Contents

1 Analysis

1.1 Problem Identification

1.1.1 Problem Description

Popular inventory management solutions are relatively expensive, and may be out of reach for individuals or small schools. Inventory systems have numerous benefits for businesses and individuals alike; a business may choose to track their supply levels where an individual may wish to catalogue their DVD collection.

My goal is to create a web-based application aimed at both businesses and individuals to manage inventory, with additional modern features such as automatic item re-ordering when stocks are running low.

Traditional inventory management solutions are typically single-user at best, whereas I intend to create a multi-user, collaborative environment.

In my view, an inventory system should be:

- Easy for end users to use.
- Cross platform
- Performant interface
- Efficient in terms of adding data
- Allow for easy cataloguing of inventory
- Allow for item scanning using QR codes / barcodes
- Be able to source data from external sources
- Support both consumable and non-consumable goods.

1.1.2 Stakeholders

Stakeholder requirements are further discussed for each stakeholder in the Stakeholder Requirements section.

Stakeholder	Description	Requirements	Capability
Claire Foley	Senior Leadership Team at The Village Prep School	Ability to manage library books. Admin and supervision of other users carrying out librarian tasks	Well-versed in computer use, at least when it comes to intuitive and well designed interfaces. Would struggle with a non-intuitive interface design.
Ella	”Head Librarian” (Pupil) at the Village Prep School	Ability to check in and out library books. User of the system; requires interface that is appropriate for her age.	Beginner user of technology, proficient in mobile applications on phones and tablets only. Rarely uses a laptop or desktop computer.
Generic Gear Rental Shop	Photography gear for hire business	Ability to manage business inventory in a fast and efficient manner.	Proficient with computers.

1.1.3 Why is it suitable to a computational solution?

1.2 Investigation

1.2.1 Preparation for interview

Question Set

- What would you consider your skill level to be regarding technology?
- Do you currently have a way to manage inventory?
- If so, what is your current solution?
- What aspects of this solution do you like?
- What aspects of this solution do you dislike?
- What features would you **require** in a custom solution?
- What features would **enhance** your experience?

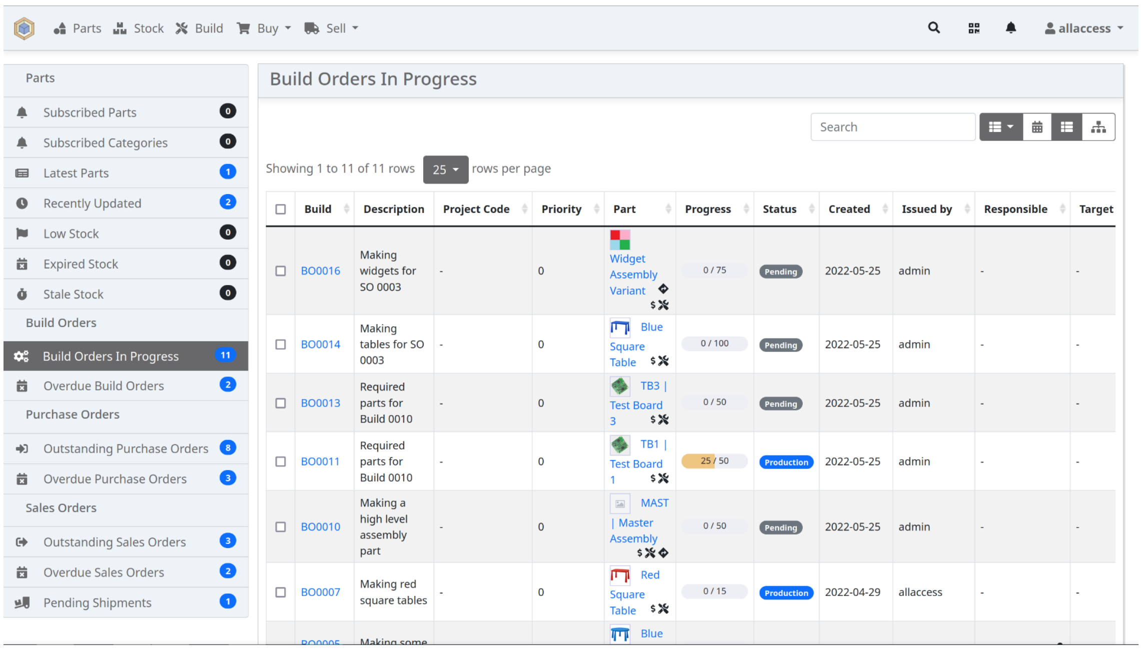
1.2.2 Interviews

1.2.3 Summary of interviews

1.3 Research

1.3.1 Existing similar solutions

InvenTree `https://inventree.org/`



Parts	
Subscribed Parts	0
Subscribed Categories	0
Latest Parts	1
Recently Updated	2
Low Stock	0
Expired Stock	0
Stale Stock	0
Build Orders	
Build Orders In Progress	11
Overdue Build Orders	2
Purchase Orders	
Outstanding Purchase Orders	8
Overdue Purchase Orders	3
Sales Orders	
Outstanding Sales Orders	3
Overdue Sales Orders	2
Pending Shipments	1

Build Orders In Progress

Showing 1 to 11 of 11 rows 25 rows per page

Build	Description	Project Code	Priority	Part	Progress	Status	Created	Issued by	Responsible	Target
BO0016	Making widgets for SO 0003	-	0	Widget Assembly Variant	0 / 75	Pending	2022-05-25	admin	-	-
BO0014	Making tables for SO 0003	-	0	Blue Square Table	0 / 100	Pending	2022-05-25	admin	-	-
BO0013	Required parts for Build 0010	-	0	TB3 | Test Board 3	0 / 50	Pending	2022-05-25	admin	-	-
BO0011	Required parts for Build 0010	-	0	TB1 | Test Board 1	25 / 50	Production	2022-05-25	admin	-	-
BO0010	Making a high level assembly part	-	0	MAST | Master Assembly	0 / 50	Pending	2022-05-25	admin	-	-
BO0007	Making red square tables	-	0	Red Square Table	0 / 15	Production	2022-04-29	allaccess	-	-
BO0005	Making some			Blue						

Overview
InvenTree is an **open-source** inventory management system, providing *low level stock control and part tracking*. It uses a Python/Django database backend and provides both a **web-based interface** as well as a REST API for interacting with other services. InvenTree also has a powerful plugin system for custom applications and other extensions.

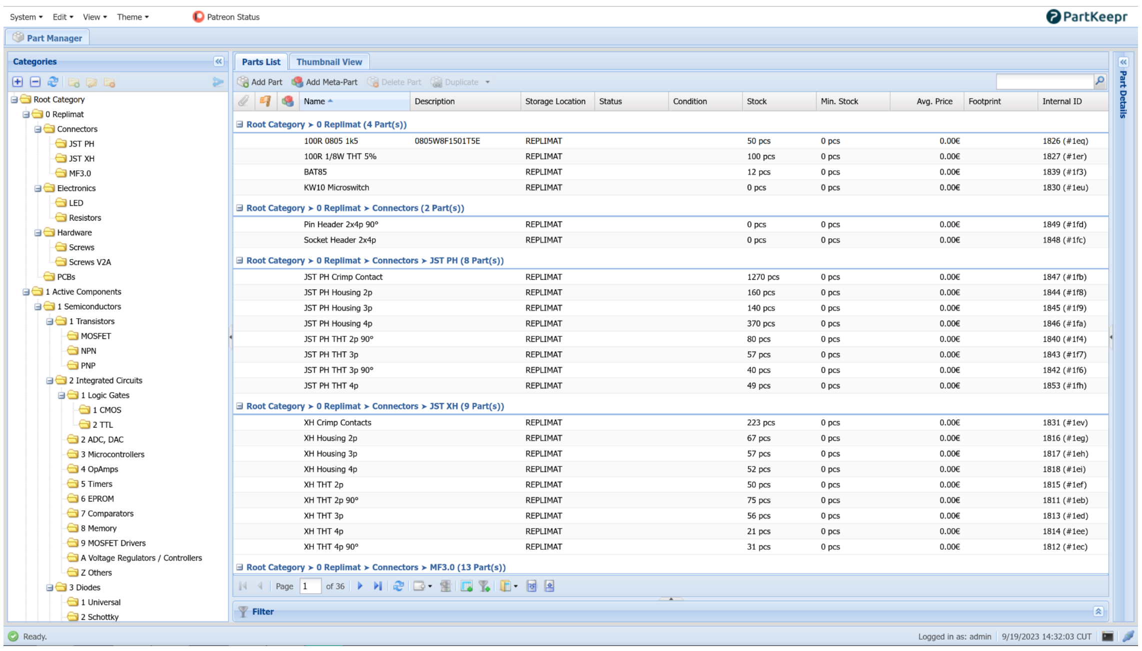
Parts applicable to my solution

- Web-based application
  The application will be web-based.
- Modern, Relatively simple user interface
  InvenTree offers a relatively simple and intuitive user interface.

Parts not applicable to my solution

- Stock control and part tracking specific features
  I am looking to implement a system that is capable of being far more generalized than just part tracking, although the system will have features for library book tracking.

PartKeepr `https://partkeepr.org/`



Name	Description	Storage Location	Status	Condition	Stock	Min. Stock	Avg. Price	Footprint	Internal ID
Root Category > 0 Replimat (4 Part(s))									
100R 0805 1k5	0805W8F1501T5E	REPLIMAT			50 pcs	0 pcs	0.00€		1826 (#1eq)
100R 1/8W THT 5%		REPLIMAT			100 pcs	0 pcs	0.00€		1827 (#1er)
BAT85		REPLIMAT			12 pcs	0 pcs	0.00€		1839 (#1f3)
KW10 Microswitch		REPLIMAT			0 pcs	0 pcs	0.00€		1830 (#1eu)
Root Category > 0 Replimat > Connectors (2 Part(s))									
Pin Header 2x4p 90°		REPLIMAT			0 pcs	0 pcs	0.00€		1849 (#1fd)
Socket Header 2x4p		REPLIMAT			0 pcs	0 pcs	0.00€		1848 (#1fc)
Root Category > 0 Replimat > Connectors > JST PH (8 Part(s))									
JST PH Crimp Contact		REPLIMAT			1270 pcs	0 pcs	0.00€		1847 (#1fb)
JST PH Housing 2p		REPLIMAT			160 pcs	0 pcs	0.00€		1844 (#1f8)
JST PH Housing 3p		REPLIMAT			140 pcs	0 pcs	0.00€		1845 (#1f9)
JST PH Housing 4p		REPLIMAT			370 pcs	0 pcs	0.00€		1846 (#1fa)
JST PH THT 2p 90°		REPLIMAT			80 pcs	0 pcs	0.00€		1840 (#1f4)
JST PH THT 3p		REPLIMAT			57 pcs	0 pcs	0.00€		1843 (#1f7)
JST PH THT 3p 90°		REPLIMAT			40 pcs	0 pcs	0.00€		1842 (#1f6)
JST PH THT 4p		REPLIMAT			49 pcs	0 pcs	0.00€		1853 (#1fh)
Root Category > 0 Replimat > Connectors > JST XH (9 Part(s))									
XH Crimp Contacts		REPLIMAT			223 pcs	0 pcs	0.00€		1831 (#1ev)
XH Housing 2p		REPLIMAT			67 pcs	0 pcs	0.00€		1816 (#1eg)
XH Housing 3p		REPLIMAT			57 pcs	0 pcs	0.00€		1817 (#1eh)
XH Housing 4p		REPLIMAT			52 pcs	0 pcs	0.00€		1818 (#1ei)
XH THT 2p		REPLIMAT			50 pcs	0 pcs	0.00€		1815 (#1ef)
XH THT 2p 90°		REPLIMAT			75 pcs	0 pcs	0.00€		1811 (#1eb)
XH THT 3p		REPLIMAT			56 pcs	0 pcs	0.00€		1813 (#1ed)
XH THT 4p		REPLIMAT			21 pcs	0 pcs	0.00€		1814 (#1ee)
XH THT 4p 90°		REPLIMAT			31 pcs	0 pcs	0.00€		1812 (#1ec)
Root Category > 0 Replimat > Connectors > MF3.0 (13 Part(s))									

Overview

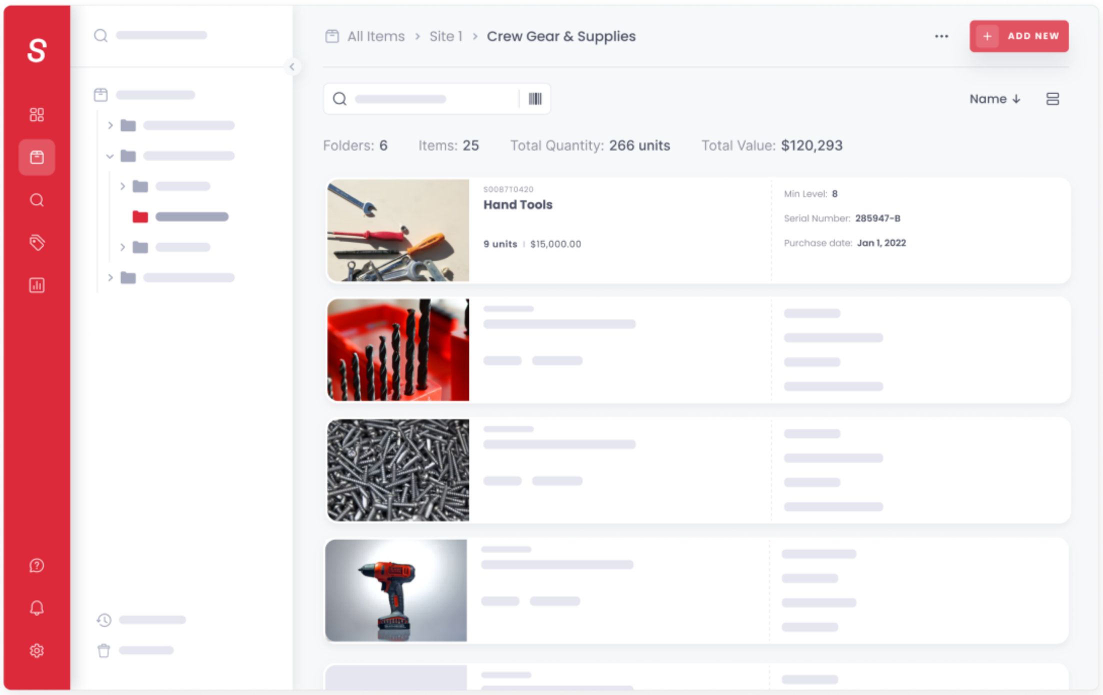
PartKeepr is an open-source inventory management system with a focus on electronic components. It is designed around four main principles:

- Fast Part Searching
- Ability to add complete part database
- Keeping track of stock
- Ease of use

Parts applicable to my solution

Like PartKeepr, I hope to implement a web-based interface. However, I am using a different approach as my solution will not be tailored specifically to electronic components.

Sortly `https://www.sortly.com/solutions/inventory-management-software/`



Overview
Sortly is a proprietary cloud-based inventory management system with a focus on small businesses and inviduals.

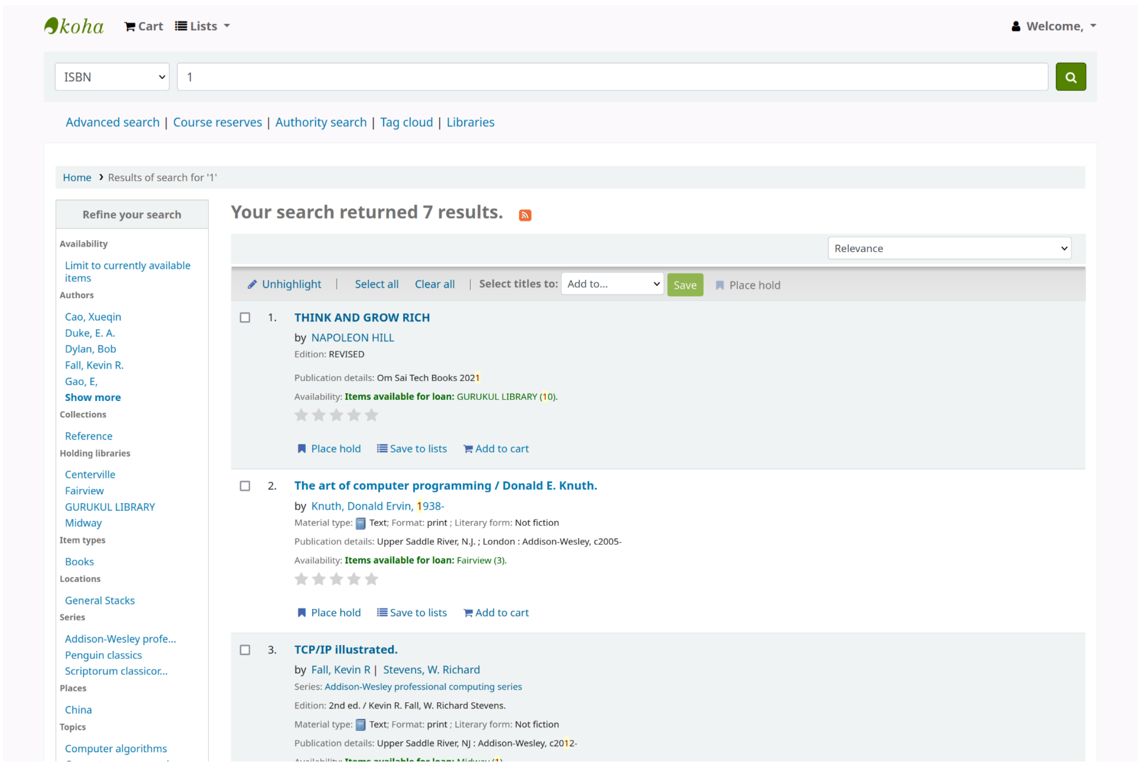
It has two plans available, an always free plan with limited functionality and a paid plan will a more complete feature-set.

Parts applicable to my solution
I hope to implement the following features from Sortly:

- Web based interface
  - Allows for easy access.
- Barcode support
  - Allows end users to print off QR codes to stick to items
  - Which can be scanned in-app to easily perform actions on the item.
- Real-time reporting insights
  - Allows for added insight into usage patterns for particular units.

Koha

`https://koha-community.org/`


koha
Cart
Lists
Welcome,
ISBN
1
Advanced search | Course reserves | Authority search | Tag cloud | Libraries
Home › Results of search for '1'
Refine your search
Relevance
Unhighlight | Select all Clear all | Select titles to: Add to... Save Place hold

Availability
Limit to currently available items
Authors
Cao, Xueqin
Duke, E. A.
Dylan, Bob
Fall, Kevin R.
Gao, E,
Show more
Collections
Reference
Holding libraries
Centerville
Fairview
GURUKUL LIBRARY
Midway
Item types
Books
Locations
General Stacks
Series
Addison-Wesley profe...
Penguin classics
Scriptorum classicor...
Places
China
Topics
Computer algorithms

Your search returned 7 results.

1. THINK AND GROW RICH
by NAPOLEON HILL
Edition: REVISED
Publication details: Om Sai Tech Books 2021
Availability: Items available for loan: GURUKUL LIBRARY (10).
Place hold Save to lists Add to cart

2. The art of computer programming / Donald E. Knuth.
by Knuth, Donald Ervin, 1938-
Material type: Text; Format: print ; Literary form: Not fiction
Publication details: Upper Saddle River, N.J. ; London : Addison-Wesley, c2005-
Availability: Items available for loan: Fairview (3).
Place hold Save to lists Add to cart

3. TCP/IP illustrated.
by Fall, Kevin R | Stevens, W. Richard
Series: Addison-Wesley professional computing series
Edition: 2nd ed. / Kevin R. Fall, W. Richard Stevens.
Material type: Text; Format: print ; Literary form: Not fiction
Publication details: Upper Saddle River, NJ : Addison-Wesley, c2012-

Overview

Parts applicable to my solution

1.3.2 Features to be incorporated into solution

1.3.3 Limitations of the solution

1.3.4 Feedback from stakeholders

1.4 Requirements

1.4.1 Stakeholder requirements

1.4.2 Software and hardware requirements

System Requirements

Hardware	**Justification**
Laptop/Desktop Keyboard and Pointing Device (eg. Mouse)	For desktop or laptop computer users, a suitable input device is required in order to interact with the software. A pointing device (a mouse) is necessary in order to interact with the user interface, to perform actions such as clicking buttons, icons, and opening menus. A keyboard will be used to manually input data into the system.
Tablet Device Touchscreen	For tablet users, it would be impractical to expect the user to have access to a keyboard and or pointing device. Therefore, we must design the system to accept inputs from a touchscreen. This will be easier to use and more intuitive for tablet users. The touchscreen will be used to input data into the system and to interact with the user interface.
Dual-Core Processor (x86, ARM, RISCV architectures)	A modern processor with sufficient resources to run an up-to-date web browser such as Chrome, Edge or Firefox is required in order to access the web-based interface.
2GB of RAM	Sufficient RAM is required to run the web browser, which can be a memory intensive task.
Monitor	To display the user interface.
Network Interface Card (NIC)	A Network Interface Card, or NIC, is required for the computer to be connected to a network, such as the Internet. This is required as the web interface will be hosted on a domain and server that is external to the user, that is to say, not on their local network.
Optional: *Wireless Network Adapter*	*A Wireless Network Adapter is an optional requirement, it will allow the user to connect to a wireless network in order to access the network or Internet so that they can access the external user interface.*
Optional: *Camera*	*A Camera is an optional requirement; devices with cameras will be able to scan barcodes or QR codes corresponding to inventory items and easily perform actions on them.*

Software Requirements

Talk about why I don't need much software since dependencies hosted on the server.

Software	**Justification**
Operating System *(Windows, MacOS, Linux, ChromeOS, iOS, iPadOS, Android)*	An operating system is required in order to run the web browser necessary to access the interface.
A web browser *(eg. Chrome, Firefox, Microsoft Edge, Safari)*	A web browser is necessary to access the interface as it will be primarily a web application.

1.4.3 Success requirements

The overall objectives for the system.
To measure the overall effectiveness of the system, targets must be set before writing the program. These targets will help in the evaluation stage to determine weather our objectives have been met. These objectives will be **SMART**, i.e:

- **Specific**
  What objective needs to be accomplished?
- **Measurable**
  How can we quantify this objective?
  How will the success of this objective be measured? (quantitatively or qualitatively)
- **Achievable**
  Is this objective achievable and realistic? If so, how to you plan to achieve them?
- **Relevant**
  How does this objective benefit the end-users of this application as a whole?
  Why has this goal been set?
- **Timely**
  Can this objective be completed within an appropriate time frame?
  At what stage in the software development lifecycle will you start implementing this goal?
  In which order will any sub-objectives be completed?

The Project's SMART Objectives

1. **To produce a solution for cataloguing a school library and recording users and books borrowed**
   At the end of the project, I will evaluate against my success criteria and determine weather this objective has been met. On the software side, I will be using React, Expo and PostgreSQL. This objective will be the main objective for this project. This objective must be completed by March 2024.
2. **To produce a solution including a database that can store details of books, borrowers, loans and returns**
3. **To produce an intuitive and easy to use solution**
   I will evaluate my success on this objective by having a new user without any prior training or advice use the system and try to carry out a number of tasks without any assistance. If the user is able to successfully complete the tasks I will consider the system to be intuitive and easy to use and therefore this objective satisfied. To achieve this I will design my system to have a consistent layout based on **Material Design 2**, (`https://m2.material.io/`) the design language used by Google products and many apps running on the Android operating system. I will also use language that is a) appropriate for the situation the product will be deployed in (with young children) and b) easy to understand (so that children can interact with the system) I will also use meaningful error messages so that the user has a clear understanding of the problem that has occurred. This objective will benefit the end user as a intuitive and easy to use solution is critical to the usefullness of the project. If the end product is not easy to use, it is less likey to be used and accepted by my stakeholders. This objective will be worked on during the development process, and so will

be completed by the time development concludes. I will mock-up a version of the user interface in the design stage and will continuously iterate on the user interface during development.

4. **To produce a solution that features a fully searchable catalogue**
5. **To produce a solution that features reporting for overdue and/or lost books**
6. **To produce a solution that includes a curated ”suggested reading list” for each borrower**
7. **To produce a solution containing a user interface that can be accessed via a mobile device**

2 Design

2.1 User Interface Design

2.1.1 Usability Features

2.1.2 Feedback from stakeholder

2.2 Modular breakdown

2.3 Algorithms

2.4 Data Dictionary

2.5 Inputs and outputs

2.6 Validation

2.7 Testing

2.7.1 Methods

2.7.2 Test Plan

3 Implementation

3.1 First Iteration — Initial Backend and Database

3.1.1 Introduction

In this sprint I will work on the backend service. This service will provide an interface for the frontend to talk to the database via an API (Application Programming Interface). I am writing the backend in Go, which is a performant, statically typed high level language designed by Google. Go is frequently used for backend development thanks to it's excellent performance and built in memory safety. I am also going to use GraphQL as the query language used by the frontend to interact with the backend.

GraphQL is an open-source query and manipulation language designed for use in APIs. The backend will serve as an API which will interface with my database. I choose to use GraphQL as it is better suited for larger, more complex data sources, and supports querying for multiple different types of data at once, unlike REST. It is also something I was interested in learning more about as I have not designed a system using it before.

James: Should this intro be in the design area instead?
TODO: explain what a graphql mutation is (it's a function)

3.1.2 User account creation

The first feature I decided to work on was user account creation. This would involve asking the user for an email address, name and password, before validating it and inserting it into the database. In addition, at a later stage, validation must be performed in order to ensure that:

- The user email is not already in use
- The generated user ID is unique and not already in use

For this early stage of development, I decided to use an SQLite database to make things easier. I hope that I can easily switch this to PostgreSQL (as specified in my design doc) later on in the development process.
I have created a GraphQL mutation called `createUser`. When it is called, the GraphQL library calls the `CreateUser` function, passing any input data from the query.
The database connection is available at "`r.db`".

My first version of this function was as follows:

```go
// CreateUser is the resolver for the createUser field.
func (r *mutationResolver) CreateUser(ctx context.Context, input model.NewUser) (*model.User,
    error) {
    // Create the user struct
    user := structs.User{FirstName: input.FirstName, LastName: input.LastName, Email: input.Email}

    // Generate a user ID
    user.ID = uuid.New()

    // Create the database entry
    r.db.Create(&user)

    // Return the created user data, converting it to a GraphQL model.
    return &model.User{
        ID: user.ID.String(),
        FirstName: user.FirstName,
        LastName: user.LastName,
        Email: user.Email
    }, nil // return nil in the error field
```

The first thing I noticed after implementing this function was that it was tedious to convert back and forth between `structs.User` (the database object) and `model.User` (the GraphQL object). I decided to merge these into a single object. This was done with the following lines in my GraphQL library's configuration file:

```
models:

[..]

    # Custom models
    Checkout:
      model: github.com/jcxldn/fosscat/backend/structs.Checkout
    Entity:
      model: github.com/jcxldn/fosscat/backend/structs.Entity
    Item:
      model: github.com/jcxldn/fosscat/backend/structs.Item
    User:
      model: github.com/jcxldn/fosscat/backend/structs.User
```

This instructs the GraphQL library to use the structs I defined for the database as if they were GraphQL models.

Unique ID generation

I decided to use UUIDs (Universal Unique Identifiers) as IDs for all of the objects in my database. (eg. Users, Items) As seen above, I initially choose to simply call `uuid.New()` to generate a new random UUID. However, I would soon realise that it would be beneficial to perform **validation** in order to ensure that the UUIDs were actually unique, ie. that they were not being used by another object of the same type. For example, I wouldn't want two users to have the same User ID.

I decided to use a **for loop** to continuously generate UUIDs to be used as a potential User ID. I then perform a database lookup to ensure that the UUID is not already in use.

This can be done with the following code:

```
func (r *mutationResolver) CreateUser(ctx context.Context, input model.NewUser) (*structs.User,
    error) {

[..]

    isFreeUuid := false
    for !isFreeUuid {
        // Generate a UUID for the user id.
        user.ID = uuid.New()
        // Check that the UUID has not been used already
        // If true, it will break out of this for loop and continue.
        isFreeUuid = util.IsUuidFree[structs.User](r.db, user.ID)
    }
```

In order to achieve this and reduce code duplication across different functions,
I created a "IsUuidFree" utility function. Here is the code:

```

    func IsUuidFree[T any](db *gorm.DB, id uuid.UUID) bool {
        obj := new(T)
        err := db.Model(obj).Select("id").Where("id == ?", id.String()).First(&obj).Error
        if errors.Is(err, gorm.ErrRecordNotFound) {
            // Record not found, so user id is free
            return true
        } else {
            // Record was returned successfully, therefore the user exists
            return false
        }
    }
```

JAMES: this initially was different but I changed it to make it simpler. Should I include the old version as well?

This function makes use of **generics**. As per the Go docs:

> *With generics, you can declare and use functions or types that are written to work with any of a set of types provided by calling code.*

To simplify, generics mean that I can pass any struct (**T**) to the function. For example, if I call the function with:

```
util.IsUuidFree[structs.User](r.db, user.ID)
```

Then T is set to the type `structs.User`.

GORM (my database library) works by defining a struct to query for which corresponds to a table in the database (in this case the `Users` struct corresponds to the `users` table). We can then perform SQL actions on this table, such as Select.

Therefore, the GORM db call from above:

```
db.Model(obj).Select("id").Where("id = ?", id.String()).First(&obj)
```

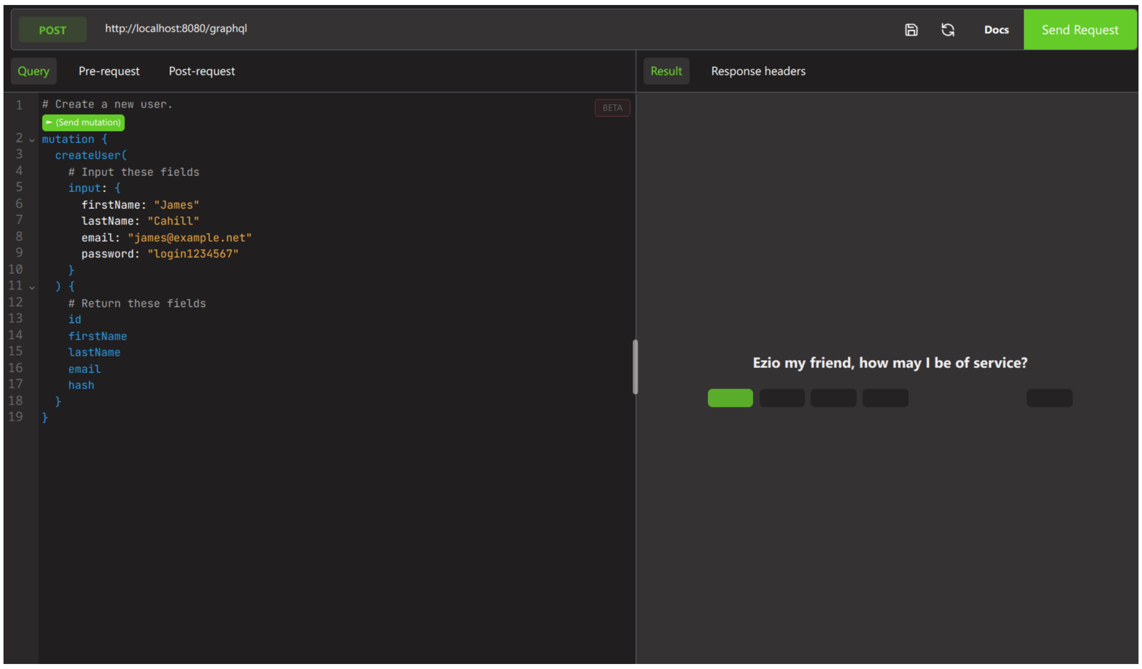
is the equivalent of:

```
SELECT id FROM users WHERE id == ? LIMIT 1 VALUES ("value of id.String()")
```

Testing the user account creation flow

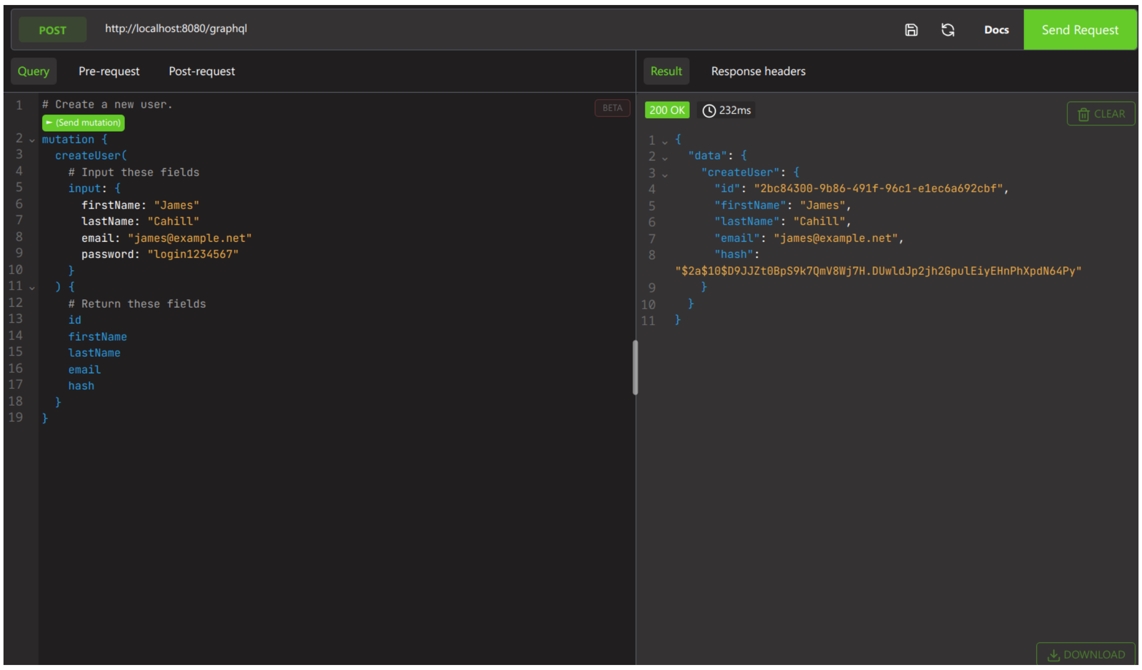
Now that I have implemented user account creation, I need to test it to verify that my solution works as expected. I have added a GraphQL query to list all users, which I will use in conjunction with the `createUser` mutation.

I am using a piece of software called **Altair**, which is an interactive way to make GraphQL queries. I start by creating my GraphQL query which includes the `createUser` mutation, making sure to set all required fields:



```
 1  # Create a new user.
 2  mutation {
 3    createUser(
 4      # Input these fields
 5      input: {
 6        firstName: "James"
 7        lastName: "Cahill"
 8        email: "james@example.net"
 9        password: "login1234567"
10      }
11    ) {
12      # Return these fields
13      id
14      firstName
15      lastName
16      email
17      hash
18    }
19  }
```

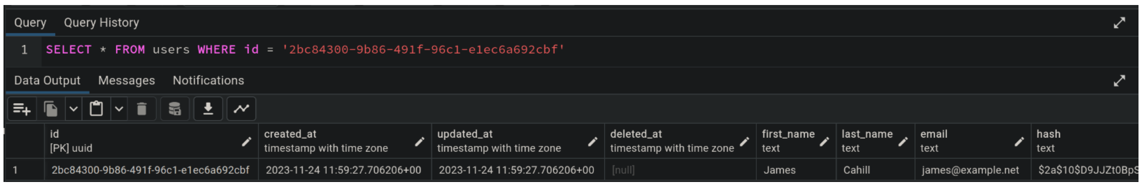
Running the query results in the following response:



```
 1  # Create a new user.
 2  mutation {
 3    createUser(
 4      # Input these fields
 5      input: {
 6        firstName: "James"
 7        lastName: "Cahill"
 8        email: "james@example.net"
 9        password: "login1234567"
10      }
11    ) {
12      # Return these fields
13      id
14      firstName
15      lastName
16      email
17      hash
18    }
19  }
```

```
{
  "data": {
    "createUser": {
      "id": "2bc84300-9b86-491f-96c1-e1ec6a692cbf",
      "firstName": "James",
      "lastName": "Cahill",
      "email": "james@example.net",
      "hash":
"$2a$10$D9JJZt0BpS9k7QmV8Wj7H.DUwldJp2jh2GpulEiyEHnPhXpdN64Py"
    }
  }
}
```

Let's check the database for our new user to ensure it was added successfully:



```
SELECT * FROM users WHERE id = '2bc84300-9b86-491f-96c1-e1ec6a692cbf'
```

	id [PK] uuid	created_at timestamp with time zone	updated_at timestamp with time zone	deleted_at timestamp with time zone	first_name text	last_name text	email text	hash text
1	2bc84300-9b86-491f-96c1-e1ec6a692cbf	2023-11-24 11:59:27.706206+00	2023-11-24 11:59:27.706206+00	[null]	James	Cahill	james@example.net	$2a$10$D9JJZt0BpS

We can see that the user is created successfully, added to the database, and the specified fields (line 12 onwards in the query) are returned to the client.

3.1.3 Problems encountered when moving to PostgreSQL

At this stage, with most of the core functionality implemented, I decided to switch back to PostgreSQL. However, when doing this I encountered two problems:

Problem 1 - Entity relation errors
When creating the database in PostgreSQL, I encountered the following error, displayed in the backend logs.

```
/backend/database.go:35 ERROR: relation "entities" does not exist (SQLSTATE 42P01)

[17.308ms] [rows:0] CREATE TABLE "checkouts" ("id" text,"created_at" timestamptz,"updated_at"
    timestamptz,"deleted_at" timestamptz,"take_date" timestamptz,"return_date"
    timestamptz,PRIMARY KEY ("id"),CONSTRAINT "fk_entities_checkouts" FOREIGN KEY ("id")
    REFERENCES "entities"("id"))
```

This error could be traced to the following line in my code, where the database is "migrated" by GORM, which means that it attempts to create the necessary tables and columns to match the structs I have defined.

```
// "Migrate" the schema
// This will create tables, keys, columns, etc.
// See https://gorm.io/docs/migration.html
// Note that we need to pass each struct in our schema.
db.AutoMigrate(&structs.Checkout{}, &structs.Entity{}, &structs.Item{}, &structs.User{})
```

Could do: For example struct A creates this table? show it off?

This error prevented me from progressing with the backend implementation, as the program would error out during table creation (should I remove this line?)

After some investigation, I found out that this error occurs when tables that have dependencies on each other are created at the same time (in the same `AutoMigrate` call). The fix was to create the dependent table first followed by the table that depended on it, the code for which can be seen below:

```
// "Migrate" the schema
// [..]
db.AutoMigrate(&structs.Checkout{})
// Item has a dependency on Entity, so do them in the correct order
// to avoid "relation does not exist" error during table creation.
db.AutoMigrate(&structs.Entity{})
db.AutoMigrate(&structs.Item{})
db.AutoMigrate(&structs.User{})
```

After making this change, I decided to validate and test it before moving on.

I decided to test this change by first deleting all the database tables and then starting the backend, which should create (or "migrate") all of the tables.

Firstly I will connect to the database and delete the tables. I have attached the console output and have annotated what I am doing to make it easier to understand.

```
// Run 'psql' to connect to the database
[james@linux cs-coursework]$ psql -h localhost -U fosscat -W
// Enter the password (it is not displayed)
Password:
psql (15.4, server 16.1 (Debian 16.1-1.pgdg120+1))
```

```
WARNING: psql major version 15, server major version 16.
        Some psql features might not work.
Type "help" for help.

// List all tables, we can see that they exist
fosscat=# \dt
         List of relations
 Schema | Name | Type | Owner
--------+-----------+-------+---------
 public | checkouts | table | fosscat
 public | entities | table | fosscat
 public | items | table | fosscat
 public | users | table | fosscat
(4 rows)

// Delete the schema containing all of the tables
fosscat=# DROP SCHEMA public CASCADE;
NOTICE: drop cascades to 4 other objects
DETAIL: drop cascades to table users
drop cascades to table checkouts
drop cascades to table entities
drop cascades to table items
DROP SCHEMA
// Re-create the schema
fosscat=# CREATE SCHEMA public;
CREATE SCHEMA
// Set default permissions on schema
fosscat=# GRANT ALL ON SCHEMA public TO public;
GRANT
// List all tables, we can see that there aren't any
fosscat=# \dt
Did not find any relations.
// Quit
fosscat-# \q
```

As can be seen above, the database now contains no tables. Next, let's start the backend, where GORM should recreate the database tables. Below is the startup log:

```
[james@linux cs-coursework]$ GIN_MODE=release ./start-backend.sh
2023/12/05 14:23:08 [database] connected. migrating...
2023/12/05 14:23:08 [database] migrated, done.
2023/12/05 14:23:08 [resolver] db not set, setting.
```

You can see that no errors were produced in the console, indicating that all of the necessary tables and columns were created successfully. To check this, let's connect to the database again and list the tables:

```
// Run 'psql' to connect to the database
[james@linux cs-coursework]$ psql -h localhost -U fosscat -W
// Enter the password (it is not displayed)
Password:
psql (15.4, server 16.1 (Debian 16.1-1.pgdg120+1))
WARNING: psql major version 15, server major version 16.
        Some psql features might not work.
Type "help" for help.

// List all tables, we can see that they exist
fosscat=# \dt
         List of relations
 Schema | Name | Type | Owner
--------+-----------+-------+---------
 public | checkouts | table | fosscat
 public | entities | table | fosscat
 public | items | table | fosscat
 public | users | table | fosscat
```

```
(4 rows)

// Quit
fosscat=# \q
```

We can see that the tables were created successfully.

Problem 2 - Operator does not exist

However, when attempting to create a new user in the database I encountered another error:

```
./backend/util/user.go:14 ERROR: operator does not exist: text == unknown (SQLSTATE 42883)

[0.369ms] [rows:0] SELECT "id" FROM "users" WHERE id == '8596a222-930e-42f0-841e-9d95993668a4'
    AND "users"."deleted_at" IS NULL ORDER BY "users"."id" LIMIT 1
```

This error states that it cannot compare `id` with the given UUID (`'8596a222-930e-42f0-841e-9d95993668a4'`) because the operator `==` does not exist. This error points to my `IsUuidFree` function, which I talked about in the **User account creation** section of this iteration.

After some research, I found that the error occurs because the operator `==` does not exist in PostgreSQL. `==` is commonly used in programming languages to perform a deep comparison of two objects, and I assumed that the same would be true for PostgreSQL. Interestingly, the issue did not manifest itself until after the switch to PostgreSQL, meaning that SQLite handles `==` as I expected. The fix was simple; Change `'=='` to `'='`.

This meant that the line

```
err := db.Model(obj).Select("id").Where("id == ?", id.String()).First(&obj).Error
```

became

```
err := db.Model(obj).Select("id").Where("id = ?", id.String()).First(&obj).Error
```

Problem 3 - "... violates foreign key constraint"

When trying to create a new user, the following error message would appear:

```
/backend/graph/resolver/mutation.go:90 ERROR: insert or update on table "users" violates foreign
    key constraint "fk_checkouts_user" (SQLSTATE 23503)
[2.691ms] [rows:0] INSERT INTO "users" ("id", "created_at", "updated_at", "deleted_at",
    "first_name", "last_name", "email", "hash") VALUES [..]
```

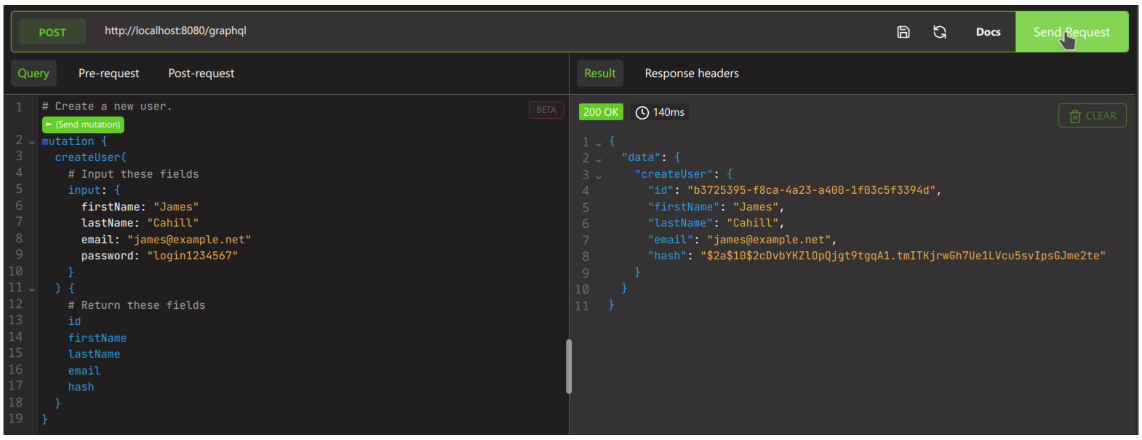
This error occurred when trying to create a user. Upon reading into it the error occurred because of how I defined my foreign keys for GORM. For example, I had the following struct definition for `Checkout`:

```
type Checkout struct {
    gorm.Model
    ID uuid.UUID
    User User `gorm:"foreignKey:ID"`
    TakeDate time.Time
    ReturnDate time.Time
}
```

Upon reading the GORM documentation, I realised this should actually be:

```
// Checkout belongs to a User, User.ID (UserID) is the foreign key
type Checkout struct {
    gorm.Model
    ID uuid.UUID
    User User
    UserID uuid.UUID
    TakeDate time.Time
    ReturnDate time.Time
}
```

After applying this change, the program worked as expected. I tested everything by attempting to create a new user in Altair:



```
# Create a new user.
mutation {
  createUser(
    # Input these fields
    input: {
      firstName: "James"
      lastName: "Cahill"
      email: "james@example.net"
      password: "login1234567"
    }
  ) {
    # Return these fields
    id
    firstName
    lastName
    email
    hash
  }
}
```

```
{
  "data": {
    "createUser": {
      "id": "b3725395-f8ca-4a23-a400-1f03c5f3394d",
      "firstName": "James",
      "lastName": "Cahill",
      "email": "james@example.net",
      "hash": "$2a$10$2cDvbYKZlOpQjgt9tgqA1.tmITKjrwGh7Ue1LVcu5svIpsGJme2te"
    }
  }
}
```

This query returned successfully and the user was created without any errors.

3.1.4 Adding foreign keys for lists

3.1.5 Issues with nested queries (queries using multiple tables)

3.1.6 Adding the remaining queries

CreateCheckout validation

3.1.7 Testing

Test Plan

My plan for testing this iteration was to create unit tests for my project. Unit tests are an automated set of tests designed to ensure that the tested application works correctly.

After some research, I decided to use the testify testing framework, which is a toolkit with assertions and mocking support that works well with standard go functions. I also decided that it would be beneficial to have a testing database, so I decided to create a script that spins up an ephemeral (short-lasting) database container using the *testcontainers* package. This database will only run for the duration of the test suite.

A snippet of this script is included below: (should I just remove this from the writeup? not *really* needed)

```
// excerpt of backend/test/common/database.go

// SetupSuite is called before any tests run
func (s *DatabaseTestSuite) SetupSuite() {
    s.dbCtx = context.Background()
    // Create a container request for a container running the "postgres" image
    req := testcontainers.ContainerRequest{
        Image: "postgres", // container image to use
        ExposedPorts: []string{"5432"}, // ports to expose to host
        WaitingFor: wait.ForLog("database system is ready to accept
            connections").WithOccurrence(2), // trigger to define when container has started up
        Env: map[string]string{ // environment variables
```

```
            "POSTGRES_DB": "fosscat",
            "POSTGRES_USER": "fosscat",
            "POSTGRES_PASSWORD": "fosscat",
        },
    }

    // Note that we do not define any persistent storage so the database will start from scratch
        every time it is created.

    // Create the container **and** wait for it to start up
    dbContainer, err := testcontainers.GenericContainer(s.dbCtx,
        testcontainers.GenericContainerRequest{
        ContainerRequest: req, // specify the container request
        Started: true, // automatically start once created
    })

    s.dbContainer = dbContainer

    if err != nil {
        panic(err)
    }

    // Determine the IP address of the container
    ctrIp, _ := dbContainer.ContainerIP(s.dbCtx)

    // Log that the database is now available
    log.Default().Printf("[test/common/database]: ephemeral db available on %s:5432", ctrIp)

    // Define connection details for the database
    dsn := fmt.Sprintf(
        "host=%s user=fosscat password=fosscat dbname=fosscat port=5432", ctrIp,
    )

    // Attempt to connect to the db
    db, err2 := gorm.Open(postgres.Open(dsn), &gorm.Config{})

    if err2 != nil {
        panic(err)
    }

    // Call migrate function (defined in backend/database/database.go) to create tables
    database.Migrate(db)

    // make the GORM instance available to tests
    s.DB = db

}
```

After setting this up, I create TestSuites that could be inherited from in order to reduce code duplication. I first created the following suites:

- `DatabaseTestSuite` - A test suite that provides the ephemeral database as seen above
    - `UserDatabaseTestSuite` - Handles creating user accounts for child test suites to use, includes assertions and it's own user tests.
    - `EntityDatabaseTestSuite` - Handles creating entities for child test suites use, includes assertions and it's own entity tests.

Now the ground-work was in place, it was time to write the unit tests themselves.
I started with User and Entity tests first, placing them in their respective suites. I used a **code coverage** tool to ensure that every line of code was covered. This means that the

complete functionality of a function or line of code was tested in my tests. For example, with an if statement, both outcomes must be tested for.

When creating tests, it was important to ensure that all combinations of inputs were tested for. For example, for **Checkouts** I created tests for:
// INCLUDE EXAMPLES OF TESTS

- Checkout creation
  - With all fields persistent
  - Without a return date
  - Without a take date
  - With only the user ID field
- Checkout deletion (TBD)
  - Deleting a non-existent checkout
  - Deleting an existing checkout

This allowed me to ensure that all parts of my program were tested properly.
TODO. Rewrite this I am repeating myself quite a bit.

Test Results
My unit testing uncovered four problems:

- (User creation): Email validation is broken
- (User creation): Password can only be 72 characters
- (Checkout creation): Take and Return dates not being stored in the database
- (Item creation): Item title not being stored in the database

Errors encountered during testing

Issues with user creation
I am using bcrypt for password hashing. When testing user creation with long passwords, I noticed that tests would fail with errors when the password was greater than 72 bytes (characters). Some further research revealed that this is a limitation of bcrypt itself. To avoid the unintentional behaviour caused by these errors, I added validation for the password length. If the validation is not successful, the user will be presented with a descriptive error and the user will not be added to the database.

After fixing this issue, I noticed that whilst writing tests for invalid e-mail addresses that the email address always appeared to be valid, even when it shouldn't. I am using the go `emailVerifier` library to validate email strings passed to it, and it turned out I was calling the library incorrectly. After reading the docs for the library, I elected to use the much simpler `ParseAddress` function instead of the more complicated `VerifyEmail` function. The VerifyEmail function attempts to connect to the specified domain in the email address string to confirm that it can receive emails, which is overkill for my use. Instead I will be using the ParseAddress function which merely applies a regular expression to ensure that the email has the correct syntax.

Issues with checkout creation // INCLUDE DB SCREENSHOTS?

When creating tests for checkouts, I noticed whilst using a database viewer that I was missing data in the database after creating a checkout. Specifically, the `TakeDate` and `ReturnDate` fields would always be empty. Upon taking a closer look at my code, it was soon obvious why this was the case.

```go
// Truncated from the original

func CreateCheckout(db *gorm.DB, input model.NewCheckout) (*structs.Checkout, error) {
    // Create a Checkout struct
    checkout := structs.Checkout{}

    // Set checkout.User to match the user object we queried from the database
    checkout.User = *user

    // Create the database entry
    db.Create(&checkout)

    // Return the entry and no error (nil)
    return &checkout, nil
}
```

As you can see, `TakeDate` and `ReturnDate` were never set in the `checkout` struct, so when we called `db.Create(&checkout)`, these values were `nil`. However, the fix was not as simple as setting `TakeDate` and `ReturnDate`, as these fields are optional during creation. As the user cannot be sure about either the take or return dates when creating a checkout, they are marked as optional so that they do not have to be specified during checkout creation, instead allowing the user to edit the record and add them at a later date.

Issues with item creation

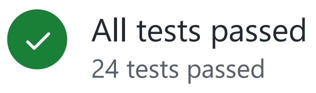
Similar to before, when creating tests for my items, I noticed in the database viewer that the Title field would always be empty, even if it was passed to my CreateItem function. Similarly to before, I had forgotten to set the item name in the struct before adding the struct to the database. This was a simple fix, where I only needed to add three lines:

```go
func CreateItem(db *gorm.DB, input model.NewItem) (*structs.Item, error) {
    // Create a Item struct
    item := structs.Item{}

    // [!] I added these three lines to set the title
    if input.Title != nil {
        item.Title = *input.Title
    }

    // [truncated]
}
```

Summary

After fixing these errors, I was able to create the rest of my tests.

As my code is hosted on GitHub, I setup their Continuous Integration (CI) service to automatically run my test suite every time I committed to the repository. After setting this up (not going to detail here as is not relevant?), I was able to queue the test and was greeted with the following result:


All tests passed
24 tests passed

Here is an excerpt from one of the test runs, showing the output from one of the test suites:

```
=== RUN TestUserTestSuite
=== RUN TestUserTestSuite/TestCreateUser
=== RUN TestUserTestSuite/TestCreateUserInvalidEmail
=== RUN TestUserTestSuite/TestCreateUserInvalidPasswordTooLong
=== RUN TestUserTestSuite/TestGetAllUsers
--- PASS: TestUserTestSuite (6.21s)
    --- PASS: TestUserTestSuite/TestCreateUser (0.91s)
    --- PASS: TestUserTestSuite/TestCreateUserInvalidEmail (0.00s)
    --- PASS: TestUserTestSuite/TestCreateUserInvalidPasswordTooLong (0.00s)
    --- PASS: TestUserTestSuite/TestGetAllUsers (0.00s)
```

As we can see both from the test summary generated by GitHub and the output of the test runs themselves, all of the tests are passing correctly.

// show code coverage?

Fixing error XYZ

3.1.8 Evaluation

In this iteration, I have successfully written code that can create a database, create the necessary tables to store data, and exposes functions to create and store data in the database.

Using GraphQL, user accounts can be created, where a unused unique identifier is found and used, as well as the creation of a salted password hash to securely store the user's password. In addition, checkouts, entities and items can be created each with their unique identifiers. Where necessary, such as with an checkout, we can use relationships to tie the checkout to a specified user object residing in the database.

The next iteration will focus on adding security to the backend. For example, I will need to add authenticated routes, where users can only view the contents of the route if they are logged in, as it will contain sensitive data.

4 Testing

5 Evaluation

6 Code Listings

Listings

6.1 Backend

Listing 1: ../../backend/database/checkout_test.go

```go
package database_test

import (
    "testing"
    "time"

    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/test/common"
    "github.com/stretchr/testify/suite"
)

type CheckoutTestSuite struct {
    common.UserDatabaseTestSuite
}

func (s *CheckoutTestSuite) SetupSuite() {
    s.UserDatabaseTestSuite.SetupSuite() // Call parent SetupSuite
    s.CreateUser() // Create a user for this test suite run
}

func (s *CheckoutTestSuite) TestCreateCheckoutAllFields() {
    existingUser := model.ExistingUser{ID: s.User.ID.String()}
    takeDate := time.Now()
    returnDate := time.Now().AddDate(0, 0, 1)
    newCheckout := model.NewCheckout{User: &existingUser, TakeDate: &takeDate, ReturnDate:
        &returnDate}

    checkout, err := database.CreateCheckout(s.DB, newCheckout)

    s.Assertions.False(checkout.TakeDate.IsZero())
    s.Assertions.False(checkout.ReturnDate.IsZero())

    s.Assertions.Equal(checkout.User.ID, s.User.ID)
    s.Assertions.Equal(checkout.UserID, s.User.ID)

    s.Assertions.Nil(err)
}

func (s *CheckoutTestSuite) TestCreateCheckoutNoReturnDate() {

    existingUser := model.ExistingUser{ID: s.User.ID.String()}
    takeDate := time.Now()
    newCheckout := model.NewCheckout{User: &existingUser, TakeDate: &takeDate, ReturnDate: nil}

    checkout, err := database.CreateCheckout(s.DB, newCheckout)

    s.Assertions.False(checkout.TakeDate.IsZero())
    s.Assertions.True(checkout.ReturnDate.IsZero())

    s.Assertions.Equal(checkout.User.ID, s.User.ID)
    s.Assertions.Equal(checkout.UserID, s.User.ID)

    s.Assertions.Nil(err)
}

func (s *CheckoutTestSuite) TestCreateCheckoutNoTakeDate() {

    existingUser := model.ExistingUser{ID: s.User.ID.String()}
    returnDate := time.Now().AddDate(0, 0, 1)
    newCheckout := model.NewCheckout{User: &existingUser, TakeDate: nil, ReturnDate: &returnDate}

    checkout, err := database.CreateCheckout(s.DB, newCheckout)

    s.Assertions.True(checkout.TakeDate.IsZero())
    s.Assertions.False(checkout.ReturnDate.IsZero())

    s.Assertions.Equal(checkout.User.ID, s.User.ID)
    s.Assertions.Equal(checkout.UserID, s.User.ID)

    s.Assertions.Nil(err)
}

func (s *CheckoutTestSuite) TestCreateCheckoutOnlyID() {

    existingUser := model.ExistingUser{ID: s.User.ID.String()}
    newCheckout := model.NewCheckout{User: &existingUser, TakeDate: nil, ReturnDate: nil}

    checkout, err := database.CreateCheckout(s.DB, newCheckout)

    s.Assertions.True(checkout.TakeDate.IsZero())
    s.Assertions.True(checkout.ReturnDate.IsZero())

    s.Assertions.Equal(checkout.User.ID, s.User.ID)
    s.Assertions.Equal(checkout.UserID, s.User.ID)

    s.Assertions.Nil(err)
}

// In order for 'go test' to run this suite, we need to create
// a normal test function and pass our suite to s.Run
func TestCheckoutTestSuite(t *testing.T) {
    suite.Run(t, new(CheckoutTestSuite))
}
```

Listing 2: ../../backend/database/checkout.go

```
package database

import (
    "errors"

    "github.com/google/uuid"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/structs"
    "github.com/jcxldn/fosscat/backend/util"
    "gorm.io/gorm"
)

func CreateCheckout(db *gorm.DB, input model.NewCheckout) (*structs.Checkout, error) {
    // Create a Checkout struct
    // Populate the TakeDate and ReturnDate fields using a util that sets them to zero-time if
         they are nil or not provided.
    checkout := structs.Checkout{TakeDate: util.GetTimeOrZero(input.TakeDate), ReturnDate:
         util.GetTimeOrZero(input.ReturnDate)}

    isFreeUuid := false
    for !isFreeUuid {
        // Generate a UUID for the checkout id.
        checkout.ID = uuid.New()
        // Check that the UUID has not been used already
        // If true, it will break out of this for loop and continue.
        isFreeUuid = util.IsUuidFree[structs.Checkout](db, checkout.ID)
    }

    // When we get here, we have found a non-used UUID.

    // Assign the checkout to a user.
    id, parseErr := uuid.Parse(input.User.ID)

    if parseErr == nil {

        user, dbErr := util.GetObjectById[structs.User](db, id)

        if dbErr == nil {
            checkout.User = *user

            // Create the database entry
            db.Create(&checkout)

            return &checkout, nil
        } else {
            return nil, errors.New("unable to create checkout: user does not exist")
        }
    } else {
        return nil, errors.New("unable to create checkout: unable to parse user id")
    }
}
```

Listing 3: `../../backend/database/database.go`

```
package database

import (
    "fmt"
    "log"
    "os"

    "gorm.io/driver/postgres"
    "gorm.io/gorm"

    "github.com/jcxldn/fosscat/backend/structs"
)

func Connect() *gorm.DB {
    dsn := fmt.Sprintf(
        "host=%s user=%s password=%s dbname=%s port=%s",
        os.Getenv("DB_HOST"),
        os.Getenv("DB_USER"),
        os.Getenv("DB_PASS"),
        os.Getenv("DB_NAME"),
        os.Getenv("DB_PORT"),
    )
    // Attempt to connect to the db
    db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
    if err != nil {
        panic("failed to connect database")
    }

    Migrate(db)

    return db
}

func Migrate(db *gorm.DB) {
    log.Println("[database] migrating...")

    var errors [4]error

    // "Migrate" the schema
    // This will create tables, keys, columns, etc. Everything really.
    // See https://gorm.io/docs/migration.html
    // Note that we need to pass each struct in our schema.

    errors[0] = db.AutoMigrate(&structs.Checkout{})
    // Item has a dependency on Entity, so do them in the correct order
    // to avoid "relation does not exist" error during table creation.
    errors[1] = db.AutoMigrate(&structs.Entity{})
    errors[2] = db.AutoMigrate(&structs.Item{})
    errors[3] = db.AutoMigrate(&structs.User{})

    for index, err := range errors {
        if err != nil {
            log.Panicf("Error with migrate call %d: '%s'", index, err)
        }
    }

    log.Println("[database] migrated, done.")
}
```

Listing 4: ../../backend/database/entity_test.go

```
package database_test

import (
    "testing"

    "github.com/jcxldn/fosscat/backend/test/common"
    "github.com/stretchr/testify/suite"
)

type EntityTestSuite struct {
    common.EntityDatabaseTestSuite
}

func (s *EntityTestSuite) TestCreateEntity() {
    s.CreateEntity()
}
```

```
func TestEntityTestSuite(t *testing.T) {
    suite.Run(t, new(EntityTestSuite))
}
```

Listing 5: ../../backend/database/entity.go

```
package database

import (
    "github.com/google/uuid"
    "github.com/jcxldn/fosscat/backend/structs"
    "github.com/jcxldn/fosscat/backend/util"
    "gorm.io/gorm"
)

func CreateEntity(db *gorm.DB) (*structs.Entity, error) {
    // Create a Entity struct
    entity := structs.Entity{}

    isFreeUuid := false
    for !isFreeUuid {
        // Generate a UUID for the user id.
        entity.ID = uuid.New()
        // Check that the UUID has not been used already
        // If true, it will break out of this for loop and continue.
        isFreeUuid = util.IsUuidFree[structs.Entity](db, entity.ID)
    }

    db.Create(&entity)

    return &entity, nil
}
```

Listing 6: ../../backend/database/item_test.go

```
package database_test

import (
    "testing"

    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/structs"
    "github.com/jcxldn/fosscat/backend/test/common"
    "github.com/stretchr/testify/suite"
)

type ItemTestSuite struct {
    common.EntityDatabaseTestSuite
    item *structs.Item
}

func (s *EntityTestSuite) TestCreateItemEmpty() {
    newItem := model.NewItem{}
    item, err := database.CreateItem(s.DB, newItem)

    s.Assertions.Empty(item.Entities)
    s.Assertions.Nil(err)
}
```

```
func (s *EntityTestSuite) TestCreateItemNameOnly() {
    title := "Example Item"
    newItem := model.NewItem{Title: &title}
    item, err := database.CreateItem(s.DB, newItem)

    s.Assertions.Equal(item.Title, title)
    s.Assertions.Empty(item.Entities)
    s.Assertions.Nil(err)
}

func (s *EntityTestSuite) TestCreateItemSingleEntityOnly() {
    s.CreateEntity()

    existingEntity := model.ExistingEntity{ID: s.Entity.ID.String()}
    existingEntities := []*model.ExistingEntity{}
    existingEntities = append(existingEntities, &existingEntity)

    newItem := model.NewItem{Entities: existingEntities}
    item, err := database.CreateItem(s.DB, newItem)

    s.Assertions.Equal(item.Entities[0].ID, s.Entity.ID)
    s.Assertions.Len(item.Entities, 1)
    s.Assertions.Nil(err)
}

func TestItemTestSuite(t *testing.T) {
    suite.Run(t, new(ItemTestSuite))
}
```

Listing 7: `../../backend/database/item.go`

```
package database

import (
    "errors"

    "github.com/google/uuid"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/structs"
    "github.com/jcxldn/fosscat/backend/util"
    "gorm.io/gorm"
)

func CreateItem(db *gorm.DB, input model.NewItem) (*structs.Item, error) {
    // Create a Item struct
    item := structs.Item{}

    if input.Title != nil {
        item.Title = *input.Title
    }

    isFreeUuid := false
    for !isFreeUuid {
        // Generate a UUID for the item id
        item.ID = uuid.New()
        // Check that the UUID has not been used already
        // If true, will break out of this for loop and continue
        isFreeUuid = util.IsUuidFree[structs.Item](db, item.ID)
    }

    // When we get here, we have found a non-used UUID.
```

```go
    // MAJOR STEP: Assign any given entities to the item

    // Check if we were given any entity IDs and iterate over them
    // First variable is index, but we are not using it so put _. (standard practice)
    for _, entityId := range input.Entities {
        // Attempt to parse this uuid
        id, parseErr := uuid.Parse(entityId.ID)

        if parseErr == nil {
            // UUID is valid, attempt to get a object by that ID
            entity, dbErr := util.GetObjectById[structs.Entity](db, id)

            if dbErr == nil {
                // Object exists
                item.Entities = append(item.Entities, *entity)
            } else {
                // Object does not exist
                // Let's cancel the whole transaction.
                // TODO: return problematic UUID
                return nil, errors.New("unable to create item: entity does not exist")
            }
        } else {
            // entityId is not a valid UUID
            // Let's cancel the whole transaction.
            // TODO: return problematic UUID
            return nil, errors.New("unable to create item: unable to parse entity id")
        }
    }

    // All entities have been parsed and added successfully
    // If error, transaction would have been cancelled.

    // Create the db entry.
    db.Create(&item)

    // Return the result
    return &item, nil
}
```

Listing 8: `../../backend/database/user_test.go`

```go
package database_test

import (
    "testing"

    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/test/common"
    "github.com/stretchr/testify/suite"
)

type UserTestSuite struct {
    common.UserDatabaseTestSuite
}

func (s *UserTestSuite) TestCreateUser() {
    s.CreateUser()
}

func (s *UserTestSuite) TestCreateUserInvalidEmail() {
    newUser := model.NewUser{FirstName: "Example", LastName: "User", Email: "example*email"}
    _, err := database.CreateUser(s.DB, newUser)

    s.Assertions.EqualError(err, "email address not valid")
}

func (s *UserTestSuite) TestCreateUserInvalidPasswordTooLong() {
    // >72 byte password is too long (72 byte limit)
    newUser := model.NewUser{Email: "example.user@example.com", Password: "Lorem ipsum dolor sit
         amet, consectetur adipiscing elit. Nunc tincidunt. "}
    _, err := database.CreateUser(s.DB, newUser)

    s.Assertions.EqualError(err, "password does not satisfy requirements")
}

func (s *UserTestSuite) TestGetAllUsers() {
    users, err := database.GetAllUsers(s.DB)

    // Assert that there was no error
    s.Assertions.Empty(err)
    // Check that the user created in this test is in the response
    s.Assertions.Len(users, 1)
    s.Assertions.Equal(users[0].ID, s.User.ID)

}

// In order for 'go test' to run this suite, we need to create
// a normal test function and pass our suite to s.Run
func TestUserTestSuite(t *testing.T) {
    suite.Run(t, new(UserTestSuite))
}
```

Listing 9: `../../backend/database/user.go`

```
package database

import (
    "errors"

    "github.com/google/uuid"
    ev "github.com/jcxldn/fosscat/backend/emailVerifier"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/structs"
    "github.com/jcxldn/fosscat/backend/util"
    "golang.org/x/crypto/bcrypt"
    "gorm.io/gorm"
)

func CreateUser(db *gorm.DB, input model.NewUser) (*structs.User, error) {
    // Create a User struct from the input model
    // Set fields that do not need further validation (name only)
    user := structs.User{FirstName: input.FirstName, LastName: input.LastName}

    // Attempt to validate the user email.
    isValidEmail := ev.VerifyEmail(input.Email)
    if !isValidEmail {
        return nil, errors.New("email address not valid")
    }

    // Email passed validation, set in the user struct.
    user.Email = input.Email

    isFreeUuid := false
    for !isFreeUuid {
        // Generate a UUID for the user id.
        user.ID = uuid.New()
        // Check that the UUID has not been used already
        // If true, it will break out of this for loop and continue.
        isFreeUuid = util.IsUuidFree[structs.User](db, user.ID)
    }

    // Salt and hash the provided password
    // I am currently using bCrypt, which has the function GenerateFromPasswords.
    // This generates a random salt for us and applies it to the hash.
    // I believe the hash result and salt are stored side by side.
    // Cost of '10' for now, seems like a good balance.
    hash, err := bcrypt.GenerateFromPassword([]byte(input.Password), 10)

    if err != nil {
        // Hashing failed (password too long/short?)
        // TODO: make errors more readable, custom errors
        return nil, errors.New("password does not satisfy requirements")
    }

    // Hashing completed successfully, set in the user struct
    // For purposes of writeup use string for now so we can remove it later.
    user.Hash = string(hash)

    db.Create(&user)
    return &structs.User{ID: user.ID, FirstName: user.FirstName, LastName: user.LastName, Email:
         user.Email, Hash: user.Hash}, nil
}

func GetAllUsers(db *gorm.DB) ([]*structs.User, error) {
    // Get all users. Proof of concept only, returns all fields!
    users := []*structs.User{}
    result := db.Find(&users)
    return users, result.Error
}
```

Listing 10: ../../backend/emailVerifier/emailVerifier.go

```
package emailVerifier

import (
    emailverifier "github.com/AfterShip/email-verifier"
)

var (
    ev = emailverifier.NewVerifier()
)

func VerifyEmail(email string) bool {
    // Check syntax
    syntax := ev.ParseAddress(email)

    return syntax.Valid
}
```

Listing 11: ../../backend/graph/resolver/checkout.go

```
package resolver
```

```go
// This file will be automatically regenerated based on the schema, any resolver implementations
// will be copied through when generating and any unknown code will be moved to the end.
// Code generated by github.com/99designs/gqlgen version v0.17.40

import (
	"context"

	"github.com/jcxldn/fosscat/backend/graph"
	"github.com/jcxldn/fosscat/backend/structs"
)

// ID is the resolver for the id field.
func (r *checkoutResolver) ID(ctx context.Context, obj *structs.Checkout) (string, error) {
	return obj.ID.String(), nil
}

// Checkout returns graph.CheckoutResolver implementation.
func (r *Resolver) Checkout() graph.CheckoutResolver { return &checkoutResolver{r} }

type checkoutResolver struct{ *Resolver }
```

Listing 12: ../../backend/graph/resolver/entity.go

```go
package resolver

// This file will be automatically regenerated based on the schema, any resolver implementations
// will be copied through when generating and any unknown code will be moved to the end.
// Code generated by github.com/99designs/gqlgen version v0.17.40

import (
	"context"

	"github.com/jcxldn/fosscat/backend/graph"
	"github.com/jcxldn/fosscat/backend/structs"
)

// ID is the resolver for the id field.
func (r *entityResolver) ID(ctx context.Context, obj *structs.Entity) (string, error) {
	return obj.ID.String(), nil
}

// Entity returns graph.EntityResolver implementation.
func (r *Resolver) Entity() graph.EntityResolver { return &entityResolver{r} }

type entityResolver struct{ *Resolver }
```

Listing 13: ../../backend/graph/resolver/item.go

```go
package resolver

// This file will be automatically regenerated based on the schema, any resolver implementations
// will be copied through when generating and any unknown code will be moved to the end.
// Code generated by github.com/99designs/gqlgen version v0.17.40

import (
	"context"

	"github.com/jcxldn/fosscat/backend/graph"
	"github.com/jcxldn/fosscat/backend/structs"
)
```

```go
// ID is the resolver for the id field.
func (r *itemResolver) ID(ctx context.Context, obj *structs.Item) (string, error) {
    return obj.ID.String(), nil
}

// Item returns graph.ItemResolver implementation.
func (r *Resolver) Item() graph.ItemResolver { return &itemResolver{r} }

type itemResolver struct{ *Resolver }
```

Listing 14: ../../backend/graph/resolver/mutation.go

```go
package resolver

// This file will be automatically regenerated based on the schema, any resolver implementations
// will be copied through when generating and any unknown code will be moved to the end.
// Code generated by github.com/99designs/gqlgen version v0.17.40

import (
    "context"
    "errors"

    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/graph"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/structs"
    "github.com/jcxldn/fosscat/backend/util/jwt"
    "golang.org/x/crypto/bcrypt"
)

// CreateCheckout is the resolver for the createCheckout field.
func (r *mutationResolver) CreateCheckout(ctx context.Context, input model.NewCheckout)
     (*structs.Checkout, error) {
    return database.CreateCheckout(r.db, input)
}

// CreateEntity is the resolver for the createEntity field.
func (r *mutationResolver) CreateEntity(ctx context.Context) (*structs.Entity, error) {
    return database.CreateEntity(r.db)
}

// CreateItem is the resolver for the createItem field.
func (r *mutationResolver) CreateItem(ctx context.Context, input model.NewItem) (*structs.Item,
     error) {
    return database.CreateItem(r.db, input)
}

// CreateUser is the resolver for the createUser field.
func (r *mutationResolver) CreateUser(ctx context.Context, input model.NewUser) (*structs.User,
     error) {
    return database.CreateUser(r.db, input)
}

// Login is the resolver for the login field.
func (r *mutationResolver) Login(ctx context.Context, email string, password string)
     (*model.LoginResponse, error) {
    // Create a user struct
    user := structs.User{Email: email}

    // Fetch the user from the database
    // SELECT * FROM users ORDER BY id LIMIT 1;
    // validation for only one uid per email needs to be done in createUser.
    r.db.First(&user)

    res := bcrypt.CompareHashAndPassword([]byte(user.Hash), []byte(password))

    if res == nil {
        // Password matches, generate and return a JWT
        lr := model.LoginResponse{Success: true}
        jwt, err := jwt.NewJwt(user)
        if err == nil {
            lr.Jwt = &jwt
            return &lr, nil
        } else {
            lr := model.LoginResponse{Success: false}
            return &lr, errors.New("Error creating jwt")
        }

    } else {
        // Password does not match (bcrypt returns error on failure)
        lr := model.LoginResponse{Success: false}
        return &lr, nil
    }
}

// Mutation returns graph.MutationResolver implementation.
func (r *Resolver) Mutation() graph.MutationResolver { return &mutationResolver{r} }

type mutationResolver struct{ *Resolver }
```

Listing 15: ../../backend/graph/resolver/query.go

```
package resolver

// This file will be automatically regenerated based on the schema, any resolver implementations
// will be copied through when generating and any unknown code will be moved to the end.
// Code generated by github.com/99designs/gqlgen version v0.17.40

import (
    "context"

    "github.com/jcxldn/fosscat/backend/graph"
    "github.com/jcxldn/fosscat/backend/structs"
    "gorm.io/gorm/clause"
)

// Checkout is the resolver for the checkout field.
func (r *queryResolver) Checkout(ctx context.Context) ([]*structs.Checkout, error) {
    // Get all checkouts. Proof of concept only, returns all fields!
    checkouts := []*structs.Checkout{}
    // Preload all associations (https://gorm.io/docs/preload.html#Preload-All)
    // TODO much later: don't preload private fields like hash unless user = logged in user.
    result := r.db.Preload(clause.Associations).Find(&checkouts)
    return checkouts, result.Error
}

// Entity is the resolver for the entity field.
func (r *queryResolver) Entity(ctx context.Context) ([]*structs.Entity, error) {
    // Get all entities. Proof of concept only, returns all fields!
    entities := []*structs.Entity{}
    result := r.db.Find(&entities)
    return entities, result.Error
}
```

```
// Item is the resolver for the item field.
func (r *queryResolver) Item(ctx context.Context) ([]*structs.Item, error) {
    // Get all items. Proof of concept only, returns all fields!
    items := []*structs.Item{}
    result := r.db.Find(&items)
    return items, result.Error
}

// Users is the resolver for the users field.
func (r *queryResolver) Users(ctx context.Context) ([]*structs.User, error) {
    // Get all users. Proof of concept only, returns all fields!
    users := []*structs.User{}
    result := r.db.Find(&users)
    return users, result.Error
}

// Query returns graph.QueryResolver implementation.
func (r *Resolver) Query() graph.QueryResolver { return &queryResolver{r} }

type queryResolver struct{ *Resolver }
```

Listing 16: `../../backend/graph/resolver/resolver.go`

```
package resolver

import (
    "log"

    "gorm.io/gorm"
)

// This file will not be regenerated automatically.
//
// It serves as dependency injection for your app, add any dependencies you require here.

type Resolver struct {
    db *gorm.DB
}

// Set DB object if not already set
func (r *Resolver) UpdateDb(db *gorm.DB) {
    if r.db == nil {
        log.Printf("[resolver] db not set, setting.")
        r.db = db
    } else {
        log.Printf("[resolver] WARN: db is already set, not overwriting.")
    }
}
```

Listing 17: `../../backend/graph/resolver/user.go`

```
package resolver

// This file will be automatically regenerated based on the schema, any resolver implementations
// will be copied through when generating and any unknown code will be moved to the end.
// Code generated by github.com/99designs/gqlgen version v0.17.40

import (
    "context"
    "github.com/jcxldn/fosscat/backend/graph"
    "github.com/jcxldn/fosscat/backend/structs"
)

// ID is the resolver for the id field.
func (r *userResolver) ID(ctx context.Context, obj *structs.User) (string, error) {
    return obj.ID.String(), nil
}

// User returns graph.UserResolver implementation.
func (r *Resolver) User() graph.UserResolver { return &userResolver{r} }

type userResolver struct{ *Resolver }
```

Listing 18: `../../backend/graph/schema/checkout.gql`

```
scalar Time

type Checkout {
    id: String!
    user: User!
    takeDate: Time
    returnDate: Time
}

input NewCheckout {
    user: ExistingUser!
    takeDate: Time
    returnDate: Time
}
```

Listing 19: `../../backend/graph/schema/entity.gql`

```
type Entity {
    id: String!
    # Array not-null, may be empty.
    checkouts: [Checkout]!
}

input ExistingEntity {
  id: String!
}
```

Listing 20: `../../backend/graph/schema/item.gql`

```
type Item {
    id: String!
    title: String
    # Array not-null, may be empty.
    entities: [Entity]!
}

input NewItem {
    title: String
    # Array not-null, may be empty.
    entities: [ExistingEntity]!
}
```

Listing 21: `../../backend/graph/schema/loginResponse.gql`

```
type LoginResponse {
    success: Boolean!
    jwt: String
}
```

Listing 22: `../../backend/graph/schema/mutation.gql`

```
type Mutation {
  createCheckout(input: NewCheckout!): Checkout!
  createEntity: Entity!
  createItem(input: NewItem!): Item!
  createUser(input: NewUser!): User!

  login(email: String!, password: String!): LoginResponse!
}
```

Listing 23: `../../backend/graph/schema/query.gql`

```
type Query {
  checkout: [Checkout!]!
  entity: [Entity!]!
  item: [Item!]!
  users: [User!]!
}
```

Listing 24: `../../backend/graph/schema/user.gql`

```
type User {
  id: String!
  firstName: String!
  lastName: String!
  email: String!
  hash: String
}

input NewUser {
  firstName: String!
  lastName: String!
  email: String!
  password: String!
}

input ExistingUser {
  id: String!
}
```

Listing 25: `../../backend/structs/checkout.go`

```
package structs

import (
    "time"

    "gorm.io/gorm"
    "github.com/google/uuid"
)

// Checkout belongs to a User, User.ID (UserID) is the foreign key
type Checkout struct {
    gorm.Model
    ID uuid.UUID `gorm:"type:uuid"`
    EntityID uuid.UUID `gorm:"type:uuid"` // Foreign key for Entity
    User User
    UserID uuid.UUID `gorm:"type:uuid"`
    TakeDate time.Time
    ReturnDate time.Time
}
```

Listing 26: ../../backend/structs/entity.go

```
package structs

import (
    "gorm.io/gorm"

    "github.com/google/uuid"
)

// Entity has many Checkouts, Checkout.EntityID is the foreign key
type Entity struct {
    gorm.Model
    ID uuid.UUID `gorm:"type:uuid"`
    ItemID uuid.UUID `gorm:"type:uuid"` // Foreign key for Item
    Checkouts []Checkout
}
```

Listing 27: ../../backend/structs/item.go

```
package structs

import (
    "gorm.io/gorm"

    "github.com/google/uuid"
)

// Item has many Entities, Entity.ItemID is the foreign key
type Item struct {
    gorm.Model
    ID uuid.UUID `gorm:"type:uuid"`
    Title string
    Entities []Entity
}
```

Listing 28: ../../backend/structs/user.go

```
package structs

import (
    "gorm.io/gorm"
    "github.com/google/uuid"
)

type User struct {
    gorm.Model
    ID uuid.UUID ‘gorm:"type:uuid"‘
    FirstName string
    LastName string
    Email string
    Hash string
}
```

Listing 29: ../../backend/test/common/database.go

```
package common

import (
    "context"
    "fmt"
    "log"

    "github.com/jcxldn/fosscat/backend/database"
    "github.com/stretchr/testify/suite"
    "github.com/testcontainers/testcontainers-go"
    "github.com/testcontainers/testcontainers-go/wait"
    "gorm.io/driver/postgres"
    "gorm.io/gorm"
)

// Define the test suite, absorbing the testify basic suite
type DatabaseTestSuite struct {
    suite.Suite
    DB *gorm.DB
    dbContainer testcontainers.Container
    dbCtx context.Context
}

// SetupSuite is called before any tests run
func (s *DatabaseTestSuite) SetupSuite() {
    s.dbCtx = context.Background()
    // Create a container request for a container running the "postgres" image
    req := testcontainers.ContainerRequest{
        Image: "postgres", // container image to use
        ExposedPorts: []string{"5432"}, // ports to expose to host
        WaitingFor: wait.ForLog("database system is ready to accept
             connections").WithOccurrence(2), // trigger to define when container has started up
        Env: map[string]string{ // environment variables
            "POSTGRES_DB": "fosscat",
            "POSTGRES_USER": "fosscat",
            "POSTGRES_PASSWORD": "fosscat",
        },
    }

    // Note that we do not define any persistent storage so the database will start from scratch
         every time it is created.

    // Create the container **and** wait for it to start up
    dbContainer, err := testcontainers.GenericContainer(s.dbCtx,
         testcontainers.GenericContainerRequest{
        ContainerRequest: req, // specify the container request
        Started: true, // automatically start once created
    })
    s.dbContainer = dbContainer

    if err != nil {
        panic(err)
    }

    // Determine the IP address of the container
    ctrIp, _ := dbContainer.ContainerIP(s.dbCtx)

    // Log that the database is now available
    log.Default().Printf("[test/common/database]: ephemeral db available on %s:5432", ctrIp)

    // Define connection details for the database
    dsn := fmt.Sprintf(
        "host=%s user=fosscat password=fosscat dbname=fosscat port=5432", ctrIp,
    )

    // Attempt to connect to the db
    db, err2 := gorm.Open(postgres.Open(dsn), &gorm.Config{})

    if err2 != nil {
        panic(err)
    }

    // Call migrate function (defined in backend/database/database.go) to create tables
    database.Migrate(db)

    // make the GORM instance available to tests
    s.DB = db

}

// TeardownSuite is called when all tests have finished
func (s *DatabaseTestSuite) TeardownSuite() {
    log.Default().Printf("[test/common/database]: teardown in progress")
    // Attempt to stop (terminate) the container, panicking if any errors are encountered.
    if err := s.dbContainer.Terminate(s.dbCtx); err != nil {
        panic(err)
    }
}
```

Listing 30: `../../backend/test/common/entity.go`

```
package common

import (
    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/structs"
)

type EntityDatabaseTestSuite struct {
    // inherit from DatabaseTestSuite
    DatabaseTestSuite
    Entity *structs.Entity
}

func (s *EntityDatabaseTestSuite) CreateEntity() {
    entity, err := database.CreateEntity(s.DB)

    if err != nil {
        panic(err)
    }

    s.Entity = entity

    s.Assertions.Nil(err)
}
```

Listing 31: ../../backend/test/common/user.go

```
package common

import (
    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/structs"
)

type UserDatabaseTestSuite struct {
    // inherit from DatabaseTestSuite
    DatabaseTestSuite
    User *structs.User
}

func (s *UserDatabaseTestSuite) CreateUser() {
    newUser := model.NewUser{FirstName: "Example", LastName: "User", Email: "user@example.com"}
    user, err := database.CreateUser(s.DB, newUser)

    if err != nil {
        panic(err)
    }

    s.User = user

    s.Assertions.Equal(user.FirstName, newUser.FirstName)
    s.Assertions.Equal(user.LastName, newUser.LastName)
    s.Assertions.Equal(user.Email, newUser.Email)
}
```

Listing 32: ../../backend/util/jwt/ecdsa.go

```
package jwt

import (
    "crypto/ecdsa"
    "crypto/elliptic"
    "crypto/rand"
    "crypto/x509"
    "encoding/pem"
    "log"
    "os"
)

var (
    key *ecdsa.PrivateKey
)

func SetupKey() {
    keyPath, exists := os.LookupEnv("JWT_PK_FILE")
    if !exists {
        log.Fatalln("[jwt/ecdsa] env var JWT_PK_FILE is not set.")
    } else {
        // getKey wil set the key variable
        getKey(keyPath)

        log.Println("[jwt/ecdsa] key loaded successfully")
    }
}

func loadKey(path string) {
    // Attempt to load the file
    dat, err := os.ReadFile(path)

    if err == nil {
        // Opened sucessfully
        block, _ := pem.Decode([]byte(dat))

        x509EncodedPrivKey := block.Bytes

        privateKey, err := x509.ParseECPrivateKey(x509EncodedPrivKey)

        if err != nil {
            log.Fatalln("[jwt/ecdsa] failed to parse keyfile")
        } else {
            key = privateKey
        }
    } else {
        log.Fatalln("[jwt/ecdsa] failed to open keyfile")
    }
}

func createKey(path string) {
    key, err := ecdsa.GenerateKey(elliptic.P384(), rand.Reader)
    if err != nil {
        log.Fatalln("[jwt/ecdsa] failed to create key")
    } else {
        bytes, marshalErr := x509.MarshalECPrivateKey(key)

        pemEncoded := pem.EncodeToMemory(&pem.Block{Type: "EC PRIVATE KEY", Bytes: bytes})

        if marshalErr != nil {
            log.Fatalln("[jwt/ecdsa] failed to marshal generated private key")
        }

        // Write the file, chmod 600 (owner r+w)
        writeErr := os.WriteFile(path, pemEncoded, 0600)

        if writeErr != nil {
            log.Fatalln("[jwt/ecdsa] failed to write generated key")
        }
    }
}

// Source: https://www.tutorialspoint.com/how-to-check-if-a-file-exists-in-golang
func fileExists(path string) bool {
    info, err := os.Stat(path)
    if os.IsNotExist(err) {
        return false
    } else {
        return !info.IsDir()
    }
}

// Returns true if successful, panics if not.
func getKey(path string) {
    if !fileExists(path) {
        createKey(path)
    }
    loadKey(path)
}
```

Listing 33: ../../backend/util/jwt/jwt_test.go

```
package jwt_test

import (
    "fmt"
    "os"
    "testing"

    "github.com/jcxldn/fosscat/backend/test/common"
    "github.com/jcxldn/fosscat/backend/util/jwt"
    "github.com/stretchr/testify/suite"
)

type UtilJwtTestSuite struct {
    common.UserDatabaseTestSuite
    jwt *string
    tempFilePath string
}

func (s *UtilJwtTestSuite) SetupSuite() {
    s.UserDatabaseTestSuite.SetupSuite() // Call parent SetupSuite
    s.CreateUser() // Create a user for this test suite run

    tempFilePath := fmt.Sprintf("%s/fosscat-test.pem", os.TempDir())

    fmt.Printf("Using temporary file '%s'", tempFilePath)

    s.tempFilePath = tempFilePath

    os.Setenv("JWT_PK_FILE", tempFilePath)
    jwt.SetupKey()
}

func (s *UtilJwtTestSuite) TeardownSuite() {
    fmt.Printf("Removing temporary file '%s'", s.tempFilePath)
}
func (s *UtilJwtTestSuite) TestCreateJwt() {
    token, err := jwt.NewJwt(*s.User)

    s.Assertions.Nil(err)

    s.jwt = &token
}

func (s *UtilJwtTestSuite) TestVerifyJwtValid() {
    token, err := jwt.NewJwt(*s.User)

    s.Assertions.Nil(err)

    parsedToken, claims, err2 := jwt.VerifyJwt(token, *s.User)

    s.Assertions.Nil(err2)

    s.Assertions.NotNil(parsedToken)
    s.Assertions.True(parsedToken.Valid)
    s.Assertions.Equal(claims.Subject, s.User.ID.String())

}

func (s *UtilJwtTestSuite) TestVerifyJwtInvalid() {

    parsedToken, claims, err := jwt.VerifyJwt("invalid jwt", *s.User)

    s.Assertions.NotNil(err)
    s.Assertions.Nil(parsedToken)
    s.Assertions.Nil(claims)
}

func TestUtilJwtTestSuite(t *testing.T) {
    suite.Run(t, new(UtilJwtTestSuite))
}
```

Listing 34: ../../backend/util/jwt/jwt.go

```
package jwt

import (
    "time"

    "github.com/golang-jwt/jwt/v5"
    "github.com/jcxldn/fosscat/backend/structs"
)

type Claims struct {
    jwt.RegisteredClaims
}

func getExpiryTime() time.Time {
    time := time.Now()
    time = time.AddDate(0, 0, 1) // one day
    return time
}

func NewJwt(user structs.User) (str string, err error) {
    claims := Claims{
        jwt.RegisteredClaims{
            Issuer: "fosscat",
            Subject: user.ID.String(),
            ExpiresAt: jwt.NewNumericDate(getExpiryTime()),
        },
    }
    token := jwt.NewWithClaims(jwt.SigningMethodES384, claims)

    str, err = token.SignedString(key)

    return
}

func VerifyJwt(jwtStr string, user structs.User) (token *jwt.Token, claims *Claims, err error) {
    token, err = jwt.ParseWithClaims(jwtStr, &Claims{}, func(token *jwt.Token) (interface{},
         error) {
        return &key.PublicKey, nil
    })

    if token == nil {
        return nil, nil, err
    }

    if claims, ok := token.Claims.(*Claims); ok && token.Valid {
        return token, claims, nil
    } else {
        return nil, nil, err
    }
}
```

Listing 35: ../../backend/util/get_test.go

```
package util_test

import (
    "testing"

    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/graph/model"
    "github.com/jcxldn/fosscat/backend/structs"
    "github.com/jcxldn/fosscat/backend/test/common"
    "github.com/jcxldn/fosscat/backend/util"
    "github.com/stretchr/testify/suite"
)

type UtilGetTestSuite struct {
    common.DatabaseTestSuite
    user *structs.User
}

func (s *UtilGetTestSuite) TestCreateUser() {
    newUser := model.NewUser{FirstName: "Example", LastName: "User", Email:
         "example.user@example.com"}
    user, err := database.CreateUser(s.DB, newUser)

    if err != nil {
        panic(err)
    }

    s.user = user
}

func (s *UtilGetTestSuite) TestGetObjectByIdUser() {
    user, err := util.GetObjectById[structs.User](s.DB, s.user.ID)

    if err != nil {
        panic(err)
    }

    s.Assertions.Equal(s.user.ID, user.ID)
}

func (s *UtilGetTestSuite) TestUuidIsFreeUser() {
    s.Assertions.False(util.IsUuidFree[structs.User](s.DB, s.user.ID))
}

// In order for 'go test' to run this suite, we need to create
// a normal test function and pass our suite to s.Run
func TestUtilGetTestSuite(t *testing.T) {
    suite.Run(t, new(UtilGetTestSuite))
}
```

Listing 36: `../../backend/util/get.go`

```go
package util

import (
    "github.com/google/uuid"
    "gorm.io/gorm"
)

func GetObjectById[T any](db *gorm.DB, id uuid.UUID) (*T, error) {
    obj := new(T)
    err := db.Model(obj).Where("id = ?", id.String()).First(&obj).Error

    return obj, err
}
```

Listing 37: `../../backend/util/time.go`

```go
package util

import (
    "time"
)

func GetTimeOrZero(t *time.Time) time.Time {
    if t == nil {
        return time.Time{}
    } else {
        return *t
    }
}
```

Listing 38: `../../backend/util/uuid.go`

```go
package util

import (
    "errors"

    "github.com/google/uuid"
    "gorm.io/gorm"
)

func IsUuidFree[T any](db *gorm.DB, id uuid.UUID) bool {
    obj := new(T)
    // NOTE: == works in SQLite, not in Postgres
    err := db.Model(obj).Select("id").Where("id = ?", id.String()).First(&obj).Error
    if errors.Is(err, gorm.ErrRecordNotFound) {
        // Record not found, so user id is free
        return true
    } else {
        // Record was returned successfully, therefore the user exists
        return false
    }
}
```

Listing 39: `../../backend/server.go`

```go
package main

import (
    "github.com/gin-contrib/cors"
    "github.com/gin-gonic/gin"

    "github.com/99designs/gqlgen/graphql/handler"
    "github.com/99designs/gqlgen/graphql/playground"
    "github.com/jcxldn/fosscat/backend/database"
    "github.com/jcxldn/fosscat/backend/graph"
    "github.com/jcxldn/fosscat/backend/graph/resolver"
    "github.com/jcxldn/fosscat/backend/util/jwt"
)

// Define the Graphql route handler
// based on https://gqlgen.com/recipes/gin/
func graphqlHandler() gin.HandlerFunc {
    // Connect to the database and place the handle in the graphql resolver
    // So that is accessible when executing graphql requests in ctx.
    resolver := &resolver.Resolver{}
    resolver.UpdateDb(database.Connect())

    // Setup JWT keys (load from file)
    jwt.SetupKey()

    // NewExecutableSchema and Config are in the generated.go file
    // Resolver is in the resolver.go file
    h := handler.NewDefaultServer(graph.NewExecutableSchema(graph.Config{Resolvers: resolver}))

    return func(c *gin.Context) {
        h.ServeHTTP(c.Writer, c.Request)
    }
}

// Define the Playground route handler
// based on https://gqlgen.com/recipes/gin/
func playgroundHandler() gin.HandlerFunc {
    h := playground.Handler("GraphQL", "/graphql")

    return func(c *gin.Context) {
        h.ServeHTTP(c.Writer, c.Request)
    }
}

// Define the Ping route handler
func pingHandler(c *gin.Context) {
    c.JSON(200, gin.H{
        "message": "pong",
    })
}

// Main function
func main() {
    // Create a gin engine instance
    r := gin.Default()

    r.Use(cors.New(cors.Config{
        AllowOrigins: []string{"*"},
        AllowHeaders: []string{"Content-Type"},
    }))

    // Define routes
    r.POST("/graphql", graphqlHandler()) // GraphQL query endpoint
    r.GET("/graphql/playground", playgroundHandler()) // GraphiQL playground
    r.GET("/ping", pingHandler) // Ping/pong endpoint (for healthcheck)

    r.Run() // Run on 0.0.0.0:8080
}
```

Listing 40: ../../backend/tools.go

```
//go:build tools
// +build tools

package tools

import (
    _ "github.com/99designs/gqlgen"
)
```

6.2 Frontend

Listing 41: ../../frontend/app/(app)/scan/_layout.tsx

```
import React from "react";
import { Button } from "react-native";
import { Stack, router } from "expo-router";

export default function ScanStack() {
    return (
        <Stack id="stack.scan">

            <Stack.Screen name="camera" options={{
                headerShown: false
            }} />

            <Stack.Screen name="result" options={{
                headerTitle: "Scan Result",
                headerBackButtonMenuEnabled: true,

                headerLeft: ({ label }) => (
                    <Button onPress={() => router.back()} title="Close" />
                ),
                presentation: "modal"

            }} />
        </Stack>
    )
}
```

Listing 42: ../../frontend/app/(app)/scan/camera.tsx

```
import React from "react";
import { View, Text } from "../../../components/Themed";
import { Button, StyleSheet } from "react-native";
import { useNavigation } from "expo-router";
import { BarcodeScanningResult, Camera, CameraView, useCameraPermissions } from
     "expo-camera/next";

const ScanPage = () => {
    // States
    const [permission, requestPermission] = useCameraPermissions();
    const [scanned, setScanned] = React.useState(false);

    // Hook to grab the parentt navigator (in this case navigator "stack.scan")
    const nav = useNavigation();

    // (fork): https://github.com/jcxldn/expo-camera-barcode-types-error/blob/main/app/modal.tsx
    if (!permission) {
        // Camera permissions are still loading
        return <View />;
    }

    // (fork): https://github.com/jcxldn/expo-camera-barcode-types-error/blob/main/app/modal.tsx
    if (!permission.granted) {
        // Camera permissions are not granted yet
        return (
            <View style={styles.container}>
                <Text style={{ textAlign: "center" }}>
                    We need your permission to show the camera
                </Text>
                <Button onPress={requestPermission} title="Request Permission" />
            </View>
        );
    }

    const handleScan = ({ type, data }: BarcodeScanningResult) => {
        setScanned(true)
        alert(‘Barcode scanned with type ${type} and data ${data}.‘)
        // Navigate to ./result
        alert(nav.getParent()?.navigate("result", { hello: "here" }))
    }

    return (
        <View style={StyleSheet.absoluteFillObject}>
            <Text>hi</Text>
            <CameraView style={StyleSheet.absoluteFillObject} onBarcodeScanned={scanned ?
                 undefined : handleScan} barcodeScannerSettings={{
                barCodeTypes: ["qr"]
            }} />
            {scanned && (
                <Button title="Tap to Scan Again" onPress={() => setScanned(false)} />
            )}
        </View>
    )
}

const styles = StyleSheet.create({
    container: {
        height: "100%",
        width: "100%"
    }
});

export default ScanPage;
```

Listing 43: ../../frontend/app/(app)/scan/result.tsx

```
import React from "react";
import { Text } from "../../../components/Themed";

export default function Bleh({ route }) {
    alert(JSON.stringify(route))
    return <Text>hi</Text>
}
```

Listing 44: `../../frontend/app/(app)/_layout.tsx`

```
import React from "react";

import { Redirect, Stack } from "expo-router";
import { useSession } from "../../components/AuthenticationContext";
import { Text } from "../../components/Themed";
import DrawerComponent from "../../components/Drawer";

const AppLayout = () => {
    const { jwt, isLoading } = useSession()

    if (isLoading) {
        return <Text>Loading...</Text>
    }

    if (!jwt) {
        return <Redirect href="/login" />
    }

    return (
        <DrawerComponent />
    )
}

export default AppLayout;
```

Listing 45: `../../frontend/app/(app)/index.tsx`

```
import React from "react";
import { Text } from "../../components/Themed";

const HomePage = () => {
    return (
        <>
            <Text>Hello, world!</Text>
        </>
    )
}

export default HomePage;
```

Listing 46: `../../frontend/app/_layout.tsx`

```
// Add better error handlers when using expo-dev-client
// https://docs.expo.dev/bare/install-dev-builds-in-bare/#add-better-error-handlers
import 'expo-dev-client';

import React from "react";
import { useColorScheme } from 'react-native';

import { ApolloClient, InMemoryCache, ApolloProvider, gql } from '@apollo/client';
import { DarkTheme, DefaultTheme, ThemeProvider } from '@react-navigation/native';
import { Slot, Stack } from 'expo-router';
import { PaperProvider } from "react-native-paper";
import { AuthSessionProvider } from "../components/AuthenticationContext";
import { ApolloClientProvider } from "../components/ApolloClientProvider";




const RootLayout = () => {
    return <RootLayoutNav />
}

const RootLayoutNav = () => {

    const colorScheme = useColorScheme();

    return (
        <ThemeProvider value={colorScheme === "dark" ? DarkTheme : DefaultTheme}>
            <PaperProvider>
                <ApolloClientProvider>
                    <AuthSessionProvider>
                        <Slot />
                    </AuthSessionProvider>
                </ApolloClientProvider>
            </PaperProvider>
        </ThemeProvider>
    )
}

export default RootLayout;
```

Listing 47: `../../frontend/app/login.tsx`

```
import React from "react";
import { View, Text } from "../components/Themed";
import { Stack, router } from "expo-router";

import { Button, TextInput } from 'react-native-paper';
import { StyleSheet } from "react-native";
import { useSession } from "../components/AuthenticationContext";
import { InvalidEmailPassword } from "../components/InvalidEmailPassword";
import { useApolloClient } from "../components/ApolloClientProvider";

const LoginPage = () => {

    const [email, setEmail] = React.useState("");

    const [password, setPassword] = React.useState("");

    const [showEmailPassError, setShowEmailPassError] = React.useState(false)


    const { login } = useSession();

    const { createClient, setServerUri, serverUri } = useApolloClient();

    return (
        <>
            <Stack.Screen options={{ title: "Login" }} />
            <View style={styles.container}>
                <Text style={styles.title}>Login</Text>
                {showEmailPassError ? <InvalidEmailPassword /> : <></>}
                <View style={styles.itemContainer}>
                    <View style={styles.itemContainer}>
                        <TextInput
                            label="Server URL"
                            value={serverUri}
                            onChangeText={text => setServerUri(text)}
                        />
                    </View>
                    <TextInput
                        label="Email"
                        value={email}
                        onChangeText={text => setEmail(text)}
                    />
                </View>
                <View style={styles.itemContainer}>
                    <TextInput
                        label="Password"
                        value={password}
                        onChangeText={text => setPassword(text)}
                    />
                </View>
                <Button
                    onPress={async () => {
                        // Call a hook in ApolloClientProvider to create the client
                        createClient();

                        const res = await login(email, password)

                        if (!res?.login.success) {
                            // Login was not sucessful
                            setShowEmailPassError(true)
                        } else {
                            // Login was a success, redirect to homepage.
                            router.replace('/')
                        }
                    }}>
                    Login
                </Button>
            </View>
        </>
    )
}

const styles = StyleSheet.create({
    title: {
        fontWeight: "bold",
        fontSize: 32,
        paddingBottom: 32
    },
    container: {
        flex: 1,
        alignItems: "center",
        justifyContent: "center",
        padding: 20
    },
    itemContainer: {
        width: "100%",
        paddingBottom: 10
    }
})

export default LoginPage;
```

Listing 48: `../../frontend/components/ApolloClientProvider.tsx`

```
import React from "react";

import { ApolloClient, InMemoryCache, ApolloProvider, gql, NormalizedCacheObject } from
    '@apollo/client';

const ApolloClientContext = React.createContext<{
    setServerUri: (uri: string) => void,
    createClient: () => void,
    serverUri: string,
    client: ApolloClient<NormalizedCacheObject> | null
}>({
    // Default values
    setServerUri: (uri) => null,
    createClient: () => null,
    serverUri: "",
    client: null
})

// Define a hook to be called within ApolloClientProvider to access the client
export function useApolloClient() {
    const value = React.useContext(ApolloClientContext)

    if (process.env.NODE_ENV !== 'production') {
        if (!value) {
            throw new Error('useApolloClient must be wrapped in a <ApolloClientProvider />');
        }
    }

    return value;
}

export function ApolloClientProvider(props: React.PropsWithChildren) {
    const [serverUriState, setServerUriState] = React.useState("")
    const [clientState, setClientState] = React.useState(new ApolloClient({ cache: new
        InMemoryCache() }))

    return (
        <ApolloClientContext.Provider value={{
            setServerUri: (uri) => {
                setServerUriState(uri)
            },
            createClient: () => {
                // https://www.apollographql.com/docs/react/integrations/react-native/
                setClientState(new ApolloClient({

                    uri: serverUriState,

                    cache: new InMemoryCache()

                }));
            },
            serverUri: serverUriState,
            client: clientState
        }}>
            {clientState ?
                <ApolloProvider client={clientState}>{props.children}</ApolloProvider> :
                // Client not available, just render children as a fallback
                <>
                    {props.children}
                </>
            }
        </ApolloClientContext.Provider>
    )
}
```

Listing 49: ../../frontend/components/AuthenticationContext.tsx

```
import React from "react";

import { useStorageState } from '../util/useStorageState';
import { useMutation } from "@apollo/client";
import { LOGIN, LoginData } from "../gql/mutations/login";
import { useApolloClient } from "./ApolloClientProvider";
import { Text } from "react-native";

const AuthContext = React.createContext<{
    login: (email: string, password: string) => Promise<LoginData | null | undefined>,
    logout: () => void,
    jwt: string | null,
    isLoading: boolean,
}>({
    login: async (email: string, password: string) => null,
    logout: () => null,
    jwt: null,
    isLoading: false
})

// Define a hook to be called within SessionProvider to access the session
export function useSession() {
    const value = React.useContext(AuthContext)

    if (process.env.NODE_ENV !== 'production') {
        if (!value) {
            throw new Error('useSession must be wrapped in a <SessionProvider />');
        }
    }

    return value;
}

export function AuthSessionProvider(props: React.PropsWithChildren) {
    const [[isLoading, jwt], setSession] = useStorageState("session")

    const { client } = useApolloClient();

    if (client) {
        const [login, { data, loading, error }] = useMutation<LoginData>(LOGIN);
        return (
            <AuthContext.Provider value={{
                login: async (email: string, password: string) => {
                    await login({ variables: { email, password } })

                    if (loading) {
                        console.error("Still loading after await fulfilled")
                    }

                    if (error) {
                        console.error(error)
                    }

                    if (data?.login.jwt) {
                        setSession(data.login.jwt)
                        console.log("JWT saved in session")
                    }
                    return data
                },
                logout: () => {
                    console.log("logout called")
                },
                jwt,
                isLoading
            }}>
                {props.children}
            </AuthContext.Provider>
        )
    } else {
        // TODO: make a prettier error message
        console.error("Apollo client not available, was it initialized?")
        return <Text>Apollo client not available</Text>
    }
}
```

Listing 50: `../../frontend/components/Drawer.tsx`

```
import * as React from 'react';

import { Drawer } from 'expo-router/drawer';


// TODO: https://docs.expo.dev/router/advanced/platform-specific-modules/
const DrawerComponent = () => {
    return (
        <Drawer id="root">
            <Drawer.Screen
                name="index" // url
                options={{
                    drawerLabel: "Home",
                    title: "Home"
                }}
            />
            <Drawer.Screen
                name="scan" // url
                options={{
                    drawerLabel: "Quick Scan",
                    title: "Scan"
                }}
            />
        </Drawer>
    );
};

export default DrawerComponent;
```

Listing 51: `../../frontend/components/InvalidEmailPassword.tsx`

```
import React from "react";

import { Text, View } from "./Themed";
import { Surface, useTheme } from "react-native-paper";
import { StyleSheet } from "react-native";

import { MaterialIcons } from '@expo/vector-icons';
```

```
export const InvalidEmailPassword = () => {
    const { colors } = useTheme();
    return (
        <View style={style.outer}>
            <Surface style={{
                backgroundColor: colors.errorContainer,
                ...style.surface
            }}>
                <MaterialIcons name="error-outline" size={24} color="black" style={{ paddingRight:
                     8 }} />
                <Text>Invalid email / password.</Text>

            </Surface>
        </View>
    )
}

const style = StyleSheet.create({
    outer: {
        paddingTop: 16,
        paddingBottom: 16
    },
    surface: {
        padding: 8,
        width: 250,
        alignItems: "center",
        justifyContent: "center",
        flexDirection: "row"
    }
})
```

Listing 52: `../../frontend/components/Themed.tsx`

```
/**
 * Learn more about Light and Dark modes:
 * https://docs.expo.io/guides/color-schemes/
 */

import { Text as DefaultText, useColorScheme, View as DefaultView } from 'react-native';

import Colors from '../constants/Colors';

type ThemeProps = {
  lightColor?: string;
  darkColor?: string;
};

export type TextProps = ThemeProps & DefaultText['props'];
export type ViewProps = ThemeProps & DefaultView['props'];

export function useThemeColor(
  props: { light?: string; dark?: string },
  colorName: keyof typeof Colors.light & keyof typeof Colors.dark
) {
  const theme = useColorScheme() ?? 'light';
  const colorFromProps = props[theme];

  if (colorFromProps) {
    return colorFromProps;
  } else {
    return Colors[theme][colorName];
  }
}

export function Text(props: TextProps) {
  const { style, lightColor, darkColor, ...otherProps } = props;
  const color = useThemeColor({ light: lightColor, dark: darkColor }, 'text');

  return <DefaultText style={[{ color }, style]} {...otherProps} />;
}

export function View(props: ViewProps) {
  const { style, lightColor, darkColor, ...otherProps } = props;
  const backgroundColor = useThemeColor({ light: lightColor, dark: darkColor }, 'background');

  return <DefaultView style={[{ backgroundColor }, style]} {...otherProps} />;
}
```

Listing 53: ../../frontend/constants/Colors.ts

```
const tintColorLight = '#2f95dc';
const tintColorDark = '#fff';

export default {
  light: {
    text: '#000',
    background: '#fff',
    tint: tintColorLight,
    tabIconDefault: '#ccc',
    tabIconSelected: tintColorLight,
  },
  dark: {
    text: '#fff',
    background: '#000',
    tint: tintColorDark,
    tabIconDefault: '#ccc',
    tabIconSelected: tintColorDark,
  },
};
```

Listing 54: ../../frontend/gql/mutations/login.ts

```
import { gql } from "@apollo/client";

interface Login {
    success: boolean,
    jwt: string | null
}

interface LoginData {
    login: Login
}

const LOGIN = gql`
    mutation Login($email: String!, $password: String!) {
        login(email: $email, password: $password) {
            success
            jwt
        }
    }
`
```

```
export {
    LOGIN, LoginData
};
```

Listing 55: ../../frontend/util/useStorageState.ts

```
// https://docs.expo.dev/router/reference/authentication/

import * as SecureStore from 'expo-secure-store';
import * as React from 'react';
import { Platform } from 'react-native';

type UseStateHook<T> = [[boolean, T | null], (value: T | null) => void];

function useAsyncState<T>(
  initialValue: [boolean, T | null] = [true, null],
): UseStateHook<T> {
  return React.useReducer(
    (state: [boolean, T | null], action: T | null = null): [boolean, T | null] => [false, action],
    initialValue
  ) as UseStateHook<T>;
}

export async function setStorageItemAsync(key: string, value: string | null) {
  if (Platform.OS === 'web') {
    try {
      if (value === null) {
        localStorage.removeItem(key);
      } else {
        localStorage.setItem(key, value);
      }
    } catch (e) {
      console.error('Local storage is unavailable:', e);
    }
  } else {
    if (value == null) {
      await SecureStore.deleteItemAsync(key);
    } else {
      await SecureStore.setItemAsync(key, value);
    }
  }
}

export function useStorageState(key: string): UseStateHook<string> {
  // Public
  const [state, setState] = useAsyncState<string>();

  // Get
  React.useEffect(() => {
    if (Platform.OS === 'web') {
      try {
        if (typeof localStorage !== 'undefined') {
          setState(localStorage.getItem(key));
        }
      } catch (e) {
        console.error('Local storage is unavailable:', e);
      }
    } else {
      SecureStore.getItemAsync(key).then(value => {
        setState(value);
      });
    }
  }, [key]);

  // Set
  const setValue = React.useCallback(
    (value: string | null) => {
      setState(value);
      setStorageItemAsync(key, value);
    },
    [key]
  );

  return [state, setValue];
}
```