The new SDL Equipment Booking System

Dr. Roy C. Davies, Senior Technician, the Smart Digital Lab

Background

The Lab Booking System has been designed to make it easy to book the equipment in the Smart Digital Lab and the Trimble Technology Lab using mobile devices. Further, it keeps track of who has been inducted to which equipment, and allows the management of the induction processes to be spread amongst several people. Bookings can be done (and checked) immediately on the phone by scanning a QR code, which links directly to the calendar and information about a piece of equipment.

The database used is PostgreSQL, with the Phoenix Framework in Elixir running a GraphQL API. This combination is both robust and scalable The WebApp is created using Svelte (a modern alternative to React) written in Typescript with various UI libraries.

GitHub

The code is at: https://github.com/uoa-smart-digital-lab/lab-booking-system. Documentation is currently minimal, but this will be improved.

Libraries / languages / frameworks

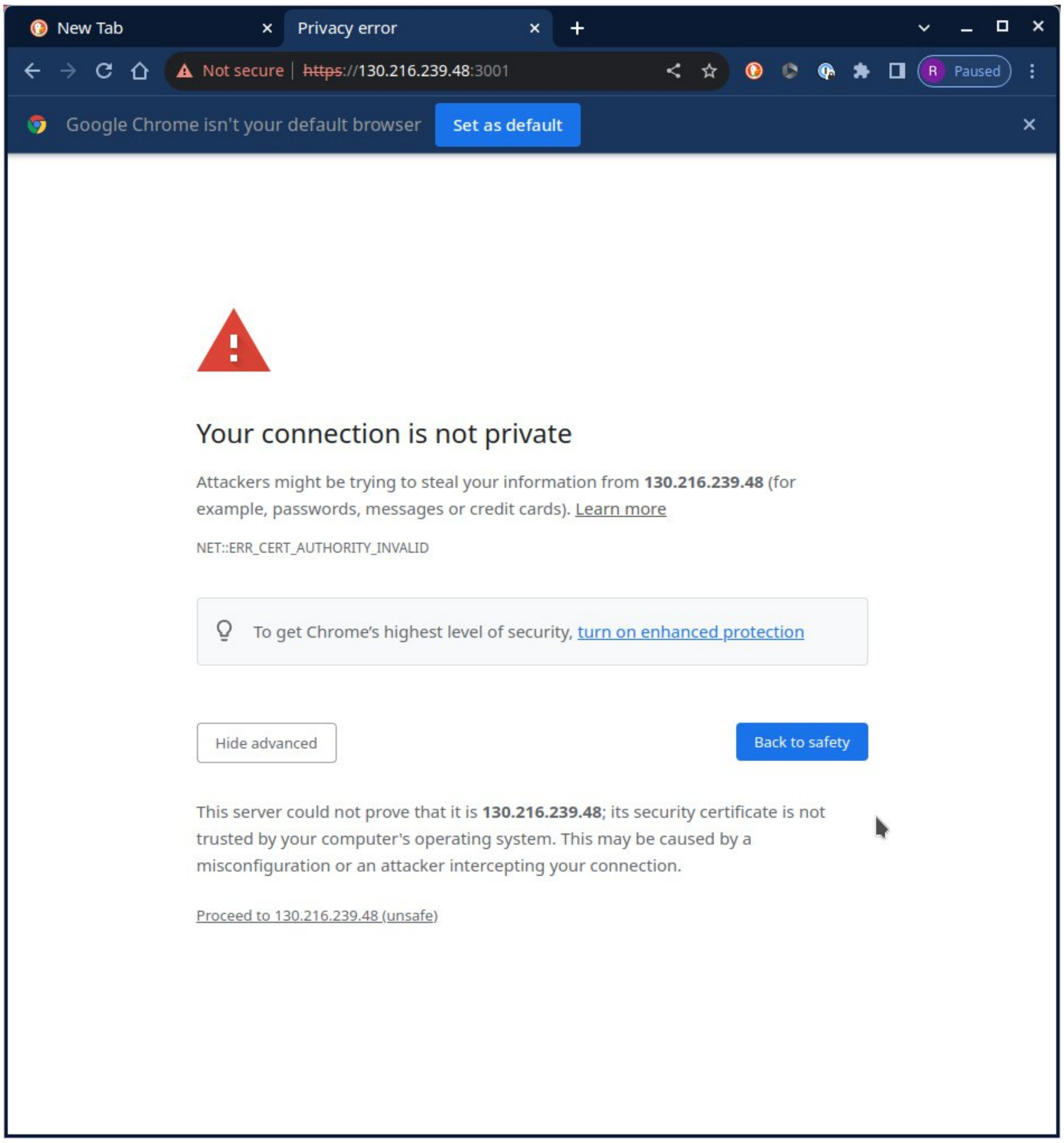
The following list includes the main libraries, languages and frameworks used, though, many of these in turn depend on others. The order is roughly from back end server to front end user experience:

- PostgreSQL: https://wiki.postgresql.org
  - A general purpose database suitable for this sort of information.
- pgAdmin4: https://www.pgadmin.org/
  - An Open Source administration and development platform for PostgreSQL.
- Elixir: https://wiki.postgresql.org
  - A scalable, fault tolerant, functional programing language for robust systems.
- Phoenix Framework: https://www.phoenixframework.org
  - Soft realtime, modern web and API server written in Elixir.
- Absinthe: https://github.com/absinthe-graphql/absinthe
  - Elixir GraphQL programming library.
- GraphQL: https://graphql.org/
  - A dynamic and efficient query language for APIs.
- Postman: https://www.postman.com/
  - An API platform for developers to design, build, test and iterate their APIs.
- Yarn: https://yarnpkg.com/
  - A package manager that doubles down as a project manager.
- Apollo Client: https://github.com/timhall/svelte-apollo
  - Javascript and Typescript programming library for GraphQL.
- Typescript: https://www.typescriptlang.org/
  - A strongly typed programming language that builds on JavaScript.
- Svelte: https://svelte.dev/
  - For building cybernetically enhanced WebApps.
- SvelteUI: https://www.svelteui.org/
  - For creating WebApps in less time than ever before.

- Svelte Calendar: https://github.com/vkurko/calendar
    - Full-sized drag & drop JavaScript event calendar with resource view.

Test Server

The development server is currently running at: https://130.216.239.48:3001/ and can be tested if you are within the University network. You will need to go past the security warnings as this presently is using self-signed certificates for encryption. In Chrome, for example, click on Advanced, and then 'proceed to ...'. If you try the server and it appears to be off, let me know – the computer may have restarted or I may be working on it.


New Tab
Privacy error
Not secure
~~https~~://130.216.239.48:3001
Paused
Google Chrome isn't your default browser
Set as default

Your connection is not private

Attackers might be trying to steal your information from **130.216.239.48** (for example, passwords, messages or credit cards). Learn more

NET::ERR_CERT_AUTHORITY_INVALID

To get Chrome's highest level of security, turn on enhanced protection

This server could not prove that it is **130.216.239.48**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

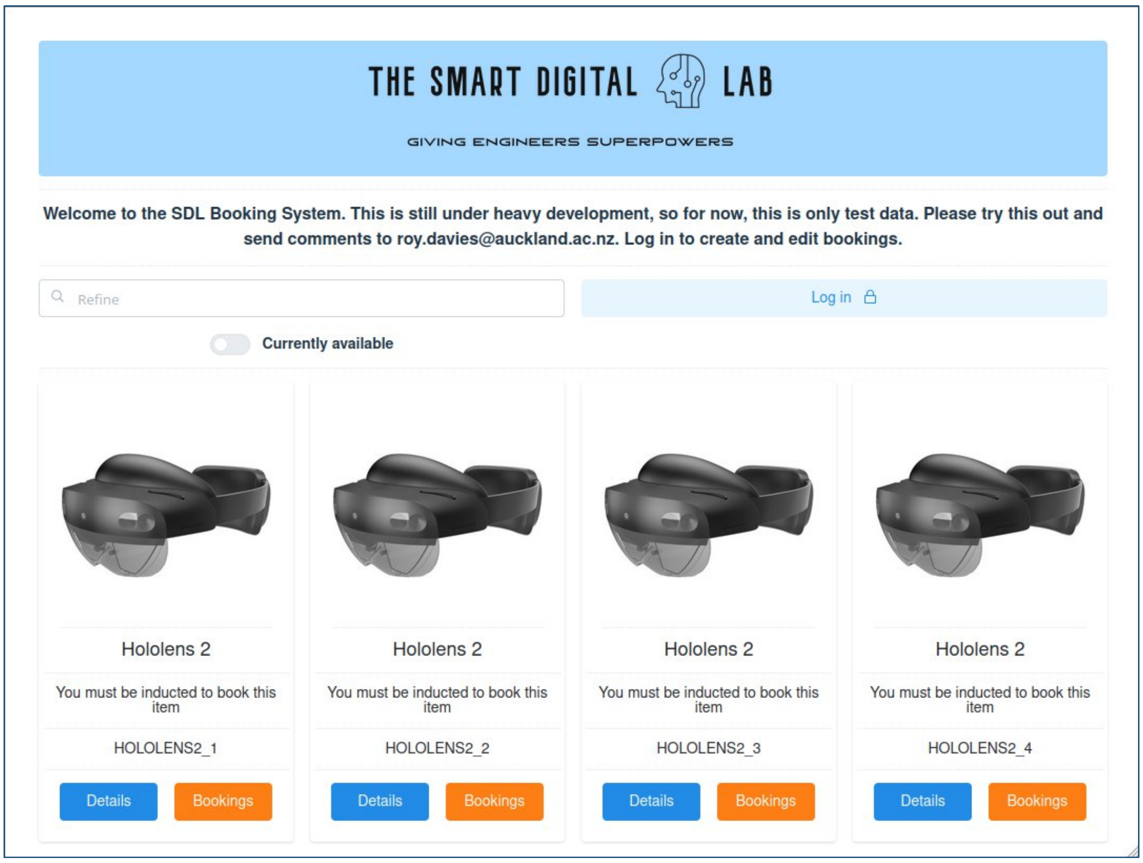
Proceed to 130.216.239.48 (unsafe)

The main WebApp user experience

The main user experience has three views – the list, the calendar and the more details views. Overall, it has been designed to be adaptive / responsive – with a preference towards mobile devices, intuitive and to make the most common tasks as easy and quick as possible.

List of items

The default view is a list of all items. This list can be tweaked by typing into the 'Refine' text area, or selecting to only show items that are currently available (ie not booked). When logged in, you can also select to only show items you are allowed to book (ie those you are inducted for or that require no induction).

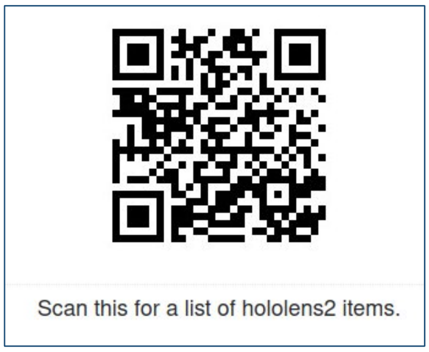


Welcome to the SDL Booking System. This is still under heavy development, so for now, this is only test data. Please try this out and send comments to roy.davies@auckland.ac.nz. Log in to create and edit bookings.

Hololens 2	Hololens 2	Hololens 2	Hololens 2
You must be inducted to book this item	You must be inducted to book this item	You must be inducted to book this item	You must be inducted to book this item
HOLOLENS2_1	HOLOLENS2_2	HOLOLENS2_3	HOLOLENS2_4
Details Bookings	Details Bookings	Details Bookings	Details Bookings

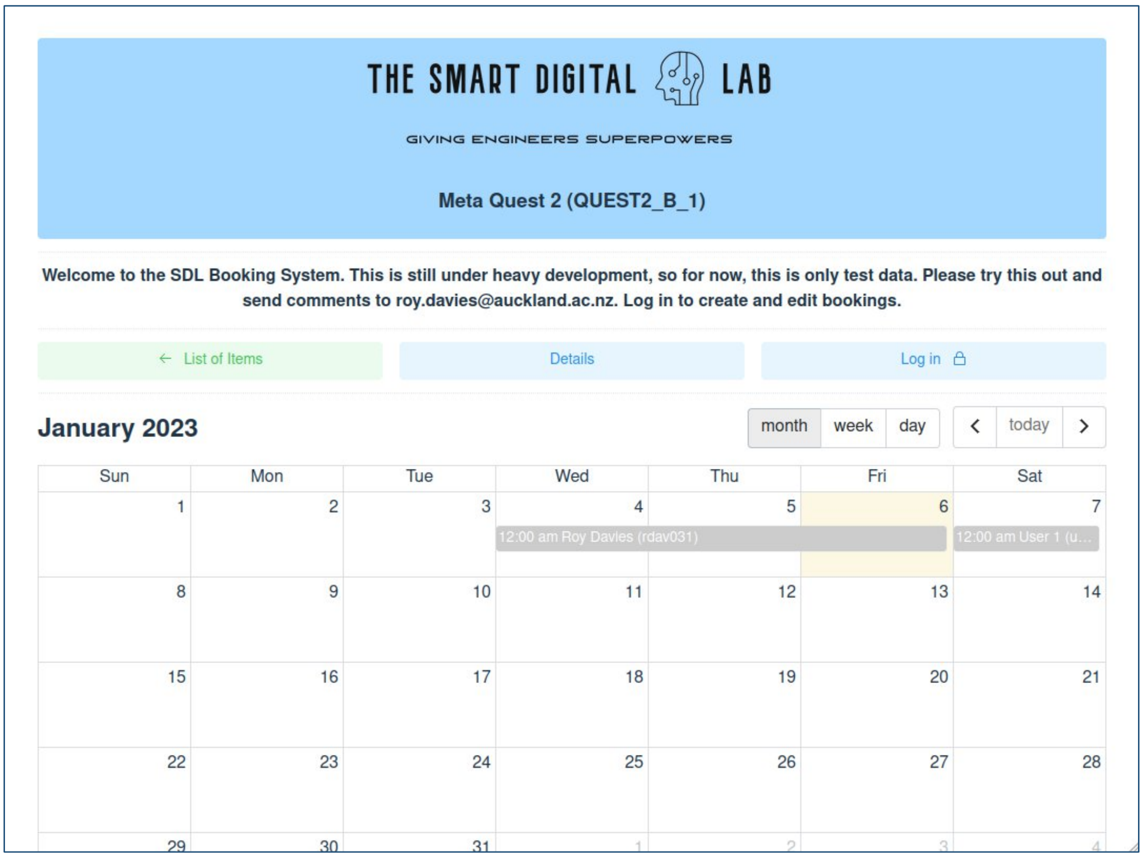
Using QR codes

Each item has a unique QR code that can be printed and placed on the item. That way, someone using (or wanting to use) the equipment can easily find out more about it, get instructions and make and check bookings from their mobile device. Groups of devices can also be accessed from a QR code. Try the codes below (again, presently only if your mobile device is connected to the university network):



The Calendar View

Scanning a QR Code for an item of equipment, or clicking on the ‘Bookings’ button in the list opens the calendar. From here, you can click on the details button to find out more, check the bookings, or go (back) to the main list of equipment items. You can only create and modify bookings after logging in. Students can only modify their own bookings, but staff members can create and modify bookings for students as well as themselves.



THE SMART DIGITAL LAB

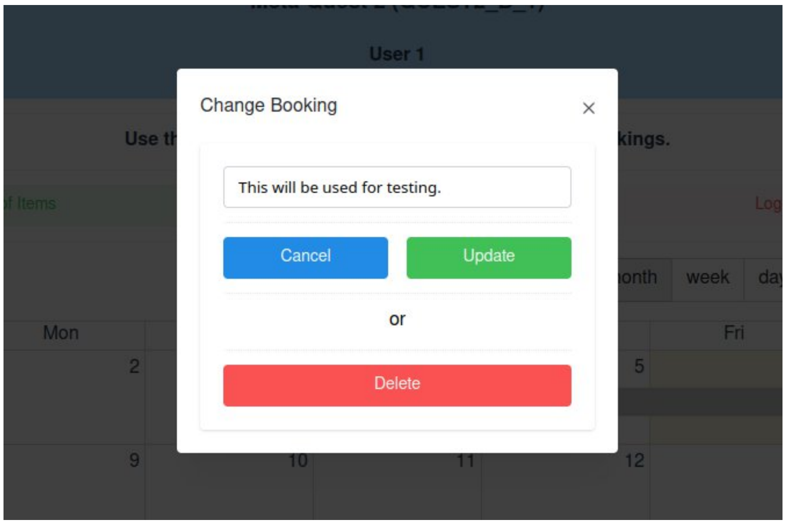
GIVING ENGINEERS SUPERPOWERS

Meta Quest 2 (QUEST2_B_1)

Welcome to the SDL Booking System. This is still under heavy development, so for now, this is only test data. Please try this out and send comments to roy.davies@auckland.ac.nz. Log in to create and edit bookings.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4 12:00 am Roy Davies (rdav031)	5	6	7 12:00 am User 1 (u...
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

After logging in, creating bookings is simply a matter of clicking and dragging out the times or days that you want, and entering a few extra details - but only for equipment you have been inducted for, or that requires no other induction aside from the general lab induction.



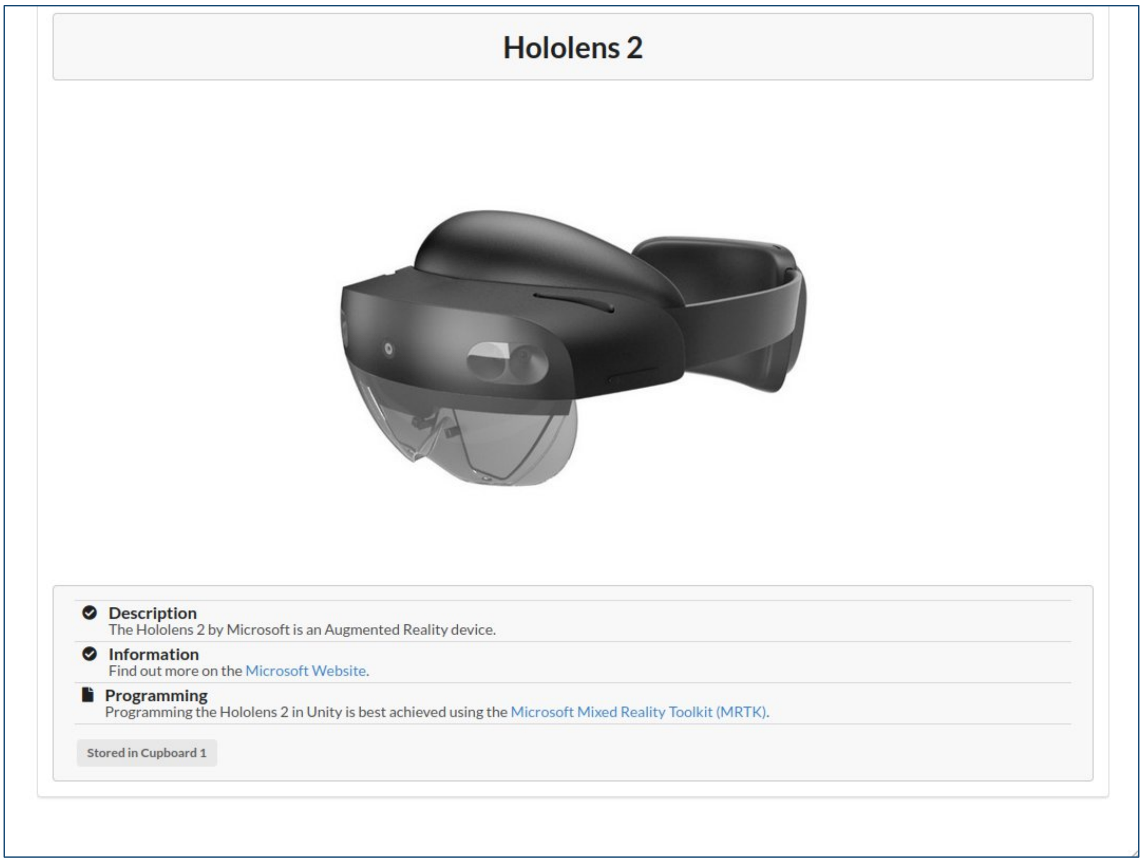
Logging in

You can log into the test server as ‘student1’, ‘student2’ or ‘student3’, as well as ‘lecturer1’ and ‘lecturer2’. For testing purposes, the password for each is presently the building and room number for the SDL (405979).

Ultimately, it would be great to link this to the University SSO. Much of the information is available without logging in, such as checking bookings, and getting information about equipment. This should facilitate the use of the booking system and reduce frustration.

The Details view

Each item of equipment can include a link to another website that can contain any information you like, such as via a canvas page. Ideally, this should link to instruction manuals, programming guides and other useful information.


Hololens 2

Description
The Hololens 2 by Microsoft is an Augmented Reality device.

Information
Find out more on the Microsoft Website.

Programming
Programming the Hololens 2 in Unity is best achieved using the Microsoft Mixed Reality Toolkit (MRTK).

Stored in Cupboard 1

Next steps

There are many possible improvements. Some are planned, such as: keeping accounts and charging for device use; an admin portal; email reminders to pick up and return equipment; and automated processes for risk assessment and off site forms. If you have ideas or suggestions, please do email me at (roy.davies@auckland.ac.nz), or send a message via the Github site.

The GraphQL API

The API is defined as follows and can be found here: https://www.postman.com/smartdigitallab/workspace/sdl-workspace/api/

```
# *****************************************************************************************************************************
# This schema is for a lab bookings system with people and items.  Items can only be booked if a person has the rights to do so as determined by their
# status and induction level for a specific item.  The Administrators have complete power, whereas some capabilities are delegated to powerusers.
# *****************************************************************************************************************************

schema {
  query: Query
  mutation: Mutation
}

# Persons can have different access levels.
enum Status {
  USER,                     # An ordinary person
  POWERUSER,                # A person with some delegated responsibility
  ADMIN                     # Overall administrator who can do anything
}

enum Access {
  FREE,                     # Anybody can book, no induction required
  INDUCTION,                # Must be inducted to book and use
  SUPERVISED                # Requires a supervisor for use (eg Lab Technician)
}

# Used to represent JSON data.  Sent in as stringified JSON, but returned as actual JSON.
scalar Json

# A Date time in ISO 8601 format
scalar DateTime

# *****************************************************************************************************************************
# A person who can use the lab and book items.
# Note that a person who is a poweruser can alter the inductions of persons for items they themselves are inducted for, and alter the bookings for any
# person of items they are inducted for.
# *****************************************************************************************************************************
type Person {
  upi: String!              # The unique University Personal ID
  name: String!             # The person's name
  password: String!         # The password (stored encrypted)
  status: Status!           # The person's level of access
  details: Json!            # Any other relevant details (for future expansion)

  tokens: Int!              # Number of tokens in the account

  inductions: [Item]!       # A list of items this person is inducted for
  bookings: [Booking]!      # A list of current bookings for this person
}
```

```
# ******************************************************************************************************************************
# A persistent session ID.
# ******************************************************************************************************************************
type Session {
  sessionid: String!      # A GUID representing a sessionid
  person: Person!         # The person logged in
}

# ******************************************************************************************************************************
# A bookable item.
# ******************************************************************************************************************************
type Item {
  name: String!           # The unique name of the item
  image: String!          # A link to an image - can be to the images folder, eg /images/hololens2.jpg
  url: String!            # A link to further information about the item (on another webserver, eg canvas)
  details: Json!          # Any other relevant details (for future expansion)

  cost: Int!              # Cost in tokens per time period

  bookable: Boolean!      # Whether bookable at all
  access: Access!         # Level of access use

  inductions: [Person]!   # A list of people for whom this item is inducted.  List will be empty if induction not required (status).
  bookings: [Booking]!    # A list of bookings for this item
}

# ******************************************************************************************************************************
# A bookable item.  Local time is assumed.
# ******************************************************************************************************************************
type Booking {
  person: Person!         # The person the item is booked for
  item: Item!             # The item being booked
  starttime: DateTime!    # The starting time
  endtime: DateTime!      # The ending time
  details: Json!          # Any other relevant details
}

type Query {
  # ******************************************************************************************************************************
  # Person
  # ******************************************************************************************************************************

  # Get the specified person's details.
  # Errors: noperson (Person doesn't exist), notadmin (You are not admin), nosession (No session ID sent in header)
  personGet (upi: String!): Person

  # Get all the people.
  # Errors: notadmin (You are not admin), nosession (No session ID sent in header)
  personAll: [Person]

  # ******************************************************************************************************************************
  # Item
  # ***********************************************************************************************************
  # Get the specified item's details.  Start and end times are used to refine the search for bookings.
  # Errors: noname (No item name given), noitem (Item with given name doesn't exist)
  itemGet (name: String!, starttime: DateTime, endtime: DateTime): Item

  # Get a list of all the items.  Start and end times are used to refine the search for bookings.
  # Errors:
  itemAll (starttime: DateTime, endtime: DateTime): [Item]
}

type Mutation {
  # ***********************************************************************************************************
  # Person
  # ***********************************************************************************************************

  # Create a new person.  Can only be done by Admin users.  Send sessionid by header.
  # Errors: personexists (Person already exists), notadmin (You are not admin), nosession (No session ID sent in header), internalerror (Internal error)
  personCreate (upi: String!, name: String!, password: String!, status: Status!, details: Json!): Person

  # Update a person.  Can only be done by Admin users.  Send sessionid by header.
  # Errors: noperson (Person doesn't exist), notadmin (You are not admin), nosession (No session ID sent in header), internalerror (Internal error)
  personUpdate (upi: String!, name: String, password: String, status: Status, details: Json): Person

  # Delete a person.  Can only be done by Admin users.  Send sessionid by header.
  # Errors: noperson (Person doesn't exist), notadmin (You are not admin), nosession (No session ID sent in header), internalerror (Internal error)
  personDelete (upi: String!): Person

  # Add or remove an item to or from a person's list of inducted items.  Can only be done by admins, and powerusers who are themselves inducted to the
  # item they are inducting someone else to.  So, powerusers can only be inducted by admins (for any item) or other powerusers who are inducted to the item.
  personInduct (upi: String!, itemName: String!): Person
  personUninduct (upi: String!, itemName: String!): Person

  # Change the number of tokens for a specific person.  Can only be done by administrator.
  personAdjustTokens (upi: String!, tokens: Int!): Person

  # ***********************************************************************************************************
  # Session
  # ***********************************************************************************************************

  # Log in.  Set sessionid returned as a header for subsequent API calls.
  # Errors: badpassword (Password incorrect), nosession (upi not given), noparam (no upi or password given)
  login (upi: String!, password: String!): Session

  # Log out.  Uses sessionid in the header to know which person to log out.
  # Errors: nosession (No session ID sent in header), internalerror (Internal error)
  logout: Session

  # ***********************************************************************************************************
  # Item
  # ***********************************************************************************************************

  # Create a new item.  Can only be done by admins and powerusers.
  # Errors: nosession (No SessionID sent), notadmin (You are not admin or poweruser), itemexists (Item already exists), internalerror (Internal error)
  itemCreate (name: String!, image: String!, url: String!, details: Json!): Item

  # Update an item.  Can only be done by admins.
  # Errors: nosession (No SessionID sent), notadmin (You are not admin or poweruser), noitem (Item doesn't exists), internalerror (Internal error)
  itemUpdate (name: String!, image: String, url: String, details: Json): Item

  # Delete an item.  Can only be done by admins.
  # Errors: nosession (No SessionID sent), notadmin (You are not admin or poweruser), noitem (Item doesn't exists), internalerror (Internal error)
  itemDelete (name: String!): Item

  # ******************************************************************************************************************************************************
  # Bookings
  # ******************************************************************************************************************************************************
  # Make a booking for an item.  Anyone can book an item they are inducted for.  Users can only book for themselves.  Powerusers can book for themselves
  # and for others for items they are inducted for, and that the user is inducted for. Items cannot be booked for a specific time of day if already booked.

  # Create a booking for the given item for the given person as identified by their UPI
  itemBook (upi: String!, itemname: String!, starttime: DateTime!, endtime: DateTime!, details: Json!): Booking

  # Unbook the item at this time
  itemUnbook (itemname: String!, starttime: DateTime!, endtime: DateTime!): Booking

  # Delete an existing booking
 itemChangebooking (upi: String!, itemname: String!, starttime: DateTime!, endtime: DateTime!, newstarttime: DateTime, newendtime: DateTime, details: Json):
Booking
}
```