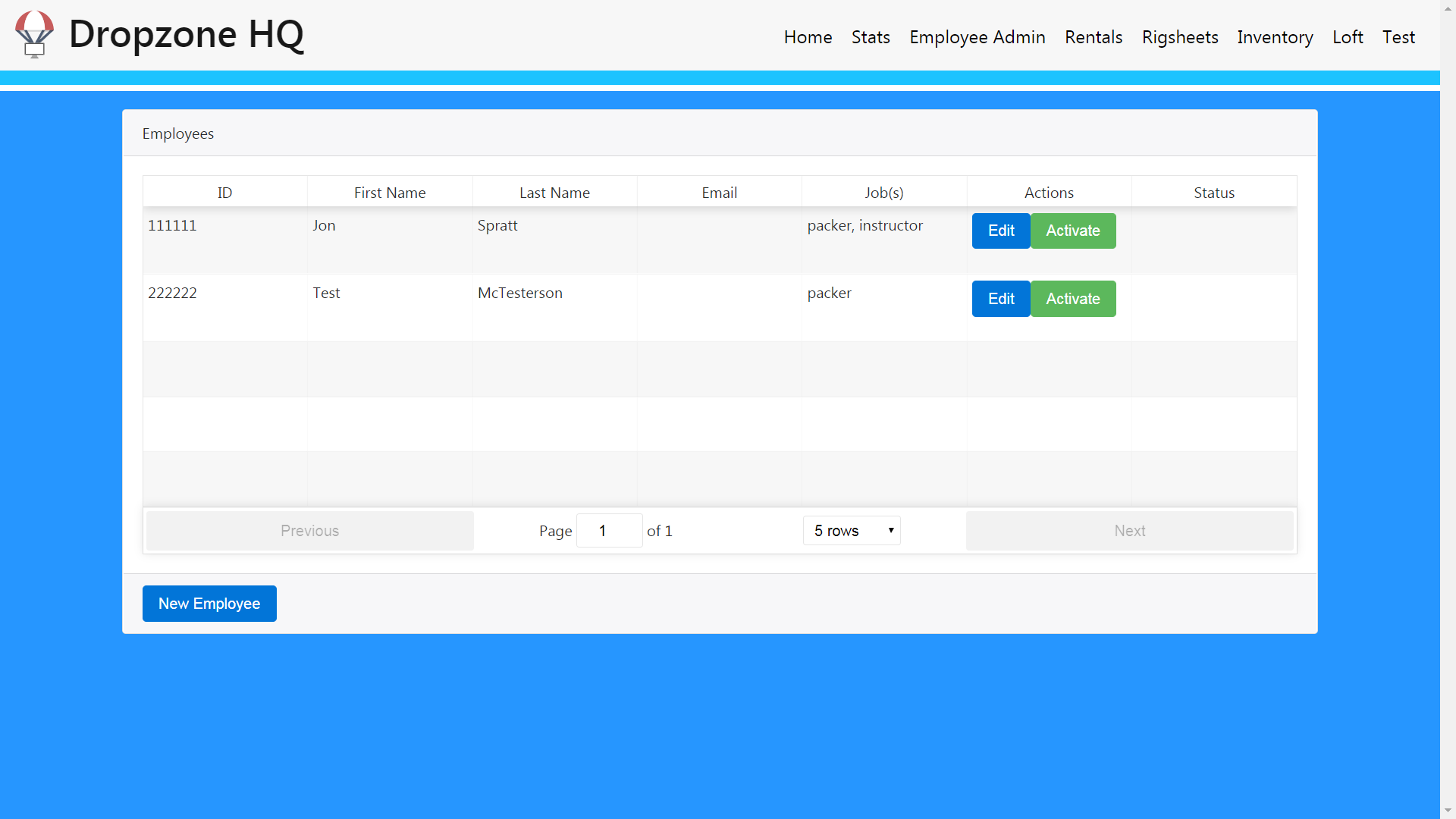
## Main Screen



The Main screen is the hub for all the screens and is where the users are brought after login. Each of the buttons leads to a different screen of the system, although some require verification before entering such as the Employee screen, the Inventory screen, the Loft screen, the Statistics screen, and the Reports screen.

*Main Screen DB Communication: verify user privileges before accessing certain screens*

## Employee Screen



The user will have to verify their PIN when attempting to access this page, which will allow the system to restrict what they see when entering. Unless the verified user is an Admin, the employee should only see their own information on this page. The Admin will have a full list of their current employees to modify as they please. The Employee screen will allow the users to add, remove, or change the info of employees working for that Dropzone, except for pin which is semi-randomly generated. If the Employee would like to change their pin they can and a new one will be emailed to them with the email that they have on record. When creating an employee the Admin will have a selection of jobs the employee would have including: Rigger, Loft Head, Loft Employee, Tandem Instructor, AFP Instructor, Packer, Manifest, Videographer, Hanger Master. The employees will input their pin individually for verification of each of their actions in the system. The employee's pin wouldn’t be used to log into anything, but instead for confirming their digital signature later, ex. Instructor walks up to sign out a parachute, chooses the parachute, inputs their pin to tell the system they are signing it out, system requests their pin for confirmation, correct input allows for the rig to be signed out while incorrect input will ask again or restart the process. This would prevent malicious users from signing things out in someone else's name. Employees would be able to change their pin from this screen where they would be sent an email included in it would be their new Employee pin.

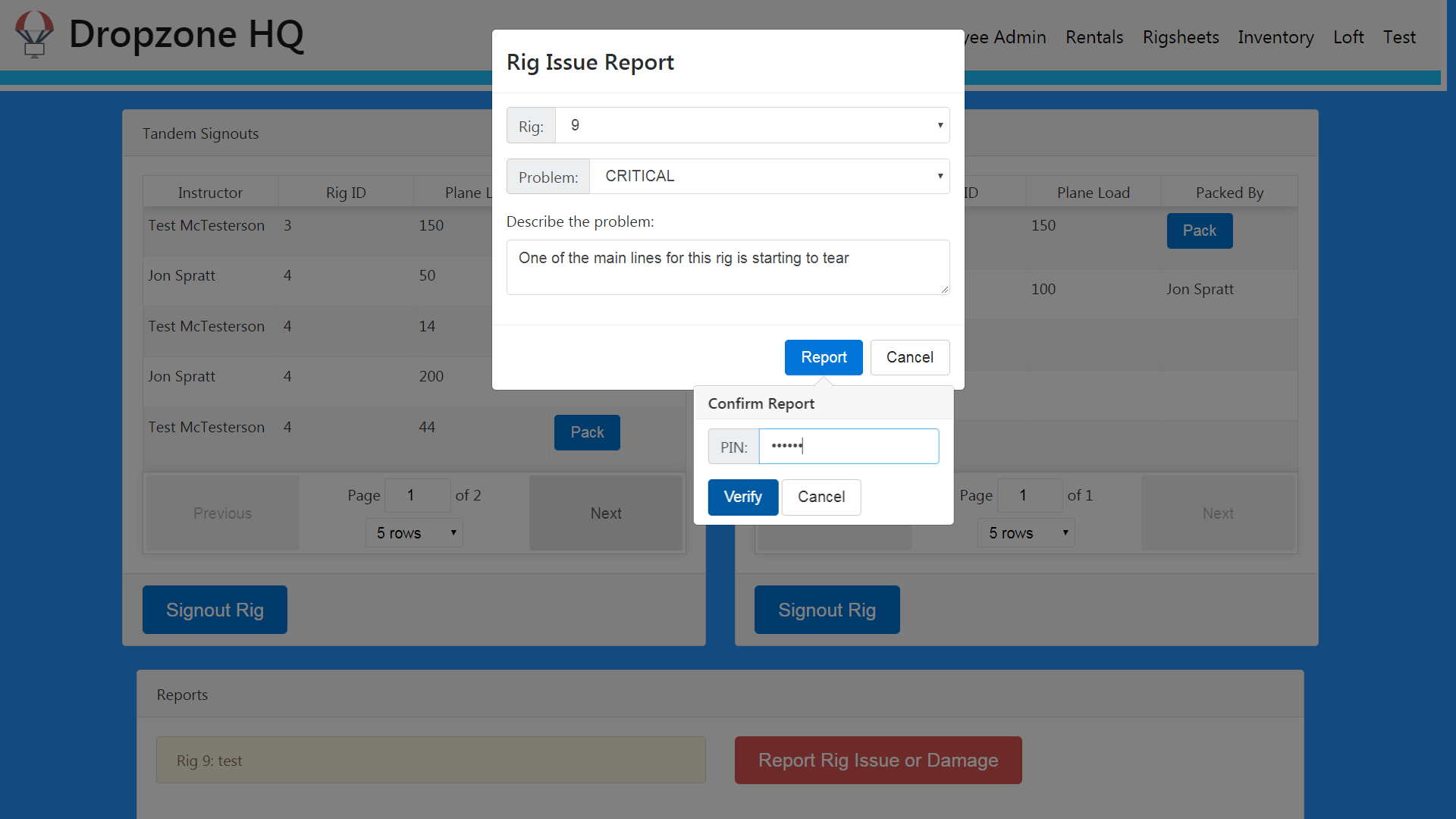
*Employee Screen DB Communication: add Employee, remove Employee, modify Employee info, show single Employee info, show every Employee’s info*

## Inventory Screen

The user will have to verify their PIN when attempting to access the Inventory screen before entering. If the verified user is not an Admin, Loft Head, or Loft Employee, they will be rejected and ask for the verification to be entered again. The Inventory screen will allow users to add, remove, and edit inventory items such as canopies, containers, AADs, tandem rigs, and student rigs. It can also make a report to the Loft Screen that there is a problem with a rig. On the left side of the screen labeled Item List will be a scrollable section with all the current equipment in the system, which can be filtered by selecting some presets in a dropdown like, “Rigs Only,” or, “Main Canopies Only,” and so on. Since our item pool is limited to the items listed above, we can create a separate view for each item type which can be either empty when “Add New Item” is selected or full of relevant data when an item is selected from the item list which would be displayed on the right side in the panel labeled Item Details.

*Inventory Screen DB Communication: add Item, edit Item, remove Item, show all Items in list, show single Item details, add Warning*

## Tandem/Student Rig Sheets Screen



The Tandem/Student Rig Sheets screen will allow employees to actively check out and check in equipment available to them from the Inventory screen. The left side of the screen will show the tandem log and the right shows the student log. An instructor will select the “Add” button on the log they want and fill out their information before verifying it. Once verified the information is added to the log with a button on the end allowing a Packer to sign the rig back in after it’s packed for reuse or report a malfunction which takes the rig offline. The Packer will verify their PIN and once verified, the log is complete. Any completed logs can be undone by the Packer that signed that row or a Rigger. The logs on either side will only show logs for the current day. This screen also allows for reporting problems with rigs to be sent to the Loft.

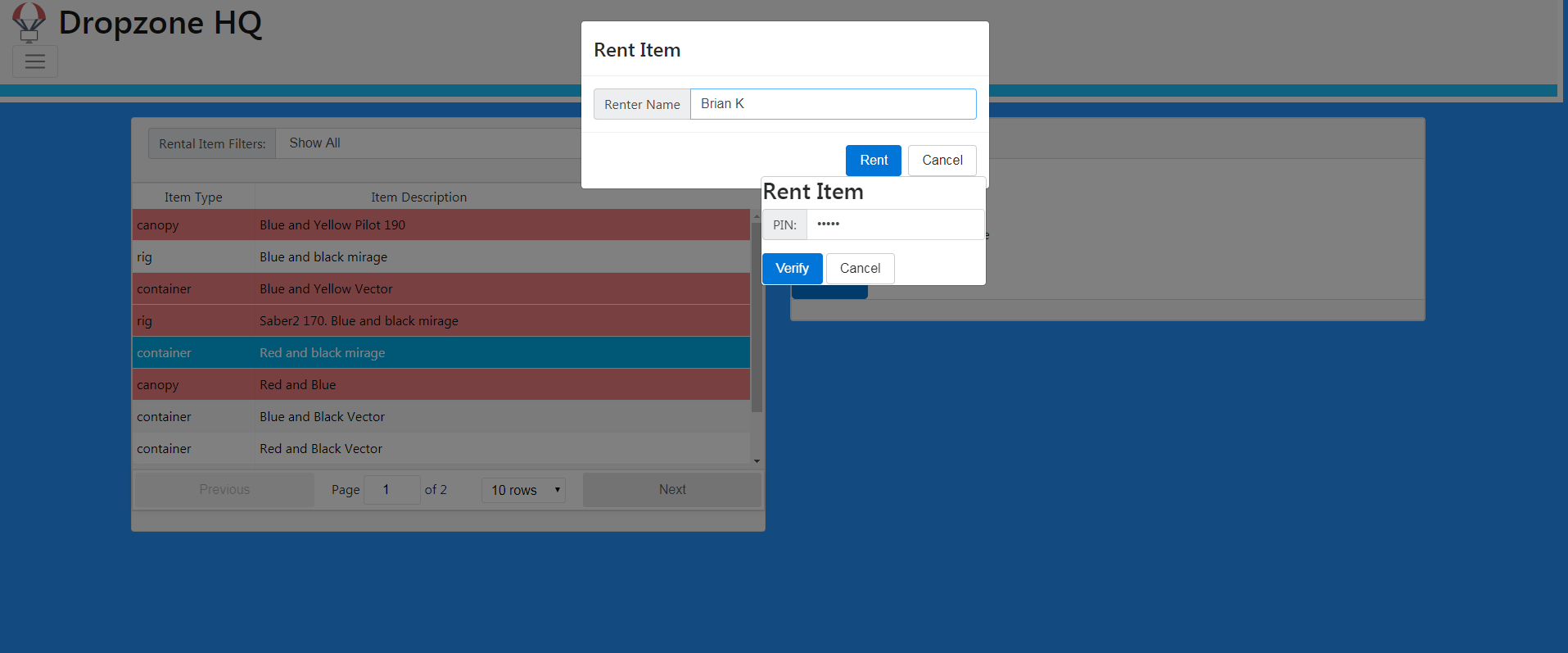
*Tandem Rig Sheets Screen DB Communication: add new Log, complete Log, edit Log, create Warning, add Service to Queue (malfunction)*

## Loft Screen

The user will have to verify their PIN when attempting to access the Loft screen before entering. If the verified user is not an Admin, Loft Head, and Loft Employee, they will be rejected and ask for the verification to be entered again. The Loft is where the equipment goes to be serviced so this screen will be able to see the Warnings entered on the other screens.

*Loft Screen DB Communication: add Service to Queue, complete Service, show whole Queue, show Queue item details, show all Warnings, show Warning details, remove Warning.*

## Rental Screen



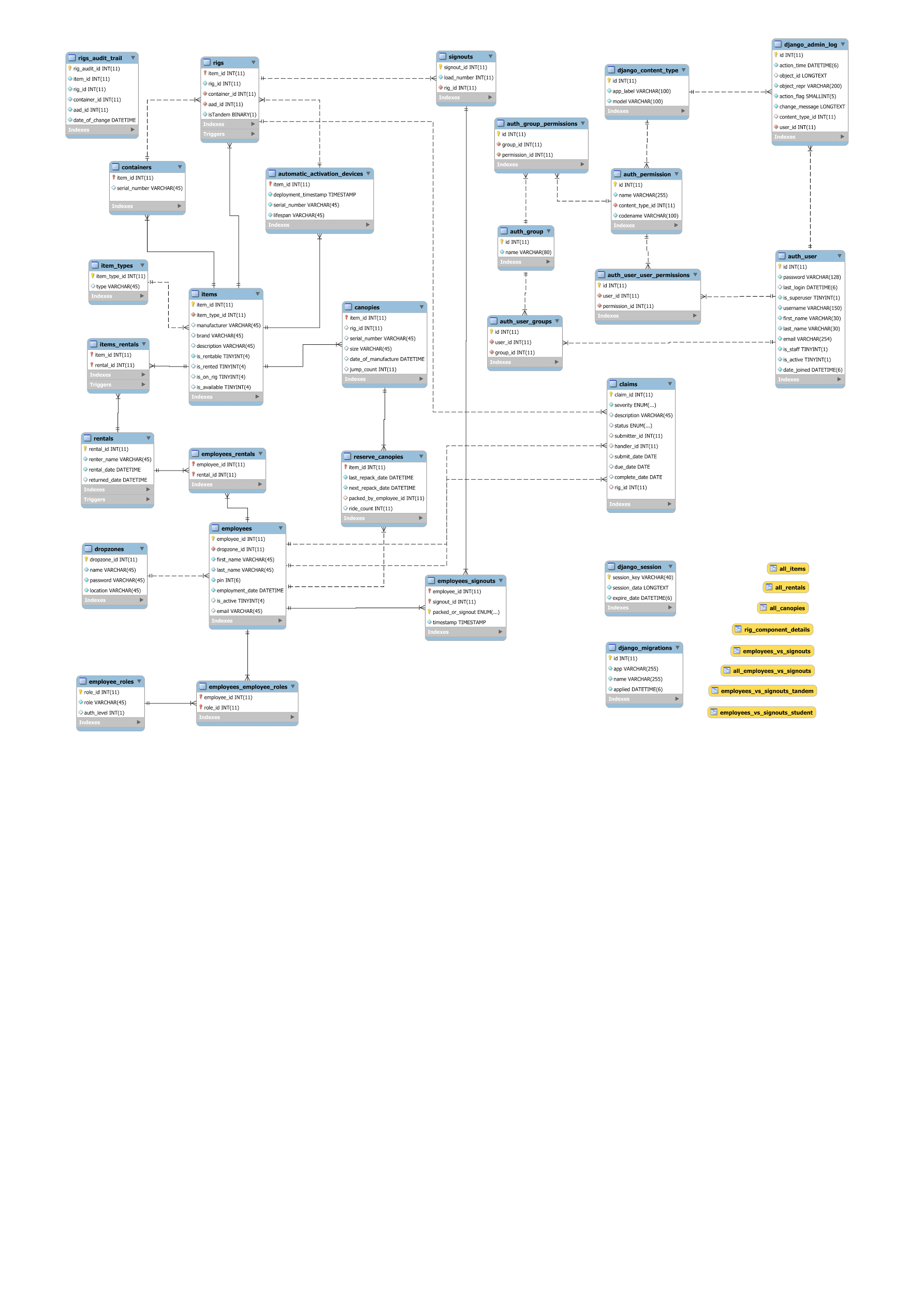
The Gear Rental Screen will allow users to loan gear out to newer fun jumpers who don’t have any or all of their own gear yet. This would be just a standard rental system in every way except we are not dealing with money. A fun jumper would request a rig from the Dropzone employee, they come to the Gear Rental screen and enter their PIN and after verification, select from the list of available rentals. Any item marked available for rent in the Inventory will be displayed in a scrollable list on the left. If the equipment is already rented, it will be visibly different from available equipment. When the renter is finished with their rental, it can be returned to the system by selecting it in the list and selecting “Return.” If a rig is marked as a rental and instructional rig, attempting to sign it out twice on any other screen or device will notify current user of the other person with the rig already signed out to talk to them about using the rig.

*Rental Screen DB Communication: sign Rental Item out, sign Rental Item in, show all Rental Items, show one Rental Item details*

# Database

## EER Diagram

[(click here for full size)](https://drive.google.com/file/d/1BAyWk755B8WrV6QJuz7GqGFF8hgF3DDj/view)



## Relationships

### Items

1. Items has multiple one to one (is a) relationships with its subclasses: containers, canopies, reserve canopies, rigs, automatic activation devices
2. Items can be Rentals, Rentals are items
3. Items have ItemTypes

### Employees

1. Employees authorize Rentals
2. Employees work at Dropzones
3. Employees have roles
4. Employees perform Services
5. Employees perform Actions
6. Employees are associated (signout rig, pack rig) with Signouts
7. Employees pack ReserveCanopies (must know who packed last)

### Rigs

1. Rigs are Items
2. Rigs have Containers
3. Rigs have Automatic Activation Devices
4. Rigs have many Canopies
5. Rigs are associated with Signouts

## 

## Triggers

1. Trigger after update on Rigs. Write old rig configuration into RigsAuditTrail to keep track of all changes made to a rig.
2. Trigger after post on Rentals. Write is\_available of the item\_id associated with rental\_id to 0.
3. Trigger after update on Rentals. Write is\_available of the item\_id associated with rental\_id to 1.

## Views of the Data

1. All Items - a view that shows all attributes about every item and item subclass in the database.
2. All Canopies - a view that simplifies viewing the two types of canopies, main and reserve, and their specific attributes.
3. All Rentals - a view that shows the rows of the rentals table in the database that have no return date.
4. Employees Vs. Signouts - a view that concisely displays all associations between employees and specific signouts. This view is used for signouts of non-tandem rigs.
5. Employees Vs. Signouts Tandem - similar to the above view but this view is used for tandem rigs only. These were separated into two different views for ease of displaying in a split screen environment.
6. Employees Vs. Signouts Student- similar to the above view but this view is used for student rigs only. These were separated into two different views for ease of displaying in a split screen environment.
7. Rig Component Detail - a view that returns a simple way to view the important information about a rig based on the id’s across various items

### Possible Future Views

1. Employees Vs. Rentals Vs. Items (Rentals Log)
2. Employees Vs. Services Vs. Actions (Service Log)

## 

## Design Choices

The goal in designing this database was to reach a balance between scalability, speed, size, ease of querying, and normalization. Some relationships in the database were only decided upon after a few key debates. The relationship between Employees and Signouts is one such example. Signouts are a way to keep track of what skydiving rigs have gone out on a plane load, what instructor used the rig, and what employee packed it upon return from the jump. As of right now, there are only two roles of employees that are recorded in a Signout: the Instructor and the Packer of the rig.

The question arose as to whether these should be two separate “one to many” relationships or a single “many to many” relationship with a bridging table. The reasoning behind having a bridging table was to preserve the potential to expand roles in a signout. Sometime in the future, there may be a third employee role that a business wants to associate with signouts (perhaps some dropzones inspect their rigs upon return and want to track “Inspected By” for each signout). As a team we ultimately decided to use a bridging table in favor of this added flexibility. Having two “one to many” relationships on a signout would depend too strongly on the constancy of having exactly two roles associated with a signout. Furthermore, the two “one to many” relationships setup lends itself more easily to queries that are employee role specific, where you would never care about having these two different types of employee roles in the same query or report. Since, in our use cases, we more often want all roles to be in the same report, the bridging table route made more sense. The “many to many” with a bridging table allows for the addition of employee roles to signouts. This approach has more variability in the amount of roles that can be associated with a signout and allows for better rig specific querying in cases where we want to see all employees who handled a specific rig.

In several parts of our database we have to store data that has a finite set of potential values. In these cases, there were discussions on the use of enumerated attributes versus another table with two fields (simple integer id and type). Items, Services, and Actions were the primary points of discussion. Ultimately, we decided to use both tables and enumerated attributes on a case by case basis. For attributes that we could foresee adding or subtracting values to or using in other tables, we used a separate table (such as item types). For attributes that we believed would be relatively constant, like employee service types, we chose an enumeration. These decisions were made knowing that adding or subtracting values to an enumeration is bad practice, since these changes are very taxing on the system. After adding or removing an enum value, the system has to find every row that contains the enumeration, update its options, and then confirm that no corruption has taken place. The system checks every row that contains the enumeration even if you’re simply adding a value for future use that doesn’t appear in the database yet. Both additions and deletions of enum values are handled in the same way.

One of the features that has been implemented is a Pin system for employees to do their specified actions quickly and efficiently. As was noted by a trip to the dropzone the employee’s wanted speed over safety. Thus the idea of the six digit pin, a quick way to lookup the employee and add their attempted log to the database. The pin is saved in a SHA256 hash that is only visible to the employee via an email that is sent at the time of the creation of the account. This keeps the number private and allows for a simple level of verification.

# API Endpoints

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Description** | **Methods** |
| /rigs/{rig ID} | Basic info and IDs of items for one rig | GET, PUT, PATCH, DELETE |
| /rigs/ | All rigs with their basic info and IDs of their items | GET, POST |
| /rig\_info/ | All rigs with details about the items that are on them | GET |
| /rigs/students/available-for-signout | All student rigs available for signout | GET, POST |
| /rigs/tandems/available-for-signout | All tandem rigs available for signout | GET, POST |
| /rigsheets/all | All rigsheet signout records | GET, POST |
| /rigsheets/{signout ID} | A specific rigsheet signout | GET, PUT, PATCH, DELETE |
| /rigsheets/tandems | All rigsheet signouts for tandem rigs | GET, POST |
| /rigsheets/students | All rigsheet signouts for student rigs | GET, POST |
| /employees/{Employee ID} | A specific employee’s info | GET, PUT, PATCH, DELETE |
| /employees/ | All employees’ info | GET, POST |
| /itemtypes/ | A list of item types | GET, POST |
| /itemtypes/{iItem type ID} | A specific item type | GET, PUT, PATCH, DELETE |
| /items/{item ID} | A specific item’s info | GET, PUT, PATCH, DELETE |
| /items/ | A view of all items, containing all fields for all items, null when they don’t apply | GET, POST |

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Description** | **Methods** |
| /aads/{AAD ID} | A specific AAD’s info | GET, PUT, PATCH, DELETE |
| /aads/ | A list of all AAD info | GET, POST |
| /containers/ | A list of all container info | GET, POST |
| /containers/{Container ID} | A specific container’s info | GET, PUT, PATCH, DELETE |
| /canopies/ | A list of all canopy info | GET, POST |
| /canopies/{Canopy ID} | A specific canopy’s info | GET, PUT, PATCH, DELETE |
| /canopies/reserve | A list of all reserve canopy info | GET, POST |
| /canopies/reserve/{Reserve Canopy ID} | A specific reserve canopy’s info | GET, PUT, PATCH, DELETE |
| /rental-items/ | A list of all items that are part of the rental system | GET, POST |
| /rentals/active | A list of all rental records that are active (have not yet been returned) | GET, POST |
| /rentals/{Rental ID} | A specific rental record | GET, PUT, PATCH, DELETE |
| /claims/ | A list of all claims (item damage reports) | GET, POST |
| /claims/warnings | A list of all claims that have a PENDING status | GET, POST |
| /claims/queue | A list of all claims that have an IN-PROGRESS status | GET, POST |
| /claims/{Claim ID} | Info for a specific claim | GET, PUT, PATCH, DELETE |

# Glossary

**Canopy:** The conventional idea of what is known as a "parachute." The canopy is the colored nylon wing used to steer and slow down the fall of the skydiver.

**Container:** The part of the system that the skydiver wears, it resembles a backpack with leg straps.

**Main:** The main canopy is the first one that is deployed, if the main malfunctions the skydiver cuts the main away and attempts to deploy the reserve.

**Reserve:** The reserve canopy is the backup canopy used in cases of the main canopy malfunctioning.

**Rig:** A Rig is the combination of all of the above. When a skydiver is ready to jump out of the plane they are wearing their rig.

**Load:** Each time the plane takes off with a group of skydivers that is considered 1 Load.