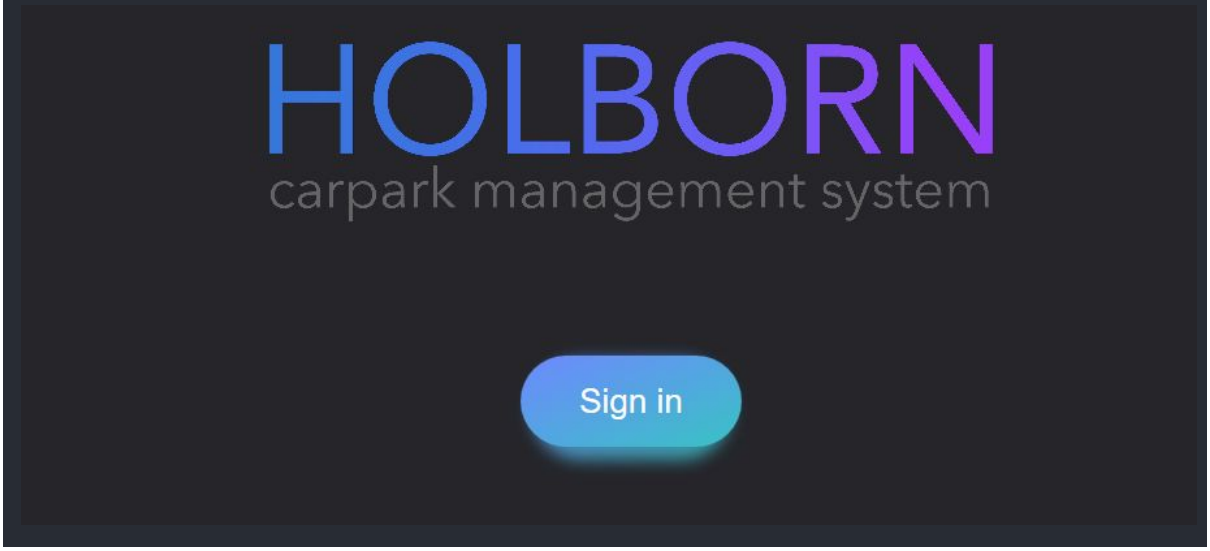
CS1813 Car Park System

Vlad, Cameron, Henry, Nanyo

```
git clone https://github.com/RHUL-CS-Projects/CS1813_2019_05
```

{



// .. Introduction

The system represented in this document is for the Holborn car park management software. It has the following features:

- Barriers that distribute and read tickets for individuals wanting to enter and leave.
- Ticket prices; which depend on the kind of customer wanting to use the car park.
- A web based management console, which allows the manager of the car park to access/update information and make reports regarding the car park and how much it is used.

The purpose of this document is to outline the work done and explain how the project works.

// .. Scope and Function based upon requirements

Our app contains most of the functions in the requirements, and is very deployable.
You can view the admin console by visiting our domain: https:holborncarpark.com/

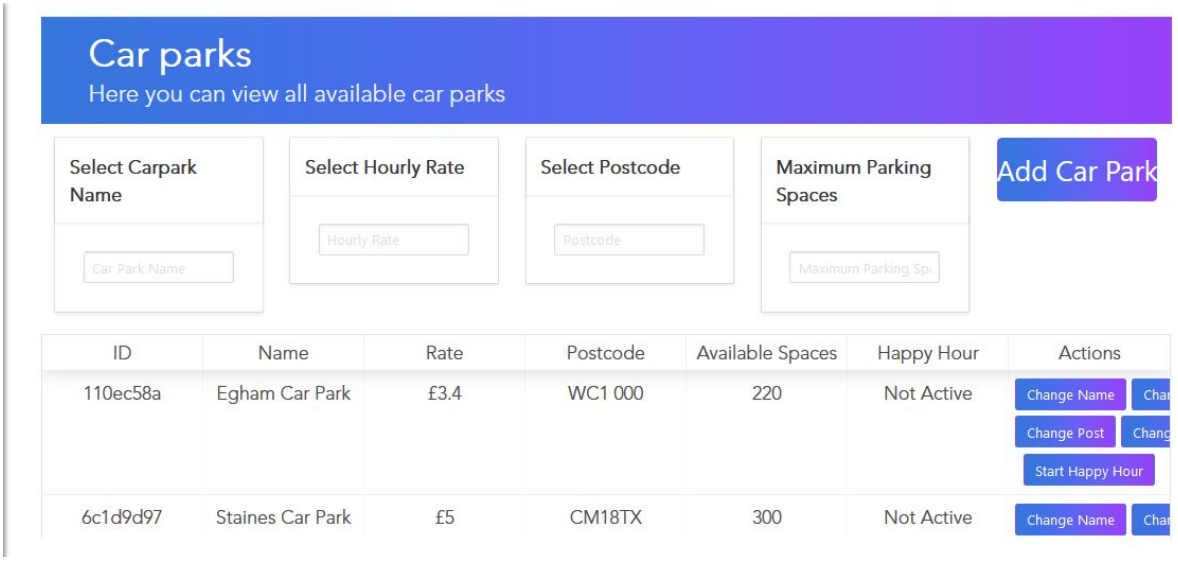
From the requirements documentation, we identified the following tasks:

Manger console

Happy Hour

Status: Completed
Priority: low

This works as defined in the specification, happy hours can be specified and created in the admin console, there are two ways to do this, one is from the carpark tab:



Report Generation

Status: Completed
Priority: High

This works as expected, the user can specify times they want to generate the report between adn hit generate, which will display a graph and some data.

Car Park Report

Requested by: boi

Date: 4 March 2019

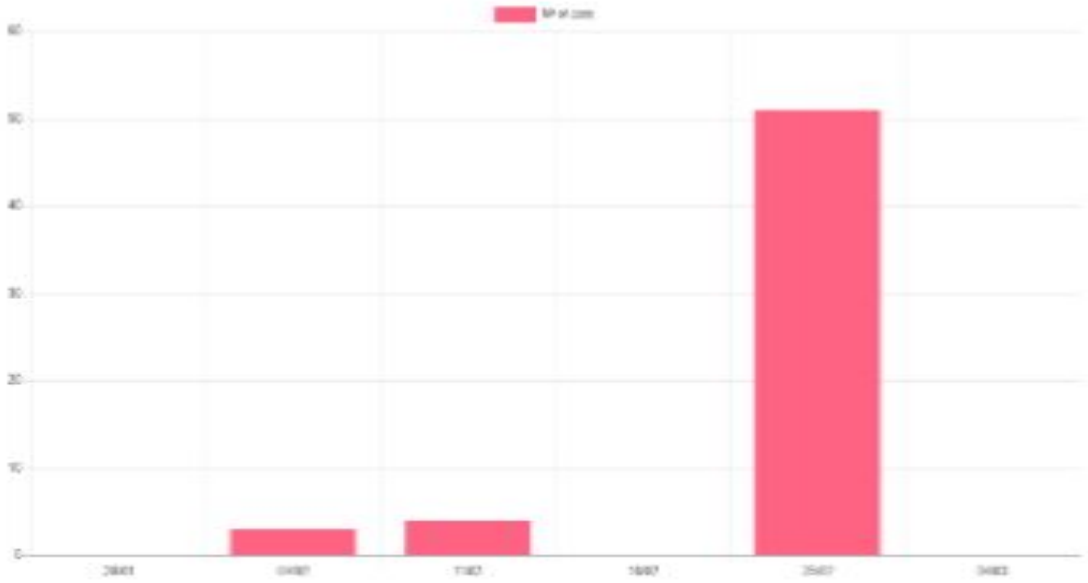
For time period: 28 January 2019 - 4 March 2019

For car park: Egham Car Park

Overview

Total cars: 58

Total revenue: £2491.20



Reports

Here you can generate reports

Select Carpark

Staines Car Park

Select Time

From: 27/01/2019 12:00 AM - To: 04/03/2019 12:08 AM

Generate

Print

No ticket data for selected period

Select Carpark

Egham Car Park

Select Time

From: 24/02/2019 12:00 AM - To: 04/03/2019 11:54 AM

Generate

Print

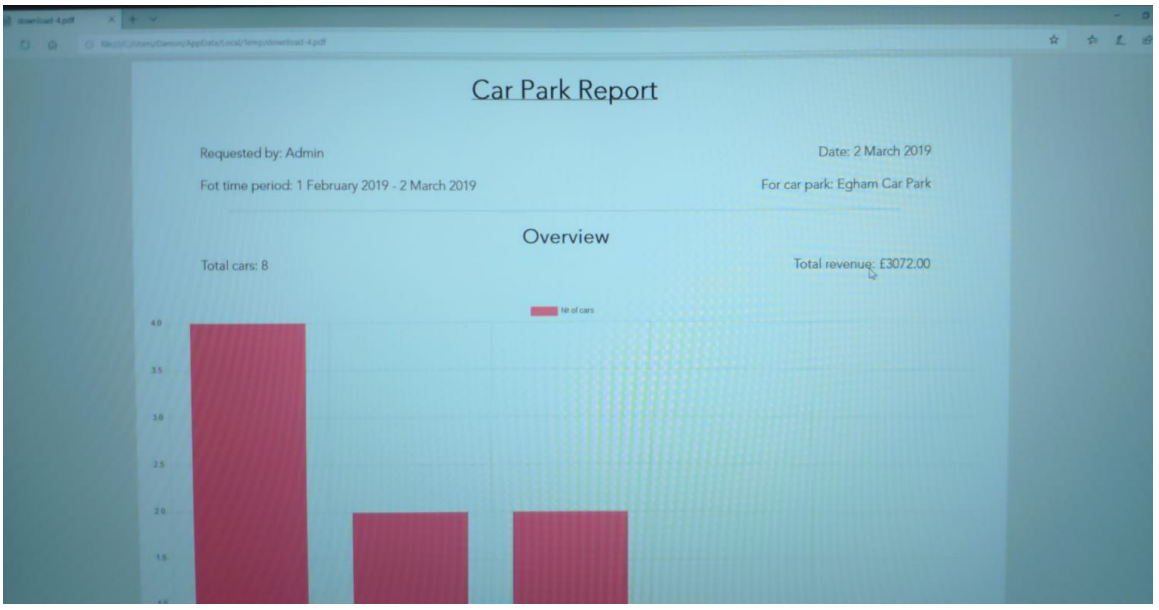
Fot time period: 24 February 2019 - 4 March 2019

For car park: Egham Car Park

Overview

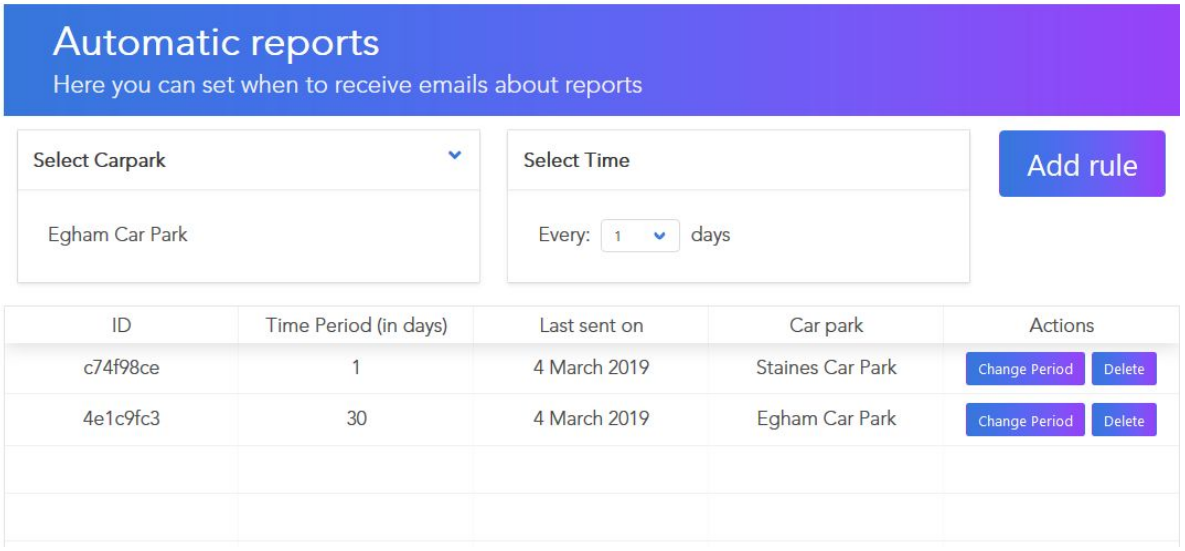
Total cars: 51

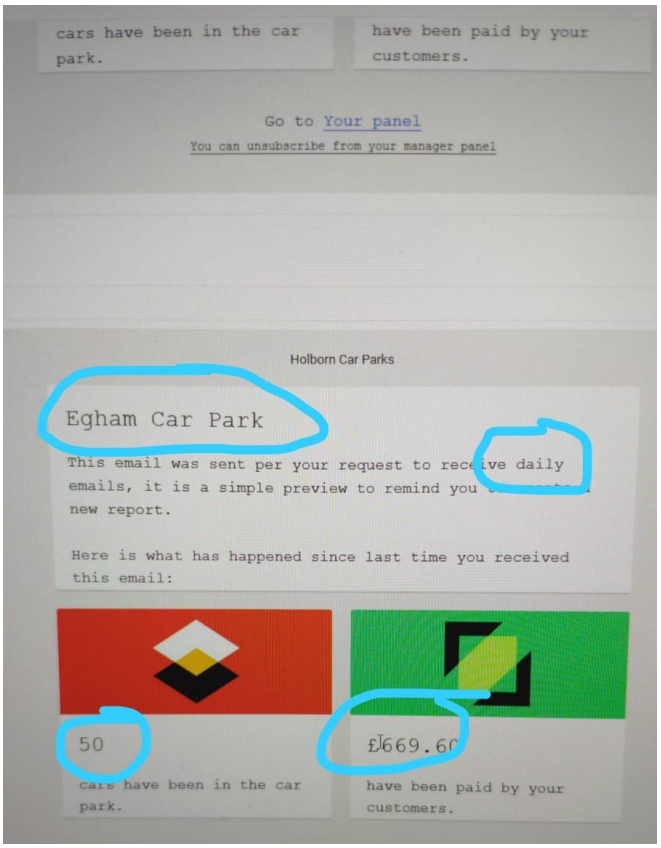
Total revenue: £711.60



Automated Reports
Status: Completed
Priority: Medium

This also works as expected, automatic emails with dynamic data are sent to the given email address specified when an account is created.



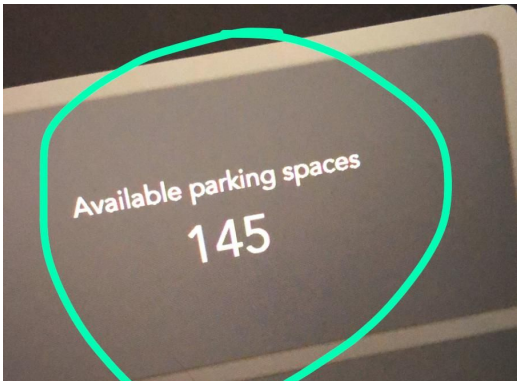


Carpark statistics

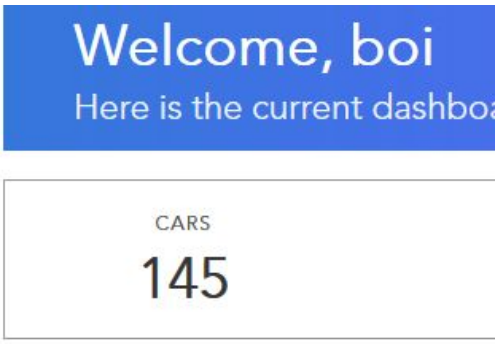
Status: Completed
Priority: Medium

The documentation stated that it should be possible to see how many cars are in the carpark at any given time. This is indeed possible in several ways:

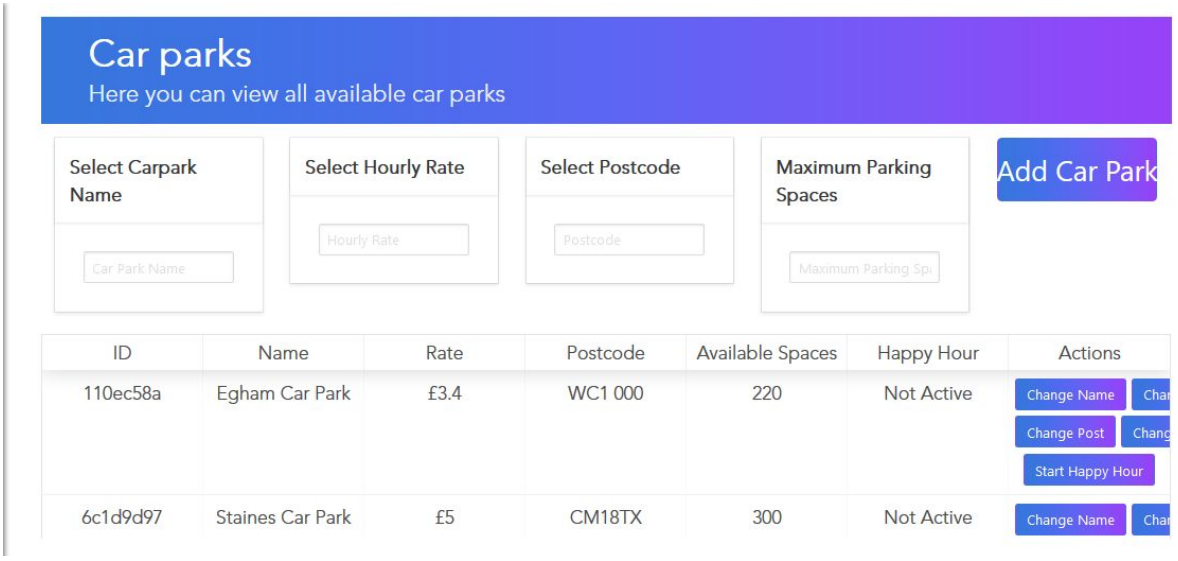
1. From the ticket machine



2. From the web console (this feature may or may not be working at the time of publishing)



3. In the Carparks tab



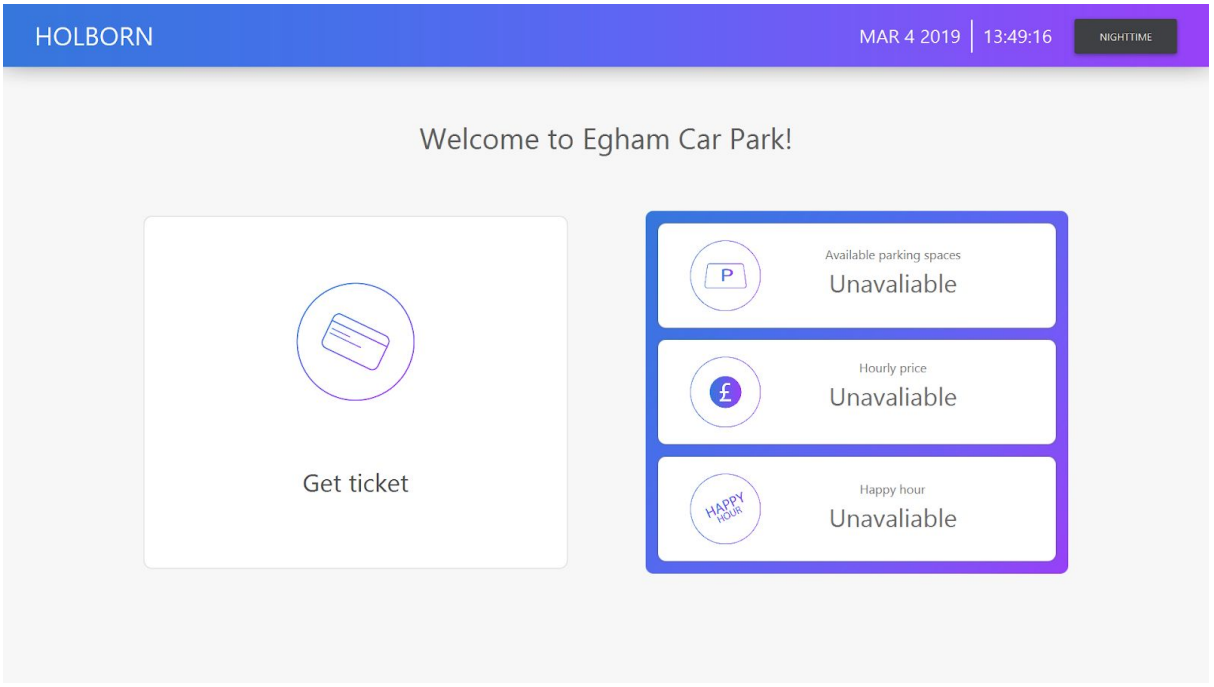
Ticket machine

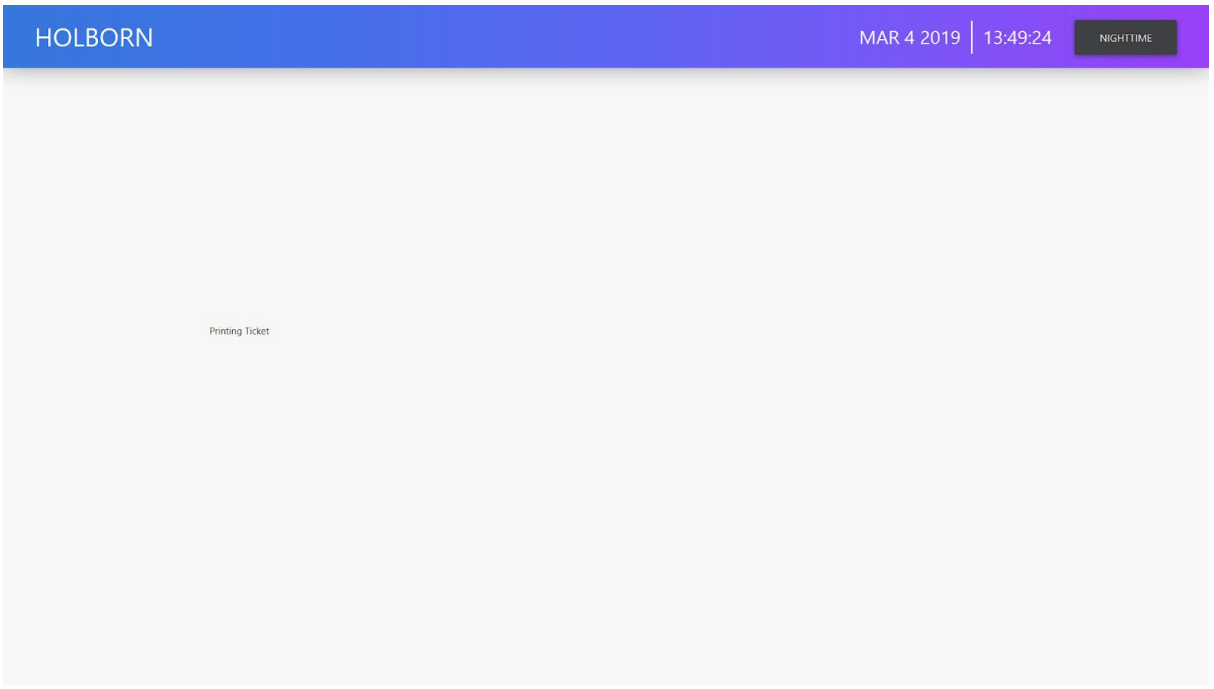
Purchase tickets

Status: Completed
Priority: High

This works, you can generate a ticket with the barriers app:

It is located in the project file, in the barriers project /Tickets/Ticket. It has a QR code the ID so you can paste it.

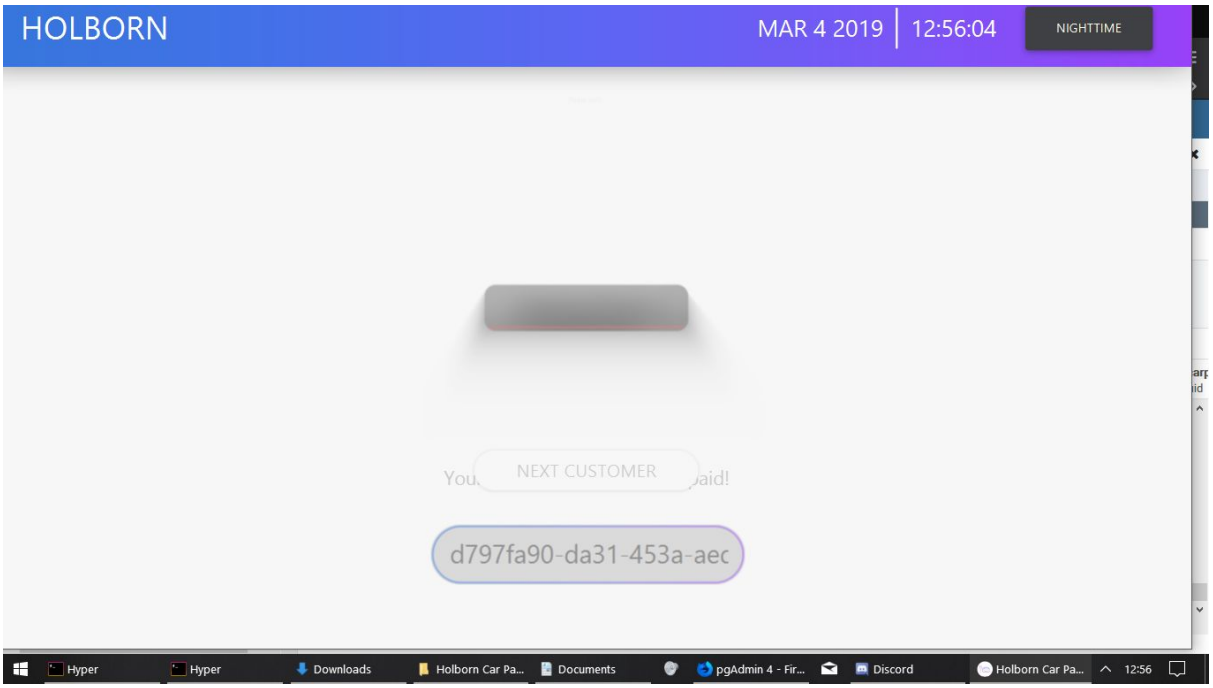


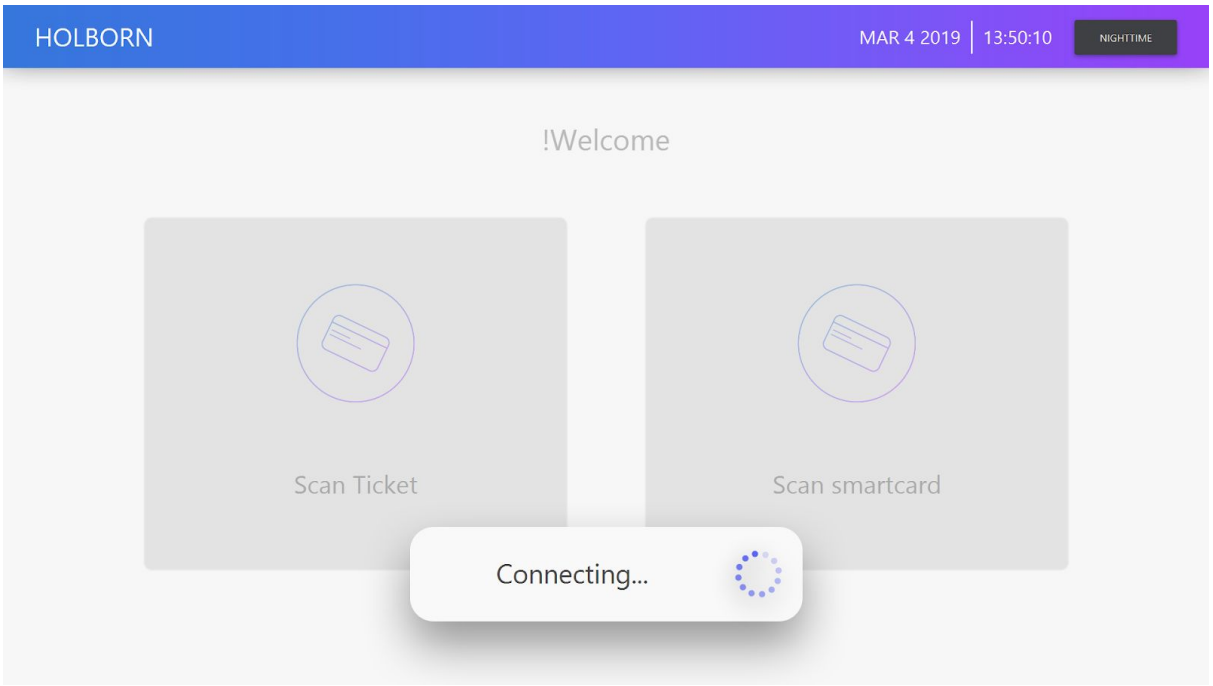


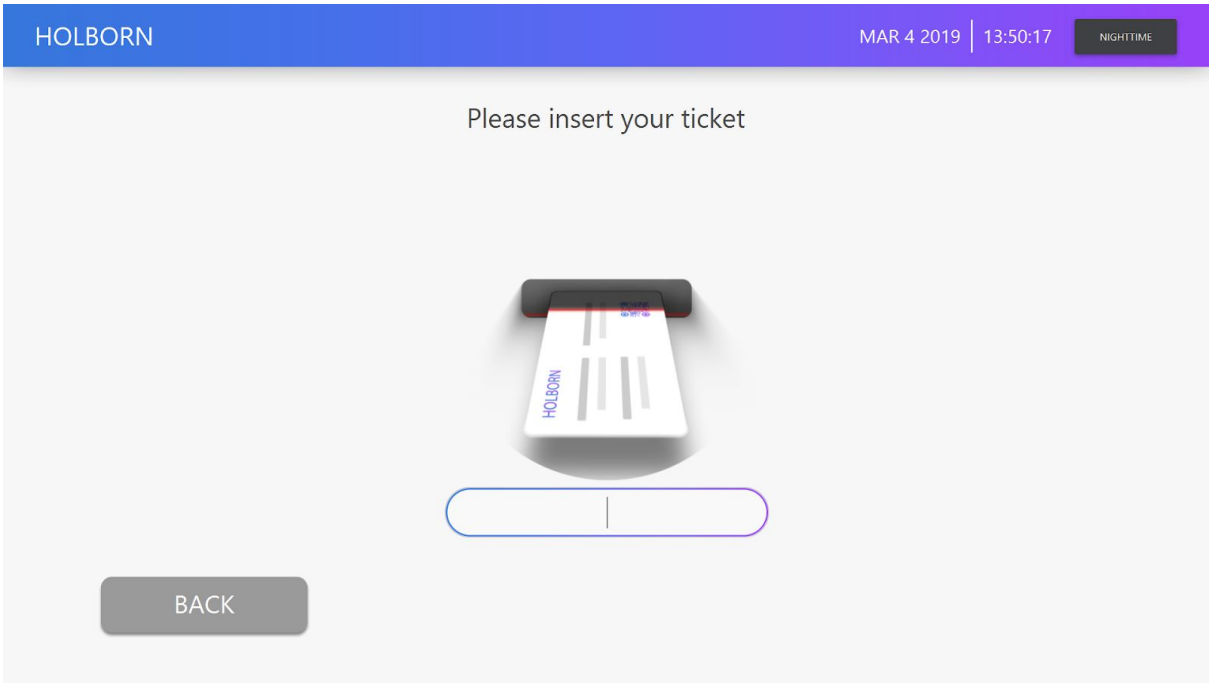
Use tickets

Status: Completed
Priority: High

Works as expected! The user can input their ticket ID and it will be accepted


HOLBORN
MAR 4 2019 | 12:56:04
NIGHTTIME
NEXT CUSTOMER
d797fa90-da31-453a-aec
Hyper
Hyper
Downloads
Holborn Car Pa...
Documents
pgAdmin 4 - Fir...
Discord
Holborn Car Pa...
12:56

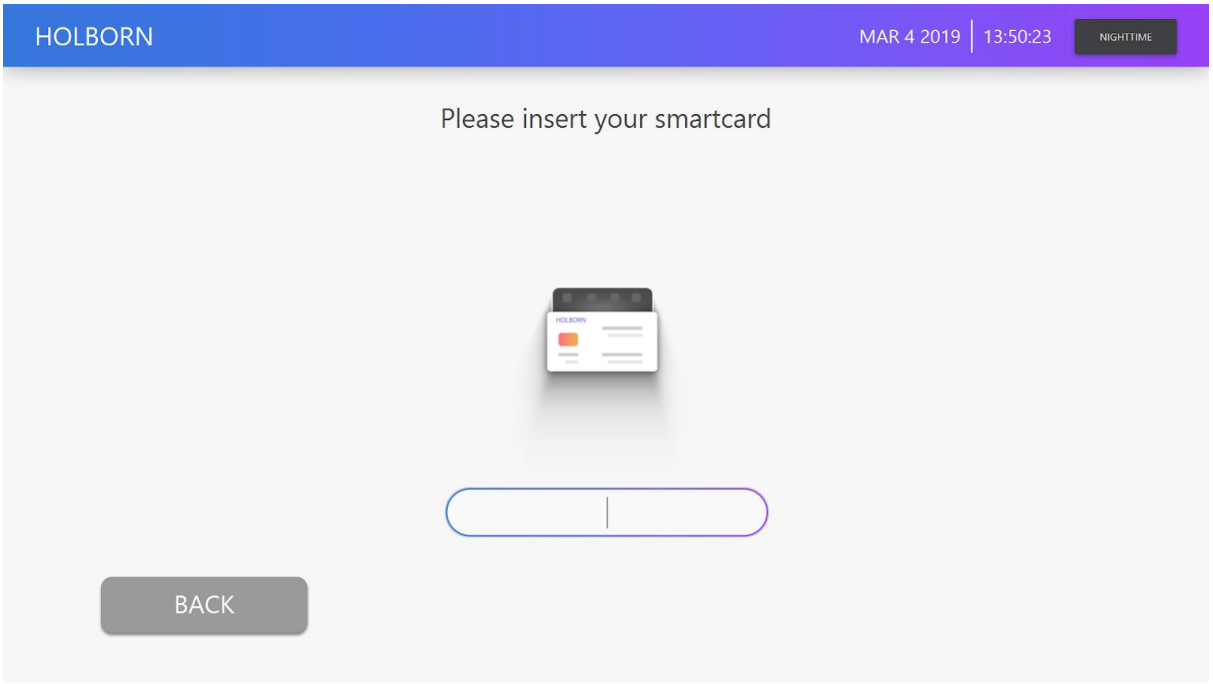

HOLBORN
MAR 4 2019 | 13:50:10
NIGHTTIME
!Welcome
Scan Ticket
Scan smartcard
Connecting...



Resident discount

Status: Completed
Priority: High

The discount works - it gives you a timestamp for the whole day

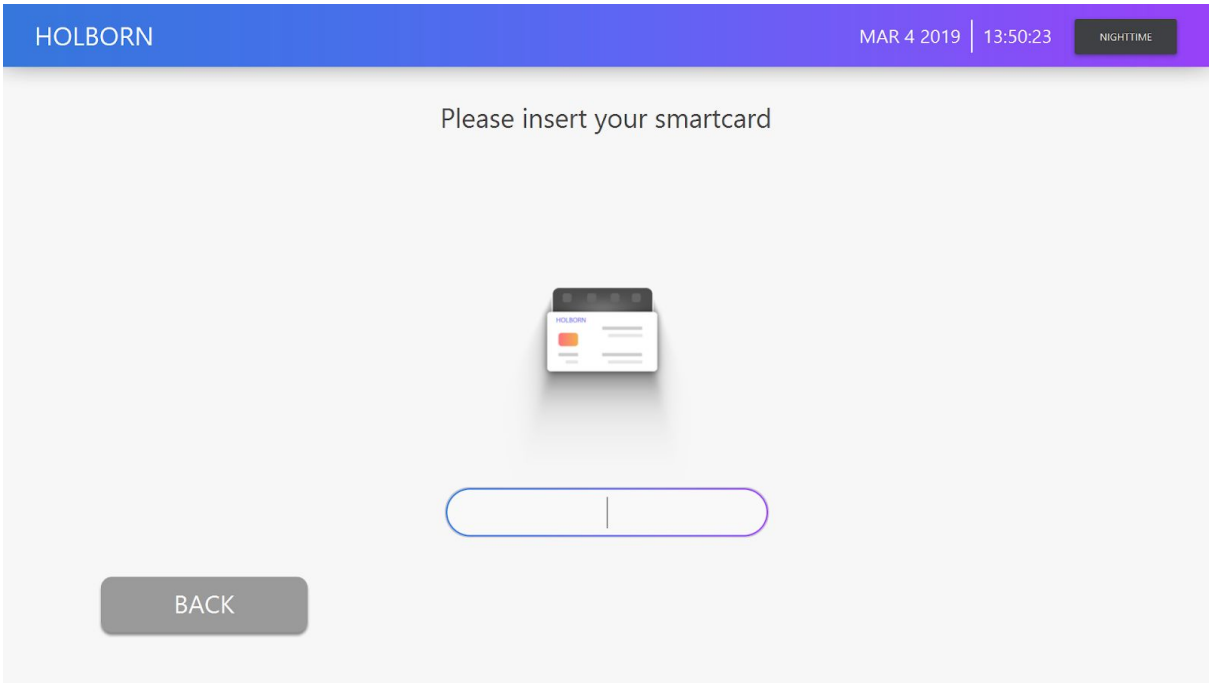


Employee Discount

Status: Completed

Priority: High

The discount works - it gives you a timestamp for the whole day



// .. Code organisation and Structure

We chose to organise our code by splitting it into 3 major parts:

- Service app - This is all the server side code such as the api and express server, it also contains the manager console code which allows the manager to manage the data in the carpark.

- Ticket machine client - This java program checks the ticket and allows the user to pay. It communicates to the server dynamically.

- Barriers - The barriers communicate with the ticket machine client to validate users and let them out out an in.

For code quality we commented our files and the java codebase can be analysed in more detail by running javadoc in the correct directory.

Below you can see the structure of our project:

.

```
├──── README.md
├──── holborn-car-park-barriers
│     ├──── config.xml
│     ├──── logs
│     │     ├──── debug.log
```

```
│   │   ├── error.log
│   │   ├── framework.log
│   │   ├── info.log
│   │   ├── perf.log
│   │   └── trace.log
│   ├── pom.xml
│   └── src
│       └── main
│           ├── java
│           │   ├── FxStuff
│           │   │   ├── Alerter.java
│           │   │   ├── Animator.java
│           │   │   ├── Controllers
│           │   │   │   ├── CheckController.java
│           │   │   │   ├── FinishController.java
│           │   │   │   ├── InfoPopUpController.java
│           │   │   │   ├── LandingInPageController.java
│           │   │   │   ├── LandingOutPageController.java
│           │   │   │   ├── MainViewController.java
│           │   │   │   └── TicketPrintingController.java
│           │   │   ├── GlobalVariables.java
│           │   │   ├── InfoPopUp.java
│           │   │   ├── MainWindow.java
│           │   │   ├── SceneManager.java
│           │   │   ├── Scenes.java
│           │   │   ├── ScreenLauncher.java
│           │   │   ├── Sprites
│           │   │   │   ├── Sprite.java
│           │   │   │   ├── SpriteSettings.java
│           │   │   │   ├── SpriteSheets.java
│           │   │   │   └── Sprites.java
│           │   │   ├── ThemeProvider.java
│           │   │   ├── Themes.java
│           │   │   └── Ticket.java
│           │   ├── Networking
│           │   │   ├── Barrier.java
│           │   │   ├── MultiServerThread.java
│           │   │   ├── NoConnectionError.java
│           │   │   ├── Protocol.java
│           │   │   └── Server.java
│           │   └── QRCode
│           │       ├── Inputs
│           │       │   ├── AlignmentPatternLocations.txt
│           │       │   ├── AlphaToInt.txt
│           │       │   ├── CharLevels.txt
│           │       │   ├── ErrorCorrection.txt
│           │       │   ├── FormatInfo.txt
│           │       │   ├── IntToAlpha.txt
│           │       │   ├── PolynomialTable.txt
│           │       │   ├── RemainderBits.txt
│           │       │   └── VersionInfo.txt
│           │       ├── Pix.java
│           │       ├── QRCode.java
│           │       ├── QRScreenLauncher.java
│           │       └── Screen.java
│           └── resources
│               ├── css
│               │   ├── application.css
│               │   ├── dark.css
│               │   └── light.css
│               ├── fxml
│               │   ├── check.fxml
│               │   ├── finish.fxml
│               │   ├── info_popup.fxml
│               │   ├── landing_in_page.fxml
```

```
│               │   ├── landing_out_page.fxml
│               │   ├── main_view.fxml
│               │   └── printing_ticket.fxml
│               ├── img
│               │   ├── Sprites
│               │   │   ├── smartcard_check_sprite.png
│               │   │   ├── thank_you_sprite.png
│               │   │   └── ticket_insert_sprite.png
│               │   ├── available_parking_spaces_1.png
│               │   ├── checkmark.png
│               │   ├── get_ticket.png
│               │   ├── happy_hour.png
│               │   ├── hourly_price.png
│               │   ├── validate_ticket.png
│               │   └── x_mark.png
│               └── log4j2.xml
├── holborn-car-park-client
│   ├── config.xml
│   ├── dependency-reduced-pom.xml
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   ├── META-INF
│       │   │   │   └── MANIFEST.MF
│       │   │   └── uk
│       │   │       └── co
│       │   │       └── holborn
│       │   │           └── carparkclient
│       │   │               ├── Alerter.java
│       │   │               ├── Animator.java
│       │   │               ├── GlobalVariables.java
│       │   │               ├── InfoPopUp.java
│       │   │               ├── MainAppLauncher.java
│       │   │               ├── MainWindow.java
│       │   │               ├── Networking
│       │   │               │   ├── MultiServerThread.java
│       │   │               │   ├── NoConnectionError.java
│       │   │               │   ├── Protocol.java
│       │   │               │   └── Server.java
│       │   │               ├── SceneManager.java
│       │   │               ├── Scenes.java
│       │   │               ├── Sprite.java
│       │   │               ├── SpriteSettings.java
│       │   │               ├── SpriteSheets.java
│       │   │               ├── Sprites.java
│       │   │               ├── ThemeProvider.java
│       │   │               ├── Themes.java
│       │   │               ├── Ticket.java
│       │   │               ├── TicketDetailsPopUp.java
│       │   │               └── controllers
│       │   │                   ├── CheckController.java
│       │   │                   ├── FinishController.java
│       │   │                   ├── InfoPopUpController.java
│       │   │                   ├── LandingPageController.java
│       │   │                   ├── MainViewController.java
│       │   │                   ├── PaymentMethodsCashController.java
│       │   │                   ├── PaymentMethodsClassicCardController.java
│       │   │                   ├── PaymentMethodsContactlessController.java
│       │   │                   ├── PaymentMethodsController.java
│       │   │                   └── TicketDetailsPopUpController.java
│       │   └── resources
│       │       ├── client_icon.png
│       │       ├── css
│       │       │   ├── dark.css
```

```
│       │       │   └── light.css
│       │       ├── img
│       │       │   ├── Apple_Pay_Mark_RGB_SMALL.png
│       │       │   ├── available_parking_spaces_1.png
│       │       │   ├── card_icon.png
│       │       │   ├── cash_icon.png
│       │       │   ├── checkmark.png
│       │       │   ├── contactless payment.png
│       │       │   ├── happy_hour.png
│       │       │   ├── hourly_price.png
│       │       │   ├── lost_ticket.png
│       │       │   ├── smartcard_check.png
│       │       │   ├── sprites
│       │       │   │   ├── apple_pay_contactless_sprite.png
│       │       │   │   ├── card_pay_anim_sprite.png
│       │       │   │   ├── classic_card_pay_sprite.png
│       │       │   │   ├── coins_in_sprite.png
│       │       │   │   ├── smartcard_check_sprite.png
│       │       │   │   ├── thank_you_sprite.png
│       │       │   │   └── ticket_insert_sprite.png
│       │       │   ├── validate_ticket.png
│       │       │   └── x mark.png
│       │       ├── log4j2.xml
│       │       └── views
│       │           ├── check.fxml
│       │           ├── finish.fxml
│       │           ├── happy_hour_view.fxml
│       │           ├── info_popup.fxml
│       │           ├── landing_page.fxml
│       │           ├── main_view.fxml
│       │           ├── payment_methods.fxml
│       │           ├── payment_methods_cash.fxml
│       │           ├── payment_methods_classic_card.fxml
│       │           ├── payment_methods_contactless.fxml
│       │           └── ticket_details_popup.fxml
│       └── test
│           └── java
│               └── uk
│                   └── co
│                       └── holborn
│                           └── carparkclient
│                               └── Tests.java
└── holborn-car-park-service-app
    ├── app.js
    ├── bin
    │   └── www
    ├── package-lock.json
    ├── package.json
    ├── public
    │   ├── js
    │   │   ├── login.js
    │   │   ├── preferredThemeScheme.js
    │   │   └── register.js
    │   ├── protected
    │   │   ├── HTML
    │   │   │   ├── 404.html
    │   │   │   ├── index.html
    │   │   │   ├── login.html
    │   │   │   └── register.html
    │   │   └── resources
    │   │       └── emailTemplate.html
    │   ├── resources
    │   │   ├── fonts
    │   │   │   └── AvenirLTStd-Light.woff
    │   │   └── img
```

```
│   │   │   ├── logo.png
│   │   │   └── logo_bg_white@4x.png
│   │   └── stylesheets
│   │       ├── global.css
│   │       ├── gradient-shadow.css
│   │       ├── landing.css
│   │       ├── login.css
│   │       └── manager.css
├── server
│   ├── SSL
│   │   ├── ca.pem
│   │   ├── cert.pem
│   │   └── priv.pem
│   ├── databases
│   │   ├── auth_db_conn.js
│   │   ├── carpark_db_conn.js
│   │   └── queries.js
│   ├── javascripts
│   │   ├── global.js
│   │   ├── json_response.js
│   │   ├── mailer.js
│   │   ├── utils.js
│   │   ├── validate.js
│   │   └── verify.js
│   ├── react
│   │   ├── Settings.json
│   │   ├── js
│   │   │   └── components
│   │   │   ├── Carparks.jsx
│   │   │   ├── Clock.jsx
│   │   │   ├── Dash.jsx
│   │   │   ├── Emails.jsx
│   │   │   ├── Employees.jsx
│   │   │   ├── HappyHour.jsx
│   │   │   ├── Missing404.jsx
│   │   │   ├── NavBar.jsx
│   │   │   ├── Notify.jsx
│   │   │   ├── Pricing.jsx
│   │   │   ├── Report.jsx
│   │   │   ├── ReportSection.jsx
│   │   │   ├── Settings.jsx
│   │   │   ├── SideBar.jsx
│   │   │   └── Tickets.jsx
│   │   ├── manager.html
│   │   └── manager.jsx
│   ├── routes
│   │   ├── api
│   │   │   ├── autoreports.js
│   │   │   ├── carparks.js
│   │   │   ├── smartcards.js
│   │   │   ├── tickets.js
│   │   │   └── users.js
│   │   ├── main.js
│   │   └── utility.js
│   └── sockets
│       ├── socket.js
│       └── socket_functions.js
└── webpack.config.js
```

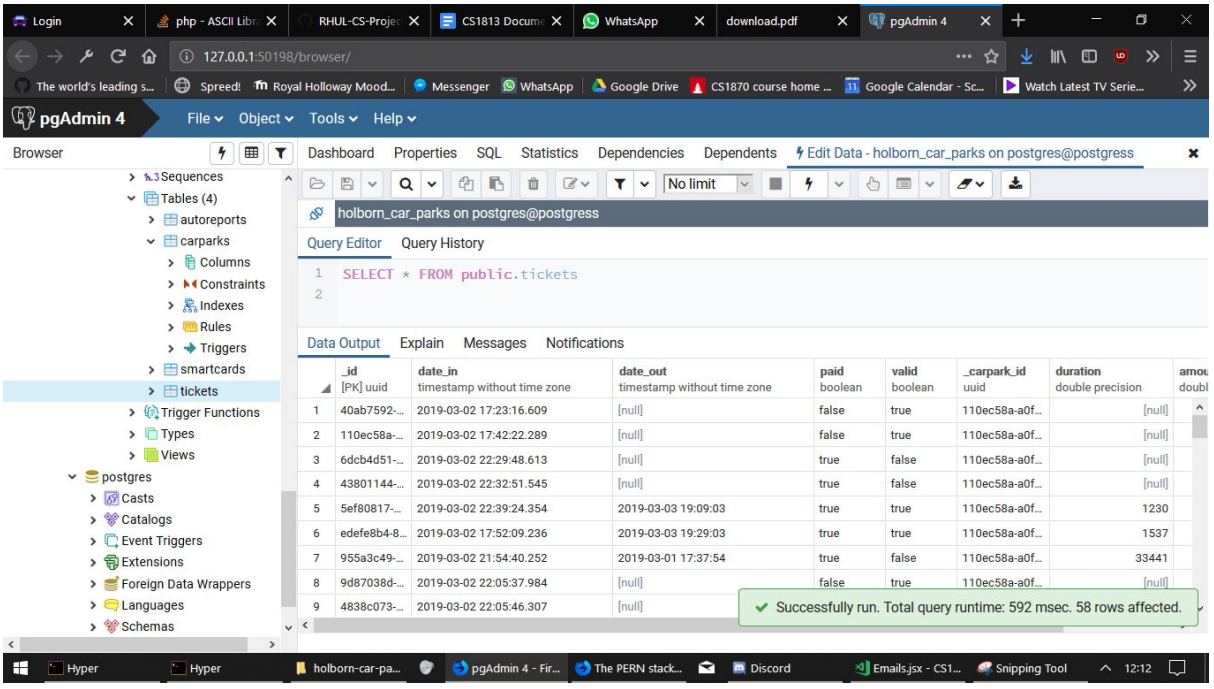
Service app:

We used the PERN stack to build our app.

Postgres:

Postgres is a relational database management system Due to the predictable nature of the data, we chose postgres as our database of choice.

Here is a screenshot of one of the tables:



Express:

Express is a node js framework which runs on the on the server used for building the api. Express also handles routing, and serves the webpages.

React:

React is a javascript framework used for building user interfaces, we decided to use this as it allowed us to break down the user interface inter logical components.

Node (& NPM):

To execute javascript code server side we used the Node.js engine. This allowed us to have the same language on the frontend and the backend.
We also used NPM as our package management system, this allowed us to keep track of modules and interface with our build options. To build the project we used the npm package webpack - this bundles all our files together and ensures they will run in the browser. We tested the build with our express server.

Ticket machine client & Barriers

Maven:

Describes how software is built, and second, it describes its dependencies. It provides an organised structure for out code and makes it easy to build and deploy.

Java:

We used java as our language of choice for the ticket machine and barriers. This is because we cannot be sure of the architecture used on the barriers. Java code can be written once and run anywhere thanks to the JVM.

Javafx:

Javafx is a modern UI library, we used it with scene builder to create the ticket machine UI and barriers UI.

General code organisation

Git:

Git is a Version Control System which we used for managing the code.

After making changes, we small changes we created commits which can be used as history to help with debugging and documentation.

We made sure to use meaningful commit messages following a name convention:

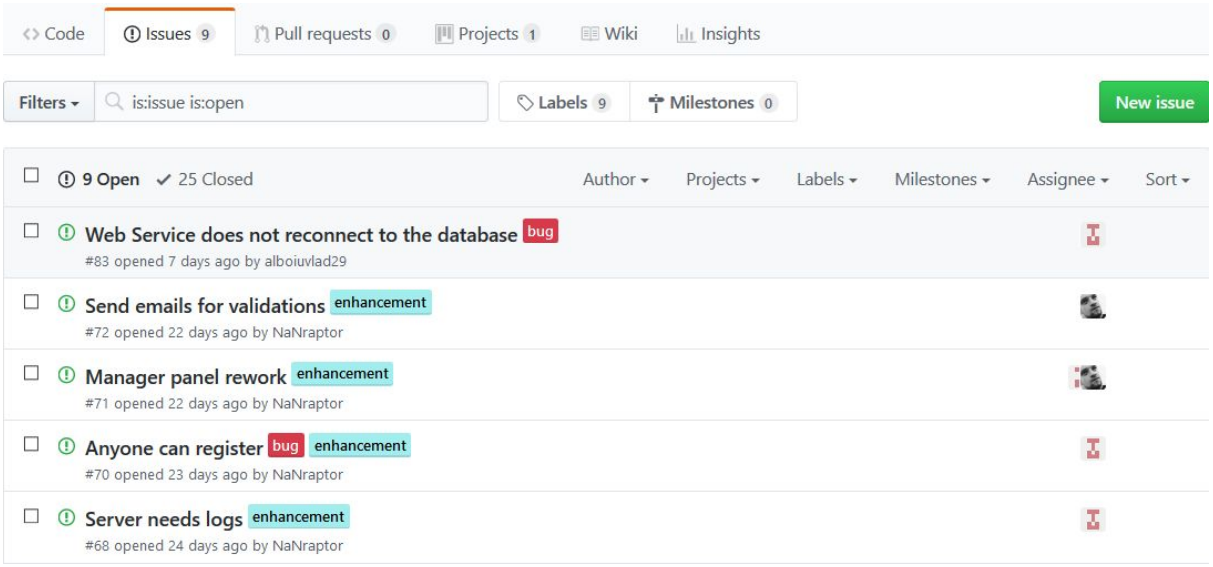
- One word which best describes the changes
- Followed by a dash and 50 characters or less describing the changes

Originally we used separate branches and created pull requests to the master, however it became clear that a better strategy was to rebase and merge the branches with the origin directly so we started to do this after.

We used github as a remote location where we could share project files between group members.

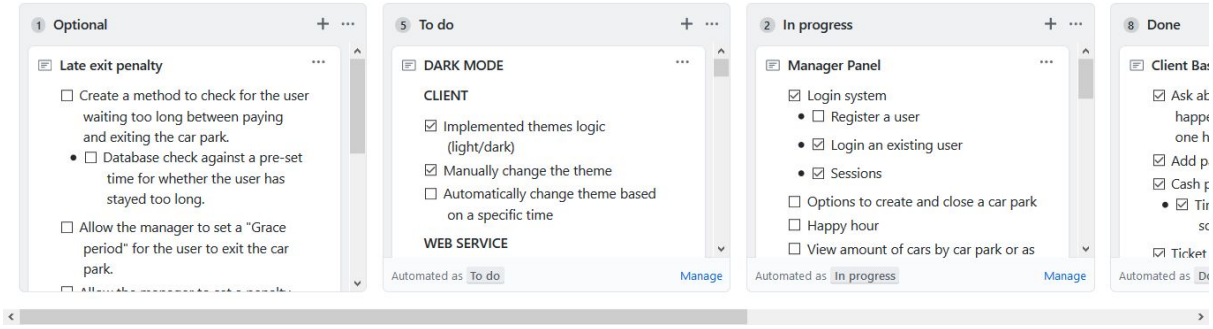
Issues:

One problem we found working as a team was that many issues and bugs were introduced and needed fixing. We found the issues tab on github a great way of keeping track of these.



Projects tab:

Another useful mechanism we found was the projects tab on github, this let us keep track of what was being worked upon.



// .. Build instructions

Web app: (ensure node js and git are installed)

1. Clone the repo:

```
git clone https://github.com/RHUL-CS-Projects/CS1813_2019_05.git
```

2. Go to the correct directory:

```
cd holborn-car-park-service-app
```

3. Install the dependencies:

```
npm install
```

4. Build the project

```
npm run build
```

5. Run the dev server

```
npm start
```

You did it! Now just open a browser window and type “localhost” in the url bar.

Ticket machine (ensure you have jdk 11.0.2 and maven installed):

1. Clone the repo:

```
git clone https://github.com/RHUL-CS-Projects/CS1813_2019_05.git
```

2. Go to the correct directory:

```
cd holborn-car-park-client
```

3. Install the dependencies

```
mvn install
```

4. Run the program

```
mvn exec:java
```

You did it! You may have to edit the config file so that it connects -> see user guide.
Don’t worry if you can’t get this to work, we also provide installers which you can use -> see user guide.

Barriers:

1. Clone the repo:

```
git clone https://github.com/RHUL-CS-Projects/CS1813_2019_05.git
```

2. Go to the correct directory:

```
cd holborn-car-park-barriers
```

3. Install the dependencies

```
mvn install
```

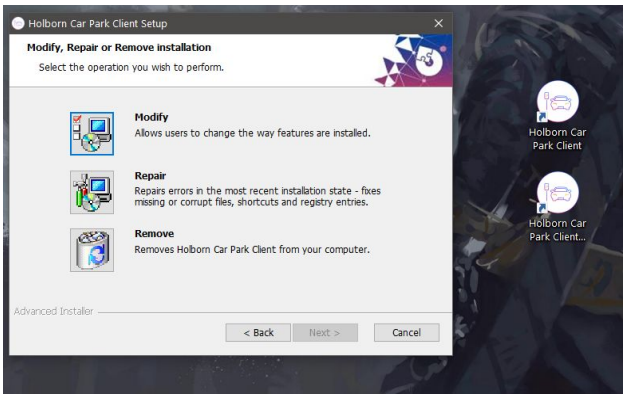
4. Run the program

```
mvn exec:java
```

You did it! You may have to edit the config file so that it connects -> see user guide.
Don’t worry if you can’t get this to work, we also provide installers which you can use -> see user guide.

Production files:

There is also install files available for the project, thesbe can be downloaded from: https://holborncarpark.com/updates/client/download.
When prompted for a config, add https://holborncarpark.com to the webservice.



// .. Testing

Tests were carried out throughout the project. Every time we made changes we would test them by running the program, this ensured that the master branch was as stable and had few bugs. Occasionally, when bugs were discovered we would write issues in the issues tab and tackle these separately.

Unit testing

One method of testing was unit testing, this involved writing function to test that

Unit testing was useful for certain aspects of the project.
Here is an example of a unit test used in the java client:

```
public void testValidTicket() {
     sleep(3000);
     clickOn(".button");
     sleep(500);
     write("cb62a50c-dd53-4856-8882-53aa3ae1c767");
     sleep(500);
     FxAssert.verifyThat(".label", LabeledMatchers.hasText("Your ticket is valid!"));
 }
```

You can see all the unit tests in the test file on the repository.

Ajax

Another form of unit testing was testing the api worked as intended using console.logs(). We tested it with GET and POST ajax requests to get data to the UI. We then looked at the objects returned to check they were what was expected:

```
$.ajax({
          url: '/api/carparks/',
```

```
        type: 'GET',
        success: (data) => {
            this.setState({
                carparks:  JSON.parse(JSON.stringify(data)),
                selectedCarpark: JSON.parse(JSON.stringify(data[0]))
            });
            console.log(JSON.parse(JSON.stringify(data)));
        },
        error: (xhr, status, err) => {
            console.error('', status, err.toString());
        }
    });
```

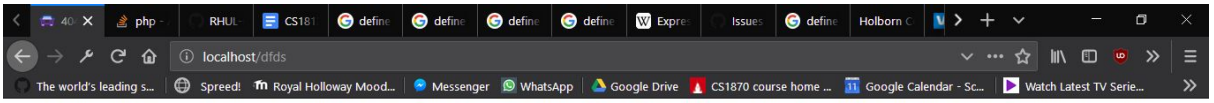
Alpha testing & Acceptance testing:

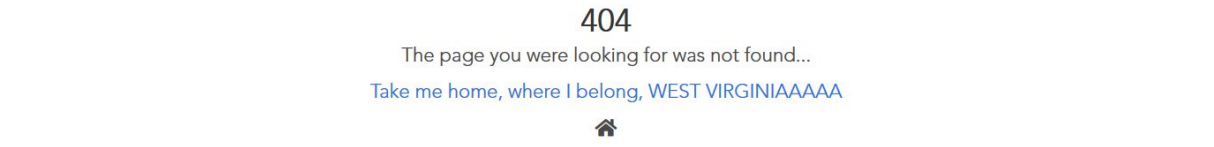
Once we had a MVP (minimum viable product) we tested it against the requirements

We also tested our pre-release build using alpha testing - this involved us as the developer running the MVP as though we were the intended users and checking it functions as intended.

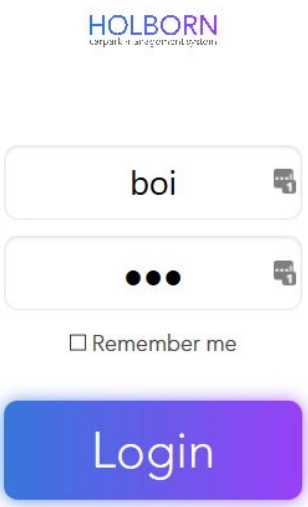
You can see our tests :

1. Checking the user goes to an invalid page (test passed)

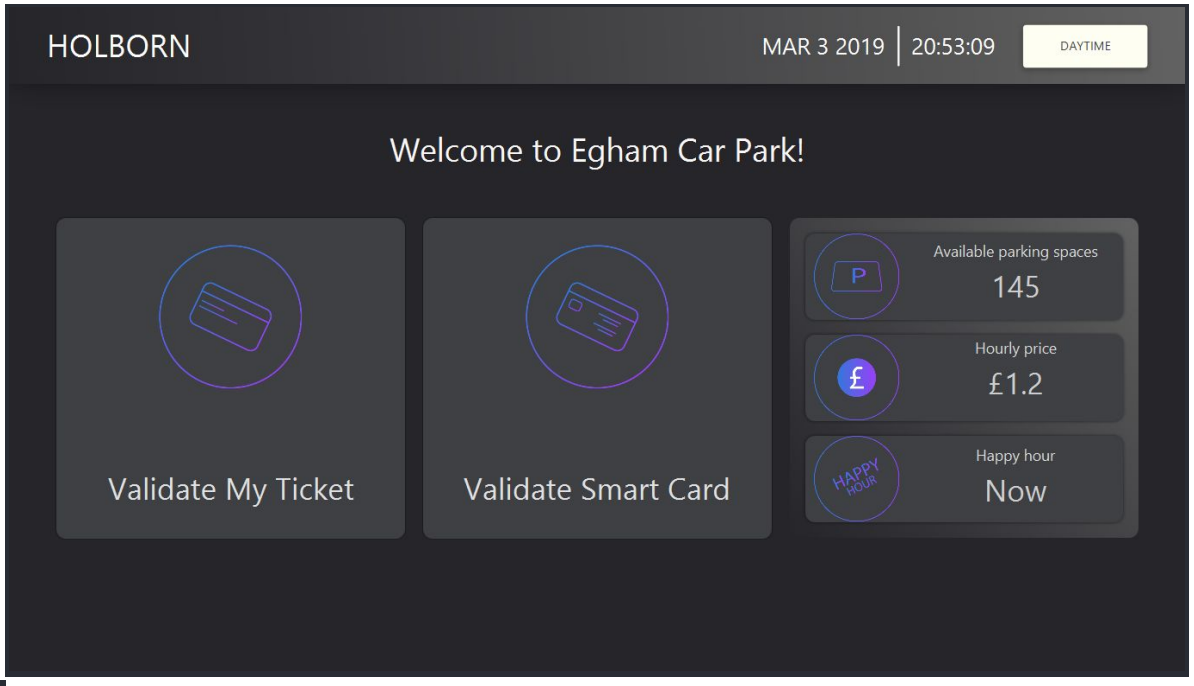




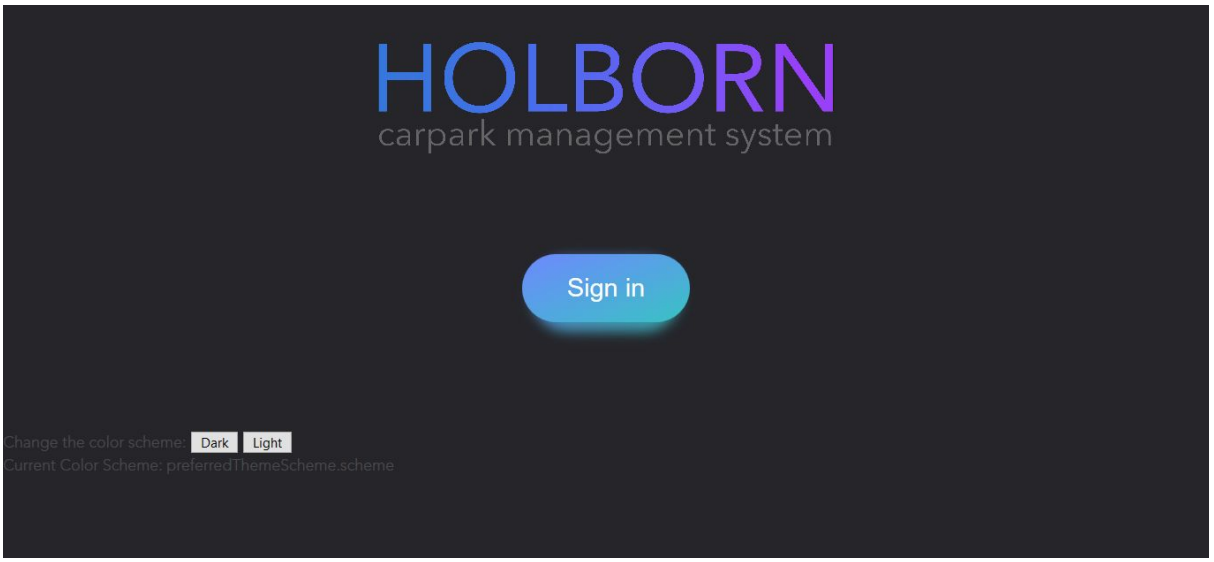
2. Mobile mode (test failed): The app works in mobile mode, however once you log into the map you cannot scroll due to a bug in the css setting overflow to hidden. This bug can easily be fixed in the future.



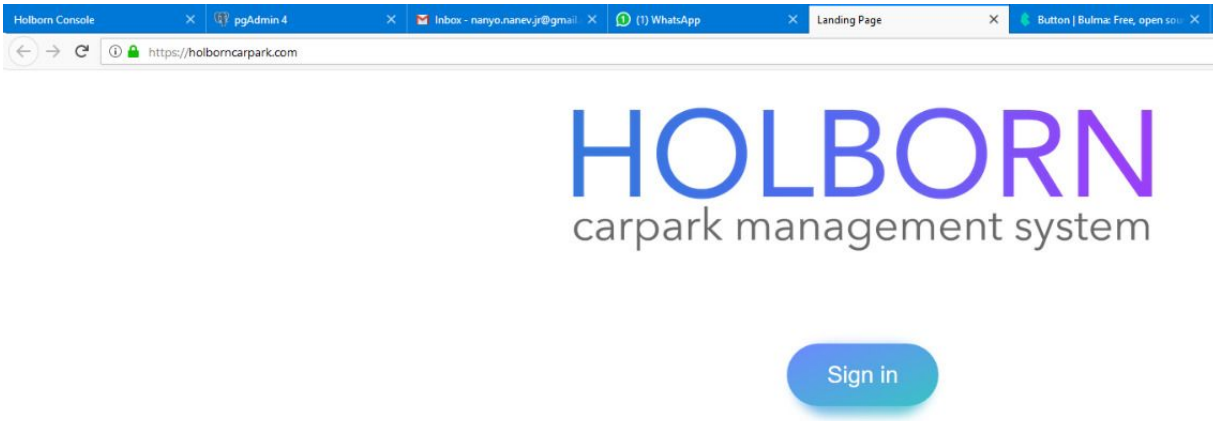
3. Dark theme: This works perfectly in the javafx app



4. It also runs well in the web browser however some things are not always styled correctly. This can be fixed with more css.



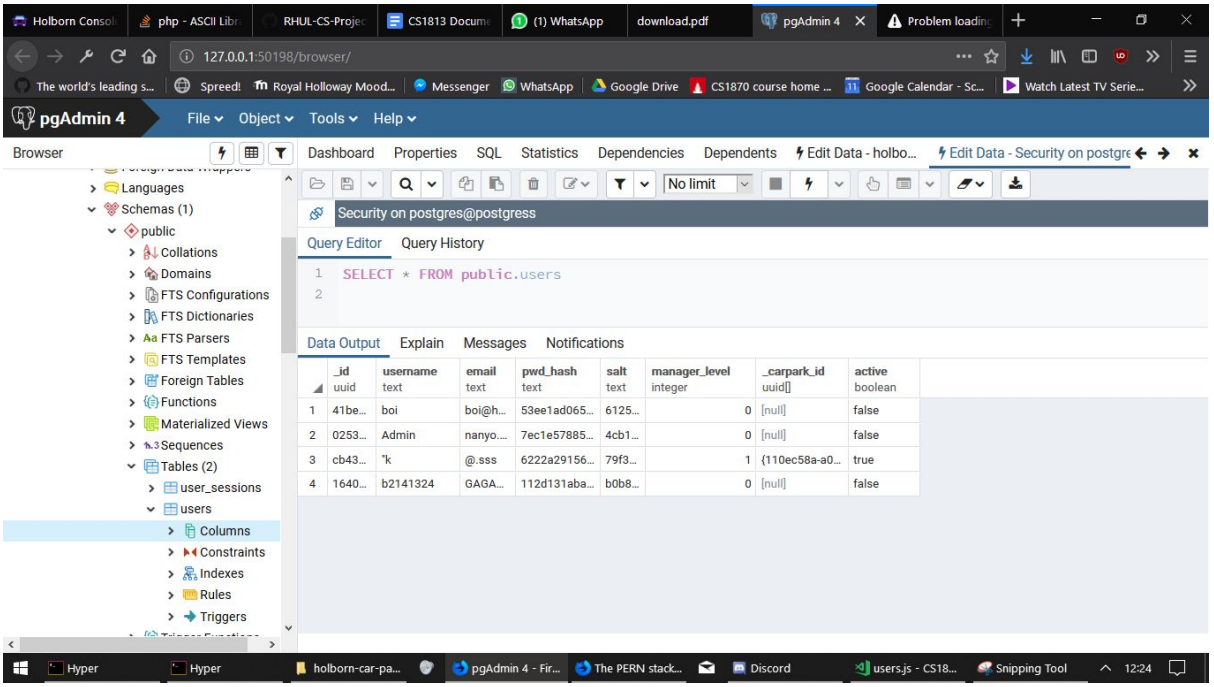
5. Domain new, SSL encryption and secure log in + data storage:



We has and salt the passwords and use crypto libraries

```
   let u_id = UUID();
   let salt = util.genRandomString();
   let hash = crypto.pbkdf2Sync(G.default_pwd, salt, G.hash_iterations, 64,
'sha512');

   let params = [u_id, req.body.username, req.body.email,
hash.toString('hex'), salt, req.body.manager_level, req.body._carpark_id,
true];
```



6. Do

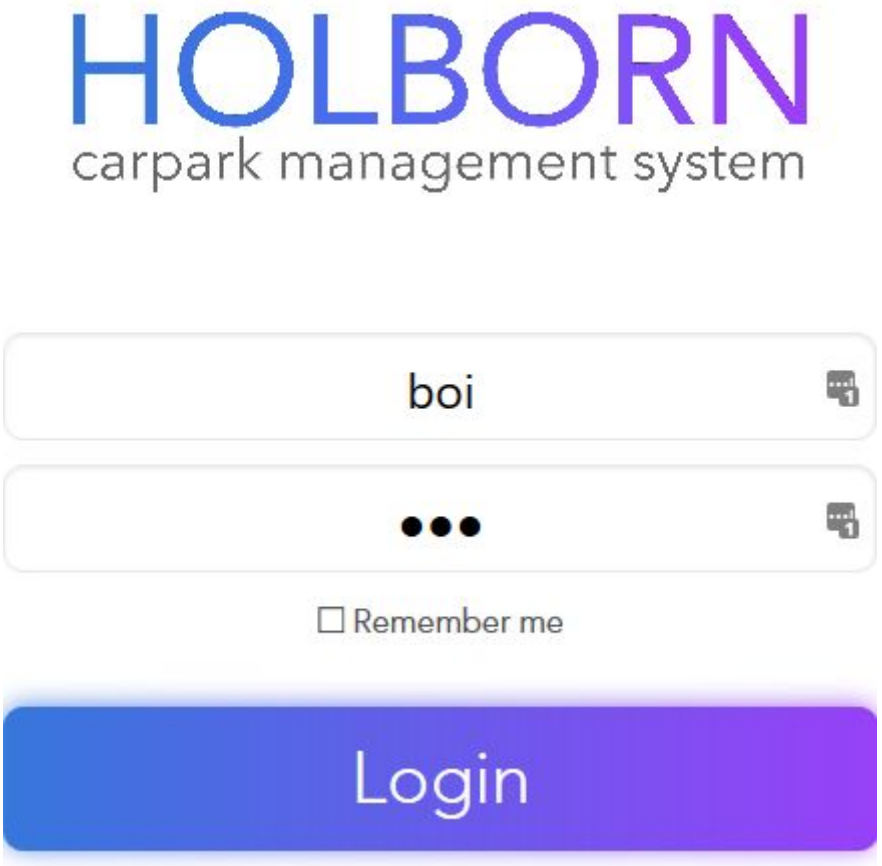
// .. User manual

Below we have created a user guide for the various aspects of the program**, some of these things do not yet work fully** however it is written as though the full product.

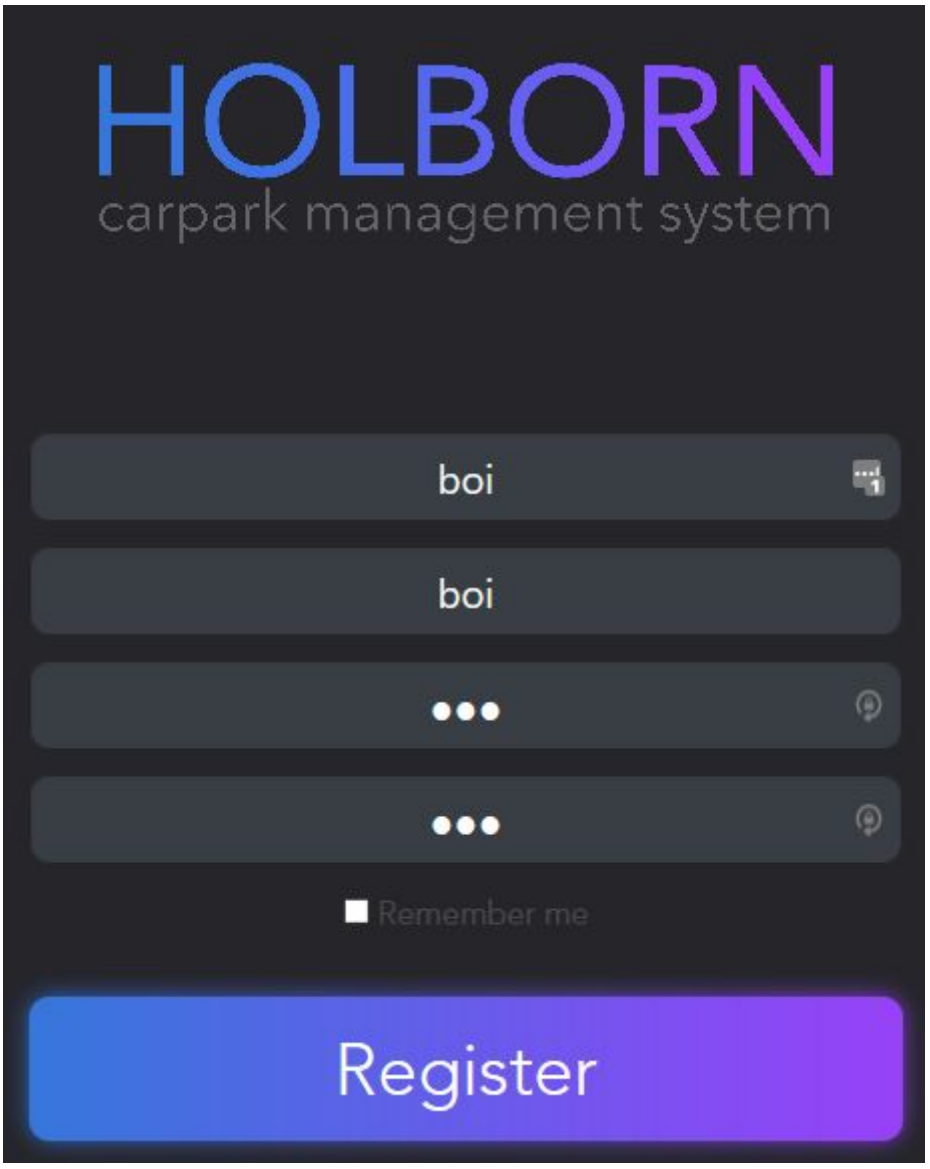
Manager Console

Thank you for choosing Holborn as your carpark system, here is a guide to show you how to use the webpage.
The admin console is a piece of software designed to allow an admin to manage the car-park and generate reports. Below are instructions on how to do this:

Navigate to the URL: https://holborncarpark.com/

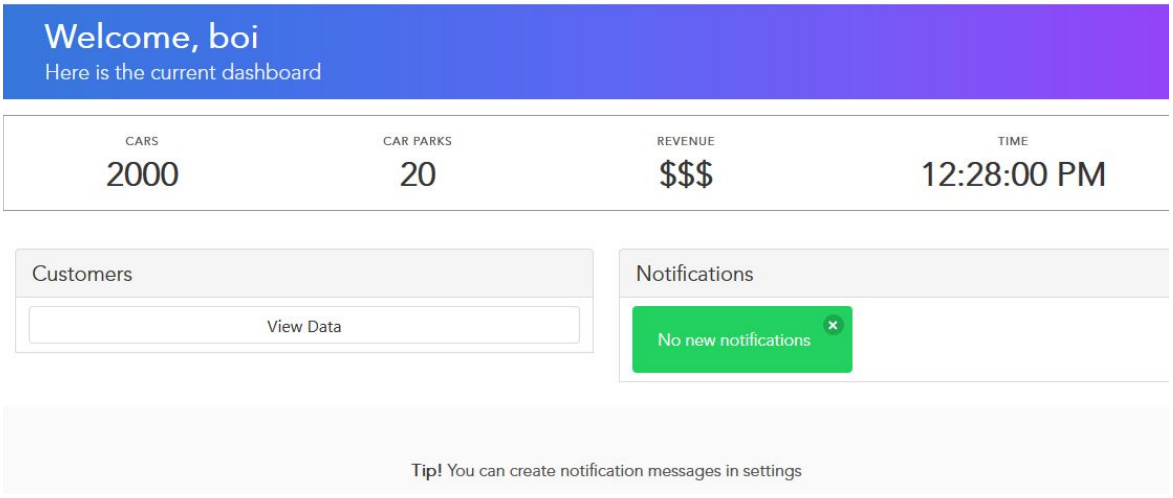


Once online you will be prompted to sign in, use your admin credentials.
If you do not have any credentials, you create an account here:
https://holborncarpark.com/register

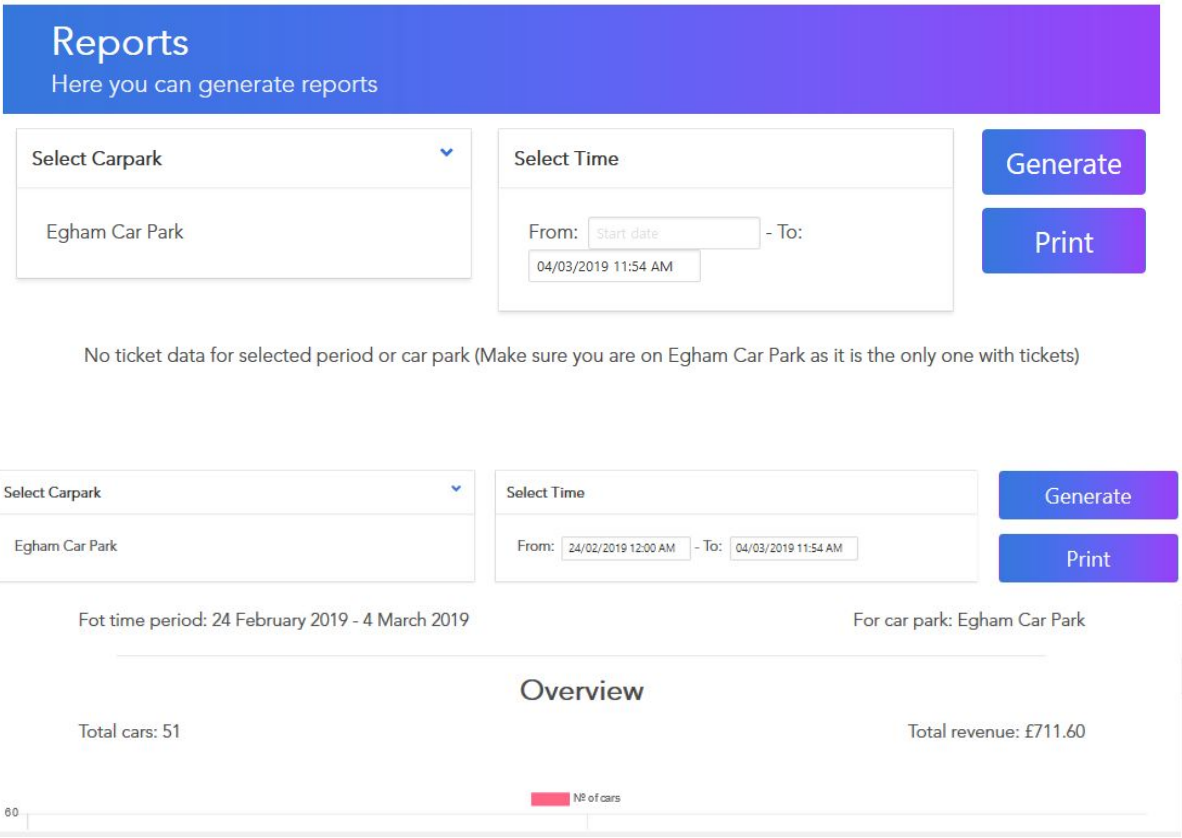


Please use a real email address as it is needed for automatic reports.
These credentials can be changed in the settings tab - a drop down from the user menu. From here you can also log out, which will end you current session and delete the session cookie.

Once logged in, you will be directed to the dashboard, this shows you some basic information about the company and allows you to view notifications. You can post new notifications from settings.



From this screen you have many options. For example, you could choose to generate a report, this is achieved by selection the report icon in the sidebar. From here you will be prompted to select a time period to generate between and select which carpark the report should be for. Once you are satisfied with the parameters click “generate”. Should you wish to save the report press print, this will download the current page in PDF format.

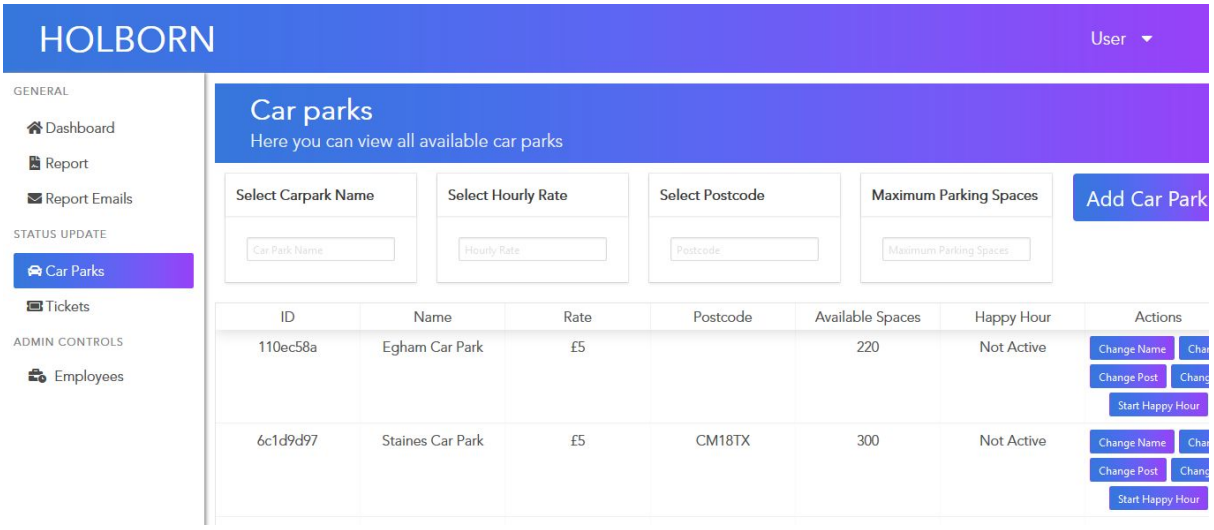


Another use case is scheduling reports, this can be done in the emails tab. An email will be sent to your account however often you specify, for example every day.
Emails will be sent by:
holbornreporting@gmail.com
(password &EaY89gWD)

This account should send emails only and never received, however you can view past reports in the sent items.

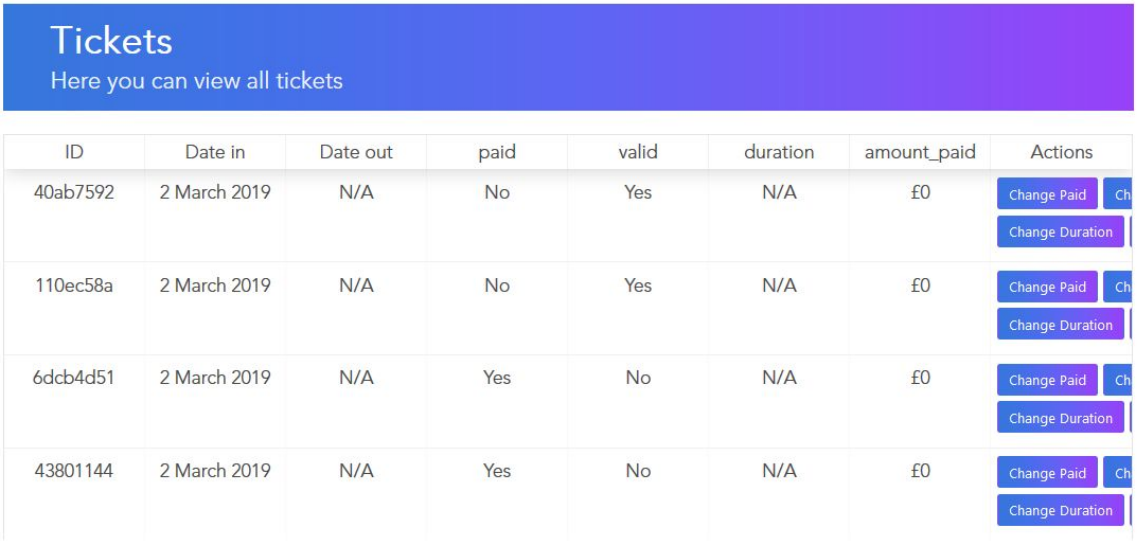
You can also view past reports in the tables provided in the app itself!

Another potential use case is to view the current state of a car park, this can be done in the carpark tab, this view simply displays each car park and information such as Carpark Name; Hourly Rate; Postcode; Maximum Parking Spaces and if happy hour is enabled. Here you can also add and remove car parks.



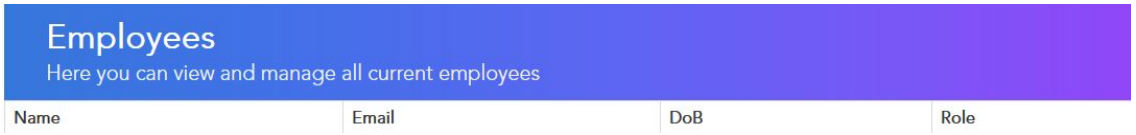
Here you can create new carparks (however you cannot delete a carpark ever as it contains important information such as employee data which should not be deleted so be careful)

Another similar tab is tickets, this displays tickets as they are purchased and information such as length of stay and the total cost and whether or not the ticket has been paid.

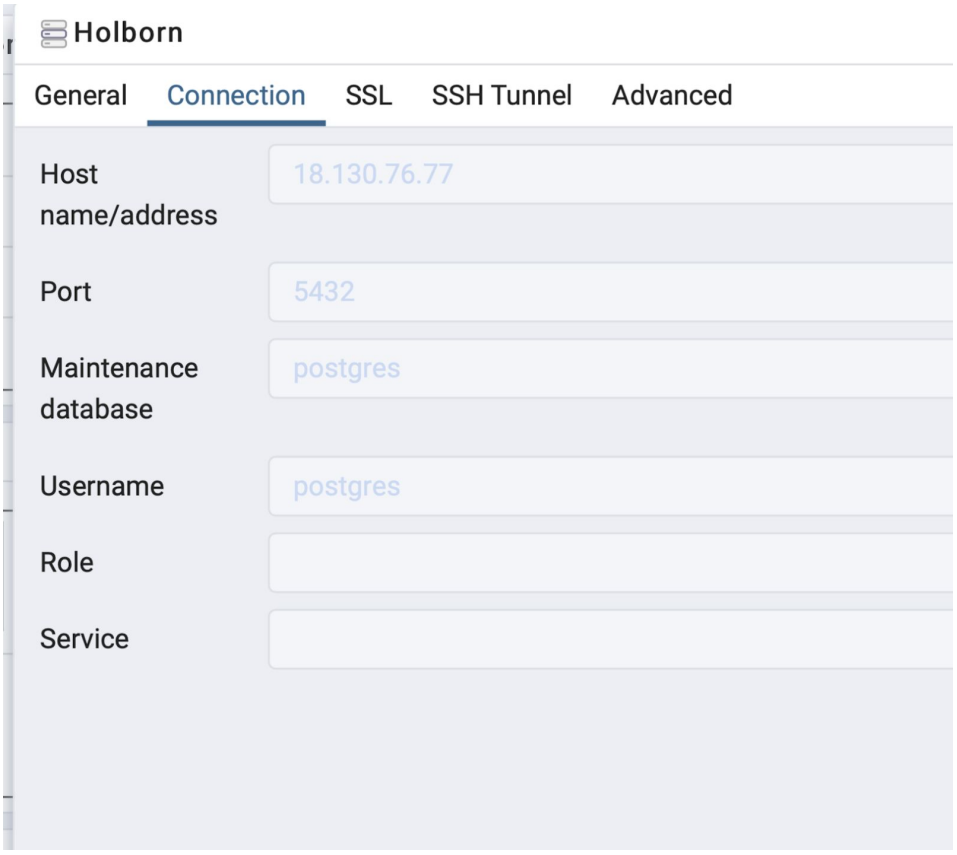


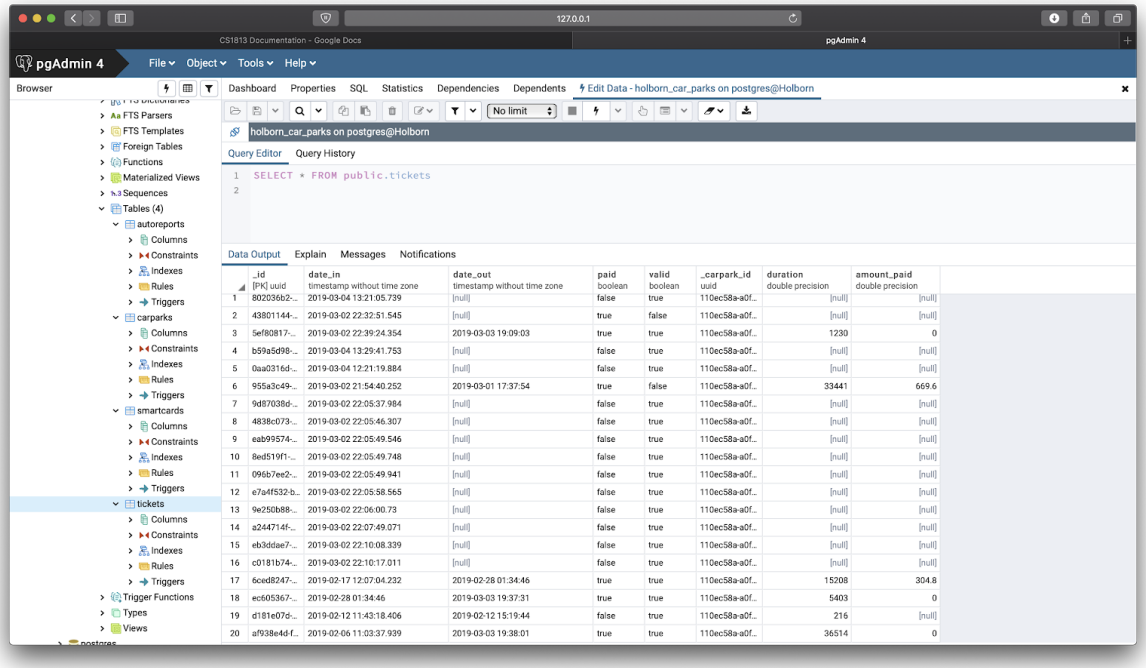
You can use these tickets in the java app!

The employees tab is displays data on the current employees such as their name, email DoB and role.



If you ever need to access the databases manually (which you should not unless you are a developer) you can do it using a piece of software called pgadmin, the location is our website and the username is postgres and the password is postgres.
Once logged in, configure it with the following:





Holborn Car Park Barriers

Please make sure you download the latest version from
https://holborncarpark.com/updates/barriers/download

Holborn Car Park Client

The Holborn Car Park Client is the middleware communicating between the database and the user. It handles all the payments and necessary validations while providing a intuitive interface and real time event driven information about the car park. It also acts as a server for the barriers.
For the client to work, the server must be properly set up.

Note: the client must be installed on a touchscreen-enabled device

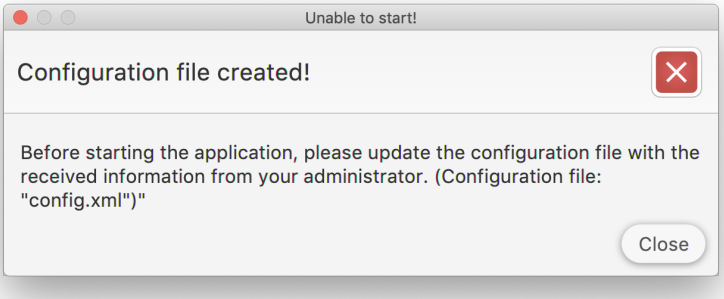
Installation

Please make sure you download the latest version from:

https://holborncarpark.com/updates/client/download

Note: Please make sure to update to the latest version before
Installing the application on a payment terminal is straightforward.
Just run the Holborn Car Park Client Setup.exe and follow the on screen prompts. Make sure to remember the installation path since you will need it to configure the client.

1. Start the application by clicking the new desktop icon on your desktop. The first time you run the application you will get the following alert:



Now configuring the application is the hardest part of the setup since it requires the some informations from the database manager.
2. Go to the installation directory (default is C:\Program Files\Holborn\Holborn Car Park Client\ ) and there should be a file called “config.xml”. Right click on it and press edit. You will have a file containing something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="session_timeout_popup_duration">3</entry>
<entry key="night_time_start">19</entry>
<entry key="car_park_id"></entry>
<entry key="car_park_name"></entry>
<entry key="server_listen_port">4444</entry>
<entry key="webservice">https://holborncarpark.com</entry>
<entry key="session_timeout_seconds">300</entry>
<entry key="night_time_end">7</entry>
<entry key="auto_night_time">true</entry>
<entry key="transaction_finished_delay">10</entry>
</properties>
```

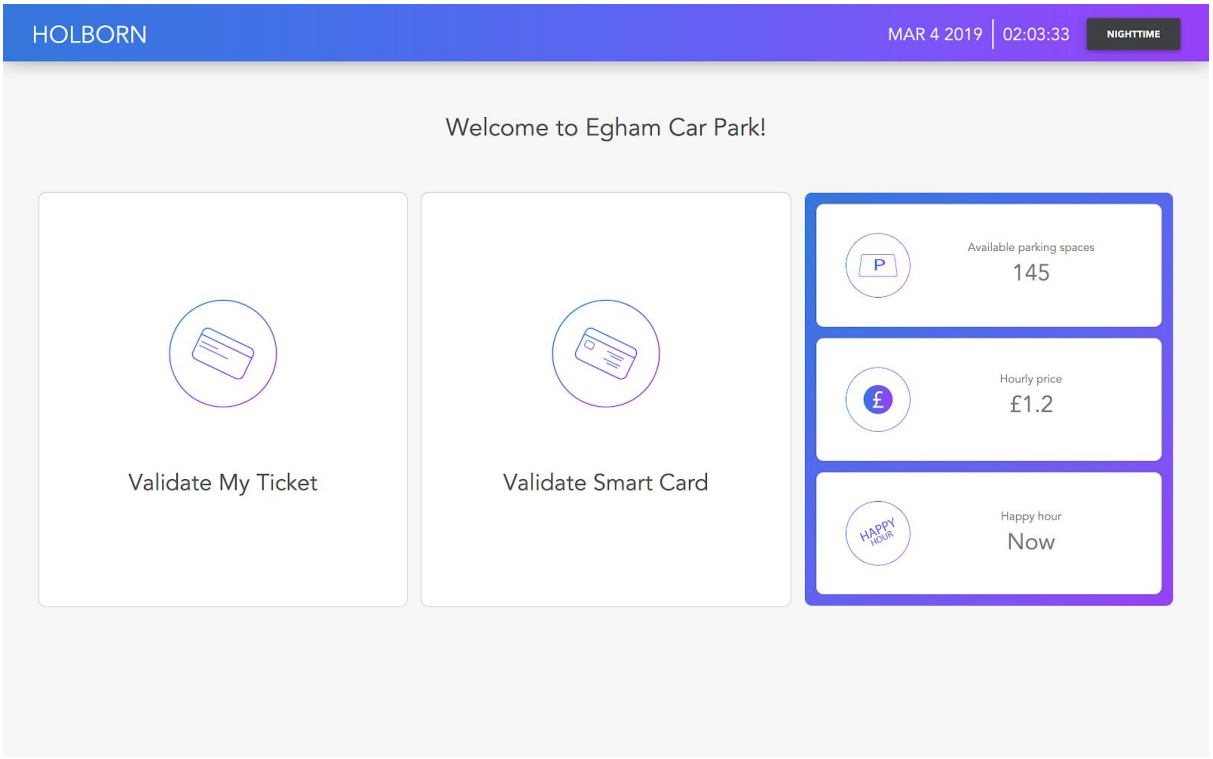
Most of the configuration file is already set to default values giving the manager’s flexibility of customising them further. Although the essential values are not set: `car_park_id` and `car_park_name`. The client authenticates to the server by its name and ID therefor for the client to work, these two values have to be provided and identical to the ones found in the database. For this guide, you will use the predefined database which already contains some car parks. Modify the configuration file with the following values:

```
<entry key="car_park_id">110ec58a-a0f2-4ac4-8393-c866d813b8d1</entry>
<entry key="car_park_name">Egham Car Park</entry>
```

Also with this configuration, the client will connect to the web service in the cloud. If you’d rather run the local instance, just change
`<entry key="webservice">https://holborncarpark.com</entry>` to
`<entry key="webservice">http://localhost</entry>`

Configuration file properties	
`car_park_id`	The ID of the carpark specified in the web service
`car_park_name`	The name of the car park specified in the web service
`server_listen_port`	The port where the the client will accept incoming connection from the barriers (must be the same with the one specified at the barriers configuration file)
`session_timeout_seconds`	The time in seconds before the UI will switch to the landing page if there is no user input
`session_timeout_popup_duration`	The time in seconds for which the message “Session Timeout “ will stay on screen
`transaction_finished_delay`	The time in seconds after which the “thank you” screen will switch back to the landing page
`auto_night_time`	Choose whether or not to use the auto night time switching feature. (true or false)
`night_time_start`	Specify the hour where the night time will start (only if auto_night_time is set to true)
`night_time_end`	Specify the hour where the night time will end (only if auto_night_time is set to true)

If everything was successful, running the application again will start it normally with no errors.



User interface

The UI has been designed with easy of use in mind in an attractive way. It is composed of the following elements:

1. Application bar, the top bar containing the date, hour and nighttime button
2. Scenes area, where all the user interaction happens

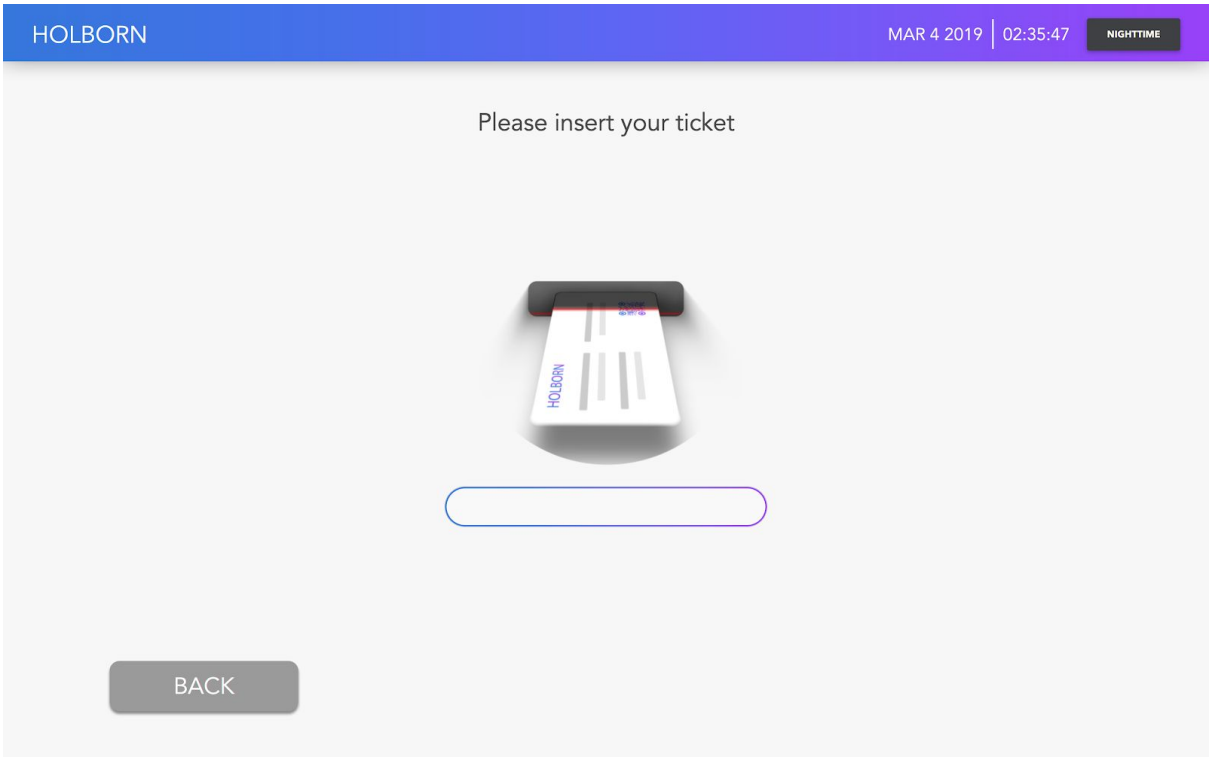
Scenes

1.Landing Page

This is the first page shown to the user. It contains two buttons, one for validating tickets, one for validating smart cards and an information panel.

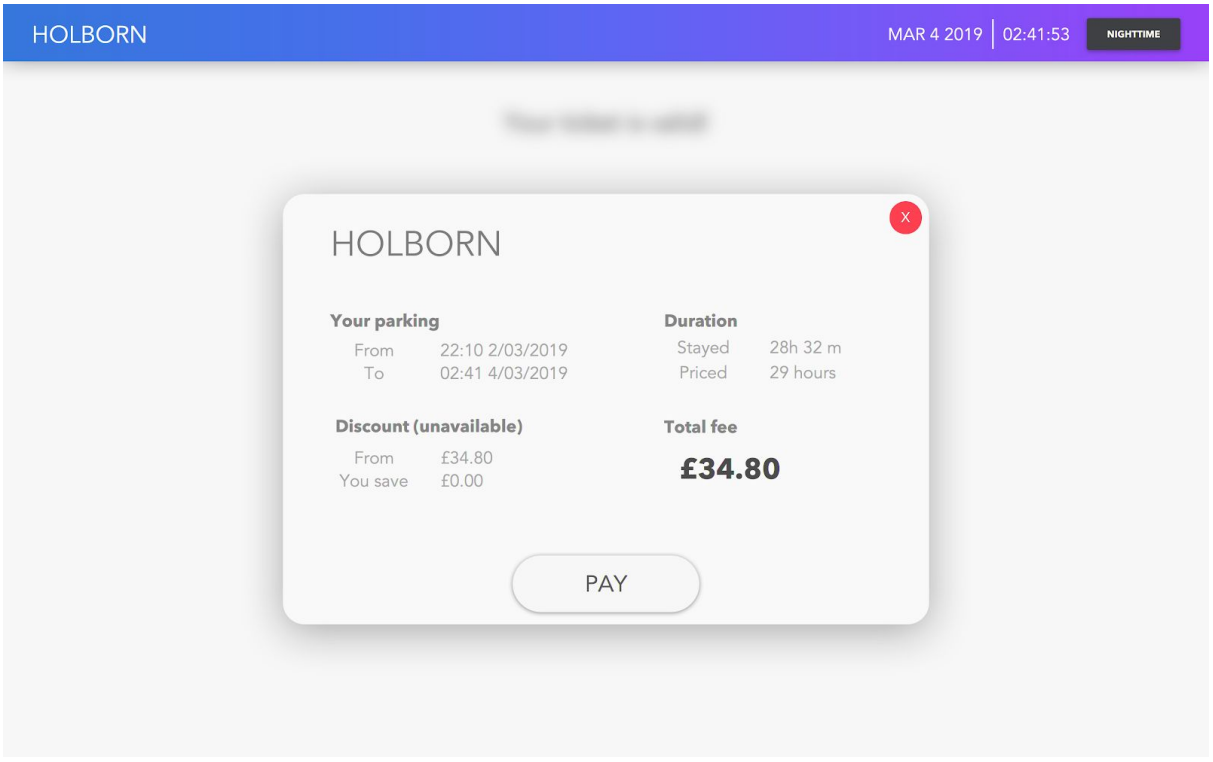
Note: the information panel is event driven, updating dynamically. I.e, a car leaves the car park, the number of parking spaces changes

2. Ticket Check

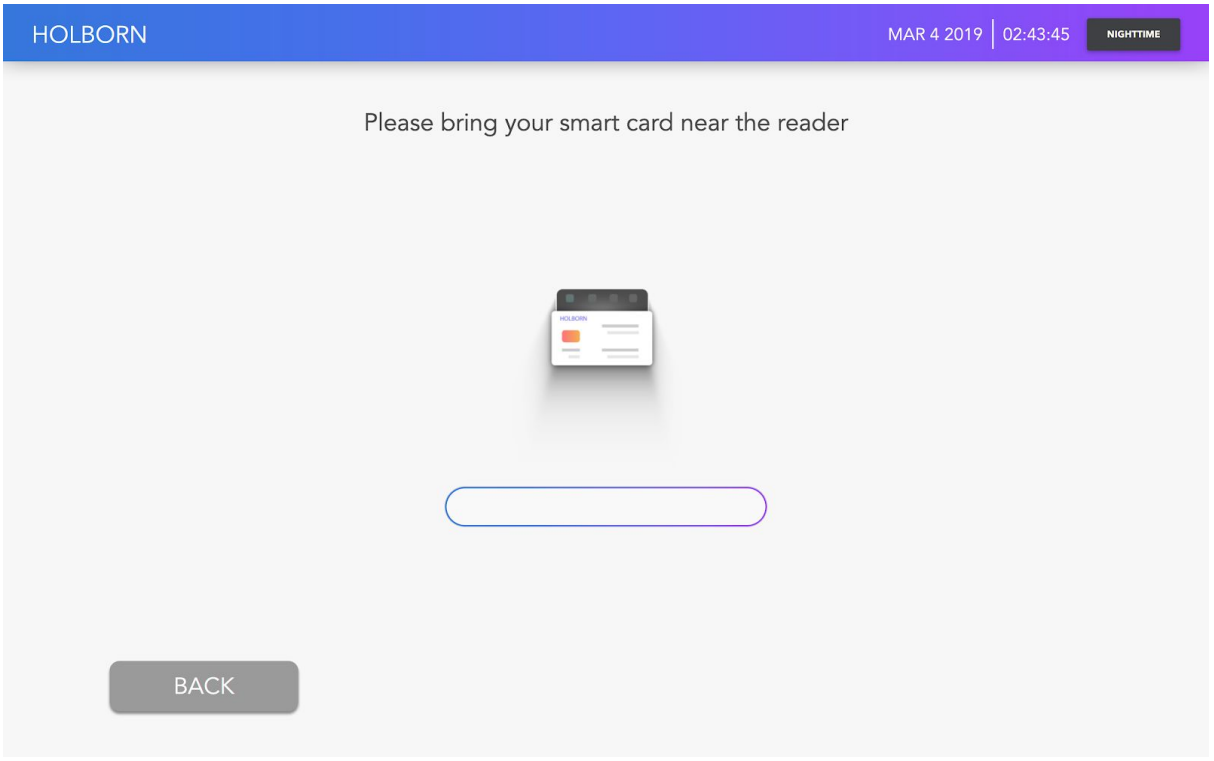


On this page, the user scans a ticket and if the ticket is valid, the ticket details will show up. Otherwise, the user is told to seek assistance from a member of staff.

(For testing purposes use this ticket: 40ab7592-4be0-42ce-8573-f9d49d933b5e)

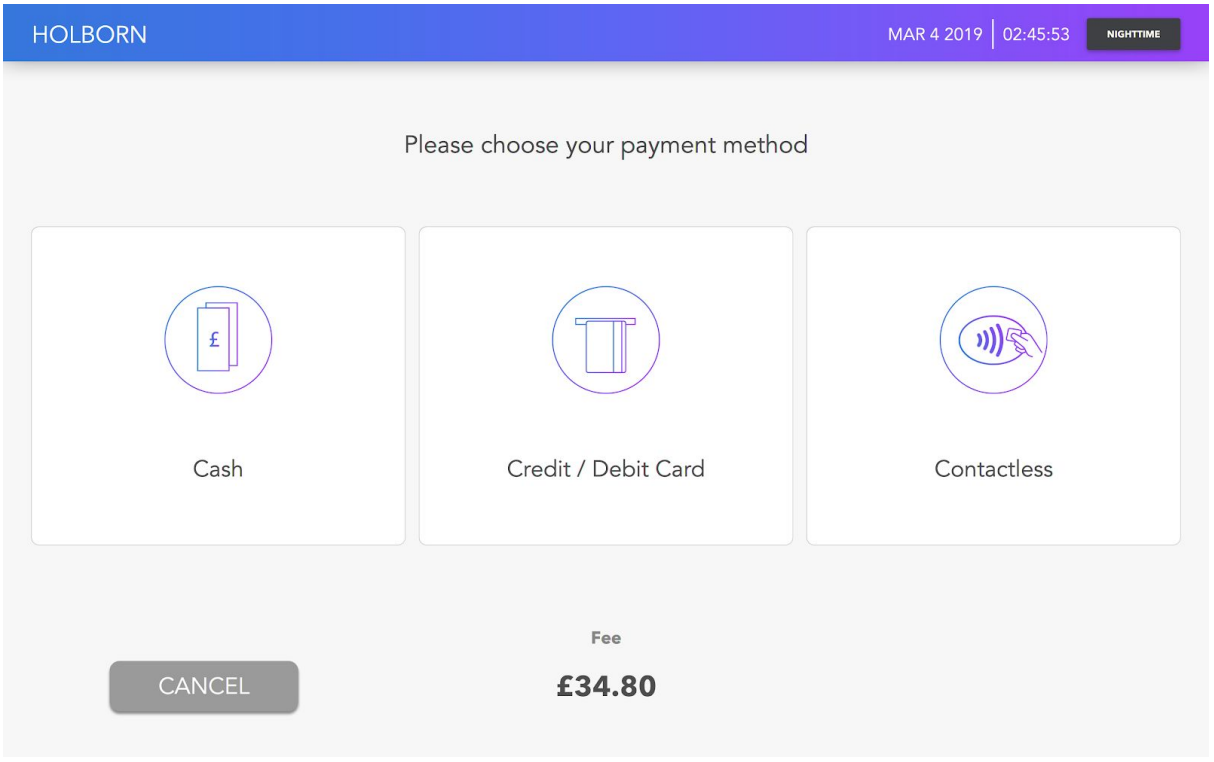


3. Smart card check



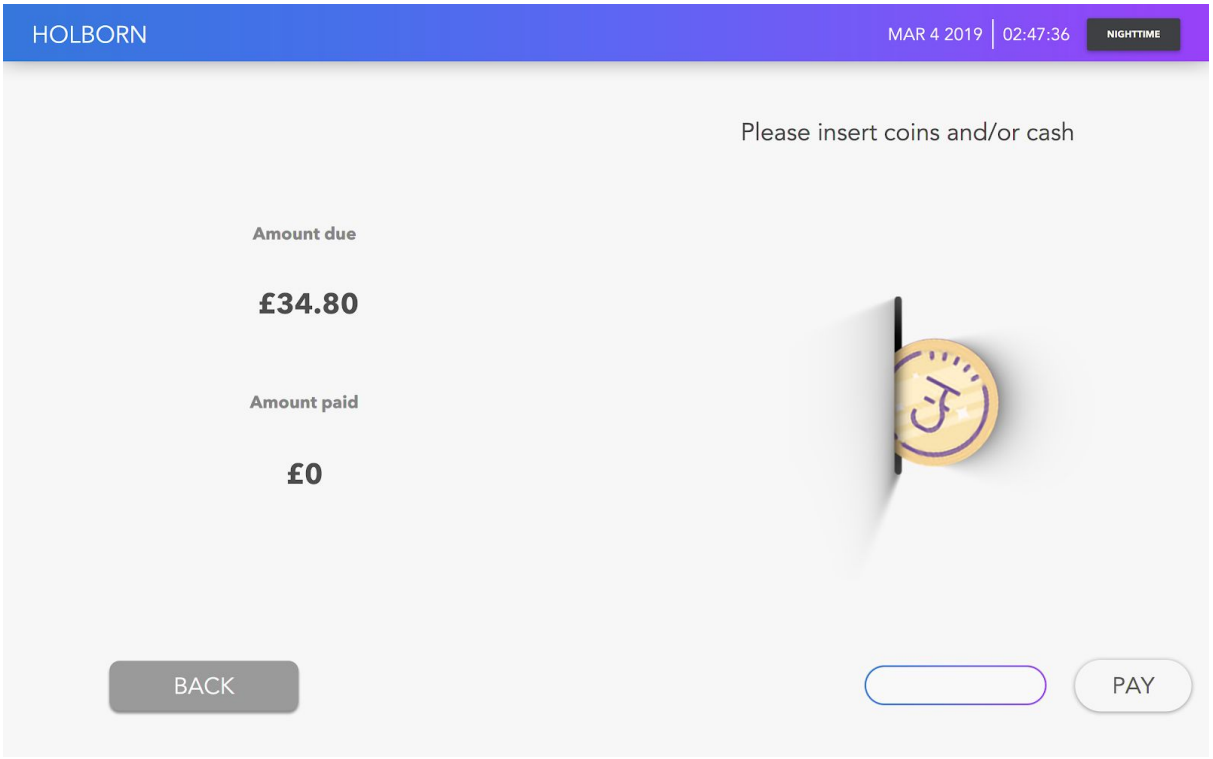
The smart card check UI works the same as the ticket ui but for the smart cards

4. Payment methods



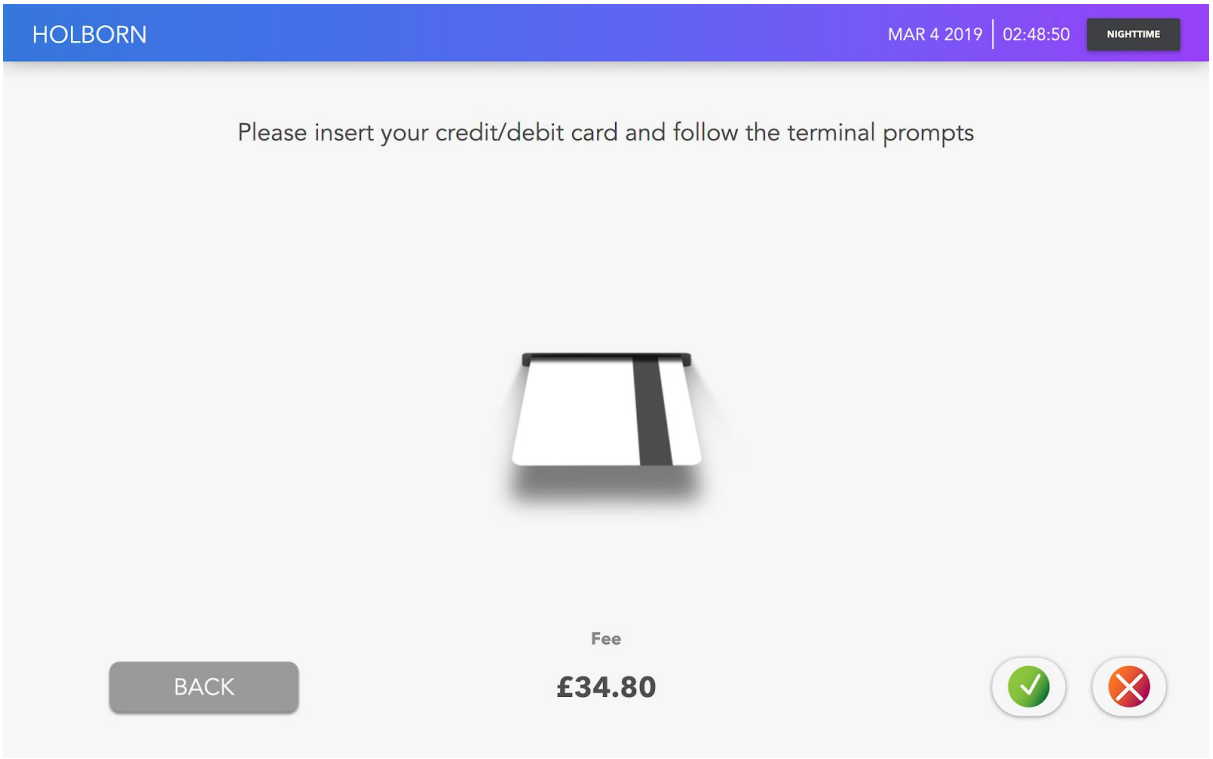
The client supports 3 payment methods: Cash, Classic Card Payment and Contactless.

5.Cash payment



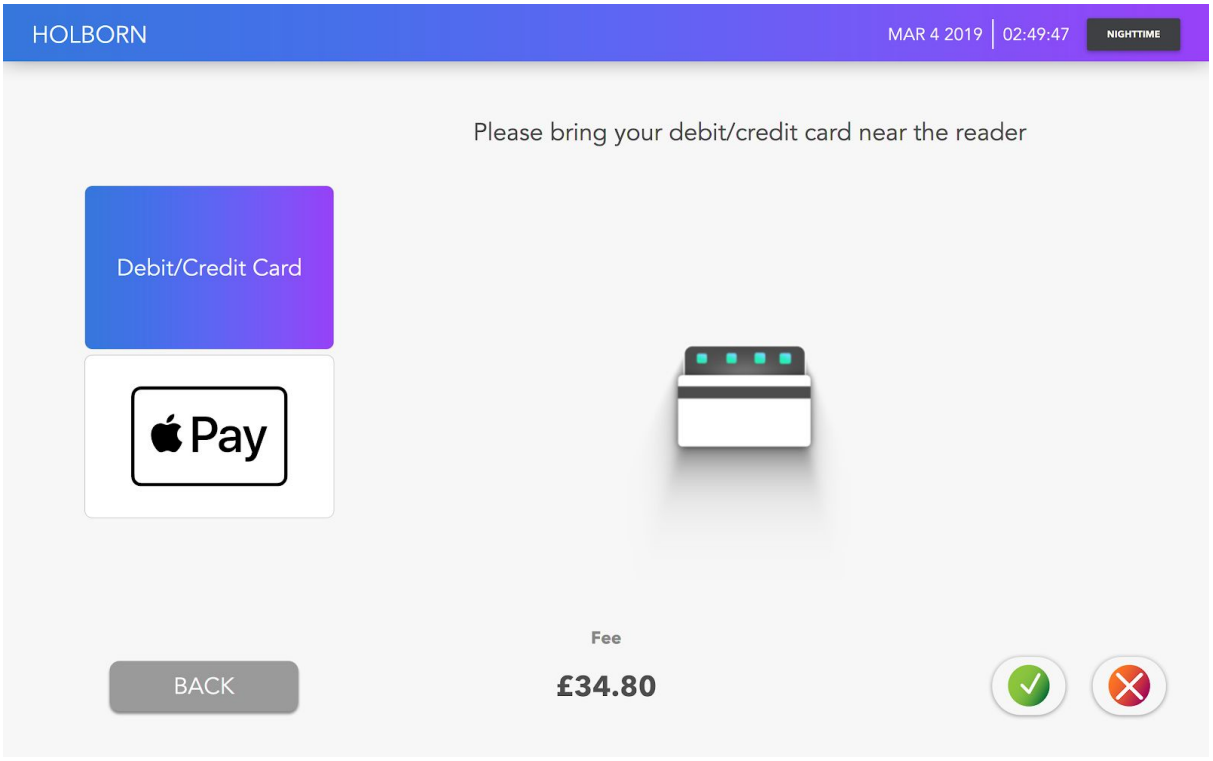
To simulate cash being inserted into the machine, add the amount to the right text field and press pay

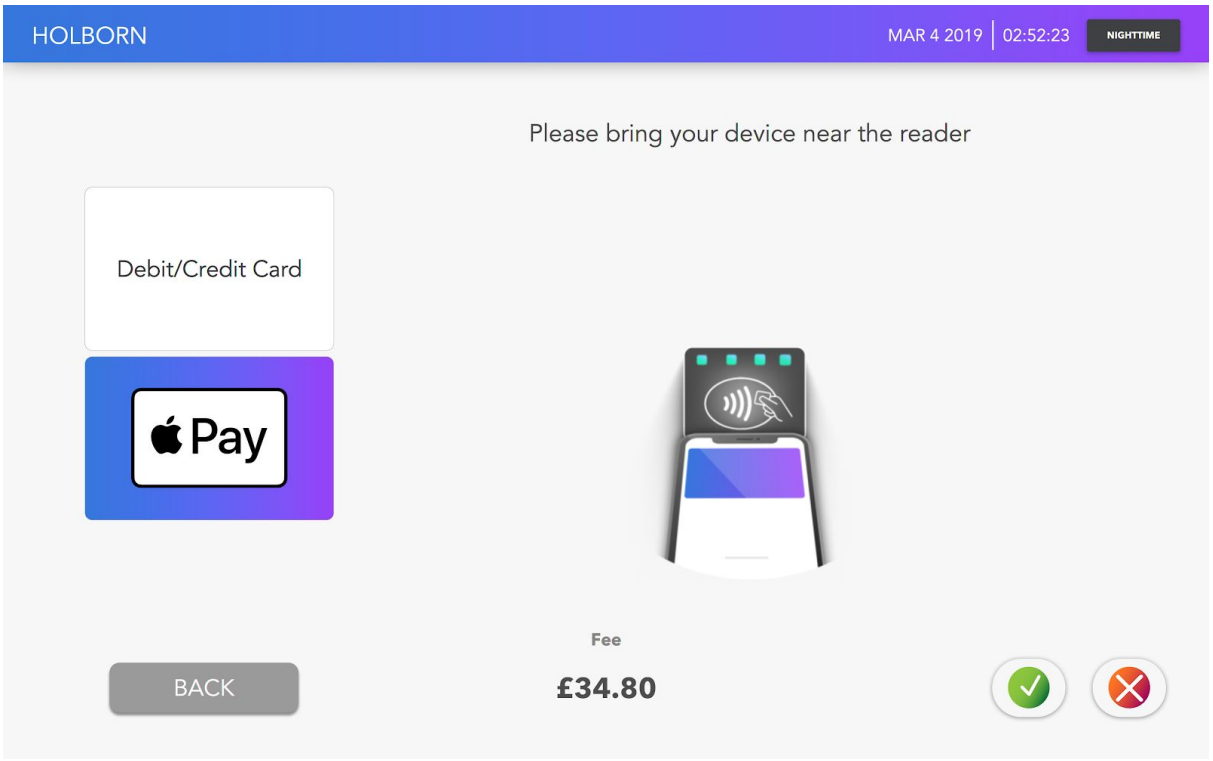
5.Classic card payment



To simulate a transaction, press one of the buttons in the right corner

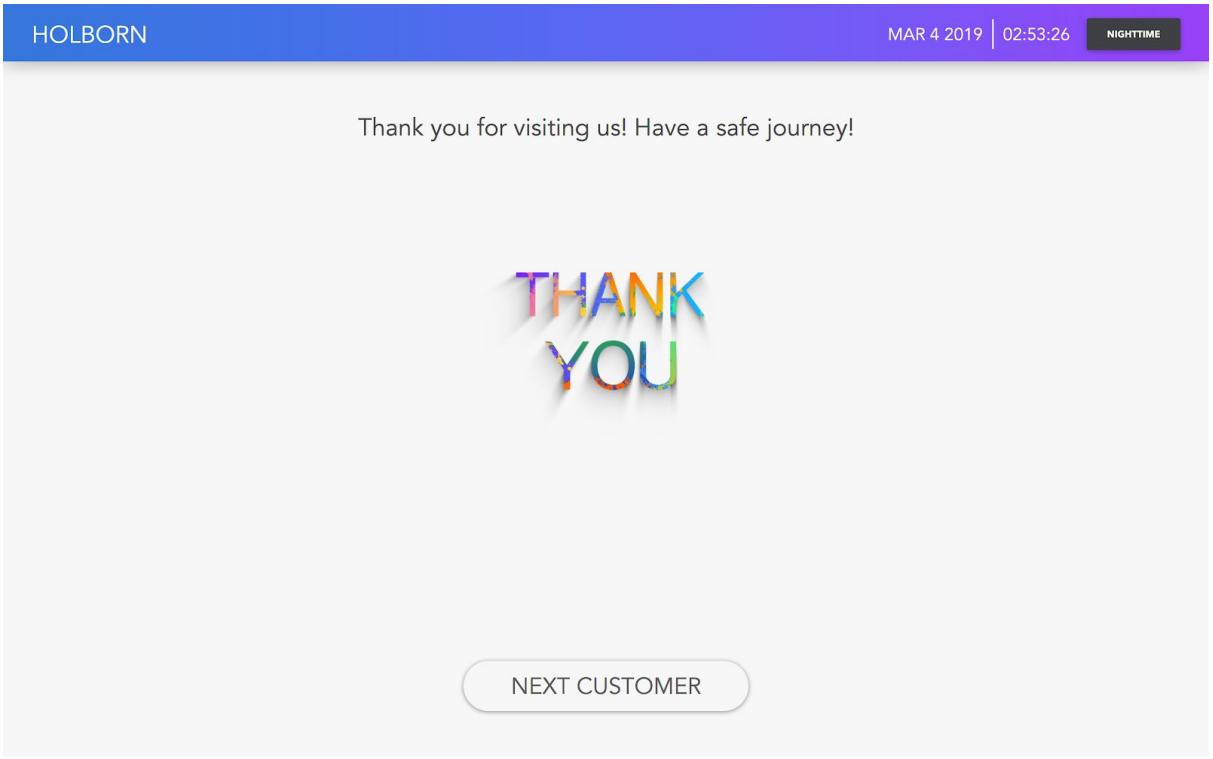
6. Contactless payment





This payment method allows for multiple contactless methods. Choose one from the left hand side and simulate a transaction by pressing one of the buttons in the right hand side

7. Finish scene

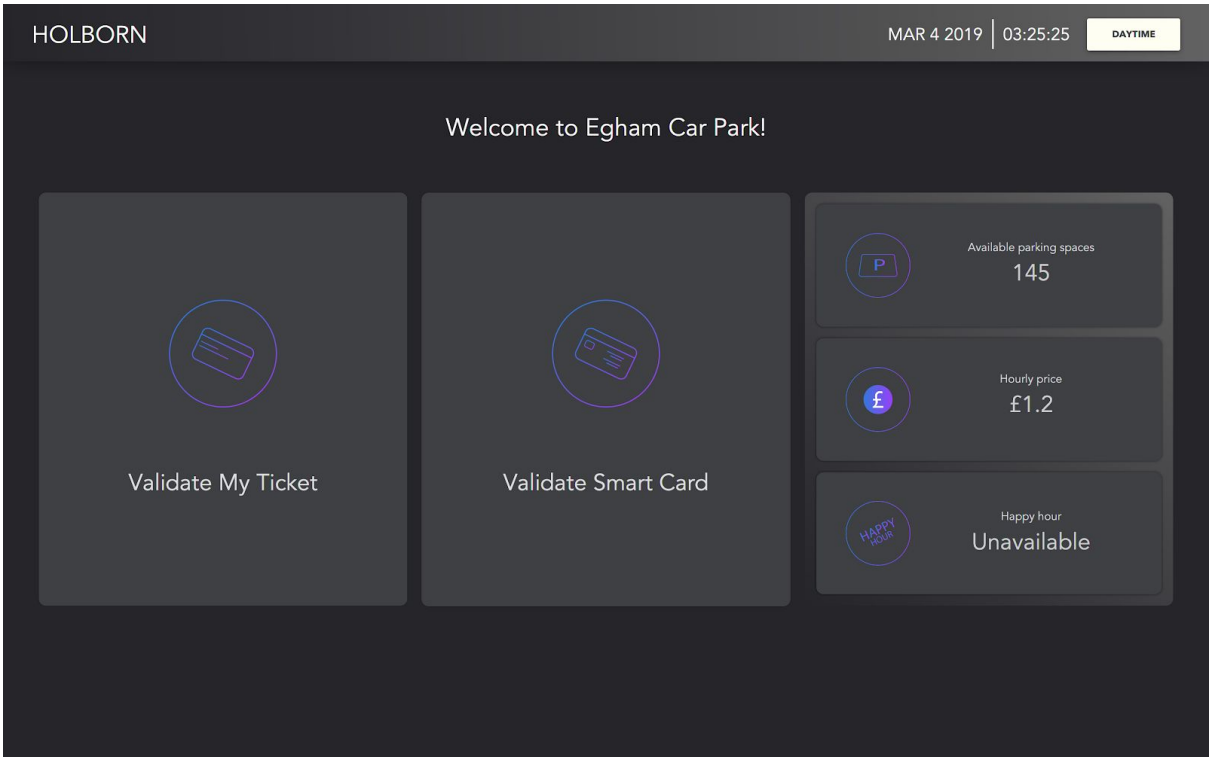


This scene is shown after every session, with its variations

Features

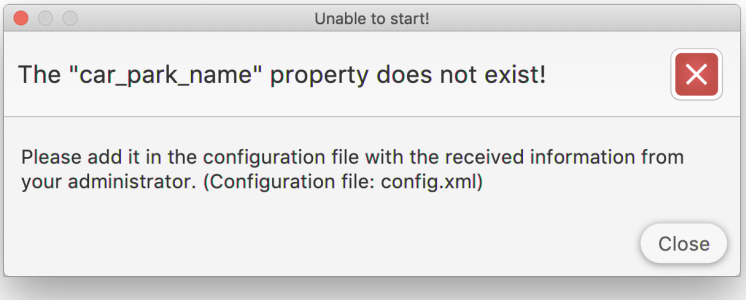
These are small and unique things that make our car park client shine from others

1. Event driven dynamic ui updates
2. Beautifully crafted content and animations
3. Logger: log all that's happening in files to keep track of any errors
4. Session based: user inactivity will cause the ui to switch back to the landing page
5. Have as many clients as you want, they won't bother each-other
6. Multiple payment methods! Yes! You can pay as much as you want with cash and the rest with by card
7. You've inserted too much money? Don't worry the client will give you back the rest.
8. Any amount inserted in the machine on canceling the transaction will be returned back
9. Nighttime!!! Are your eyes tired of being burned by the bright light when you go interact with the terminal? Auto nightime comes to the rescue. Each UI element has been carefully designed to help during the tiring hours

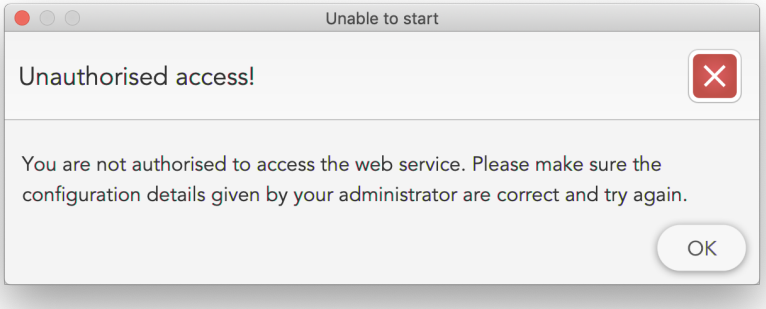

HOLBORN
MAR 4 2019
03:25:25
DAYTIME
Welcome to Egham Car Park!
Validate My Ticket
Validate Smart Card
Available parking spaces
145
Hourly price
£1.2
Happy hour
Unavailable

Troubleshoot

1. Application won't start, receive the following messages:

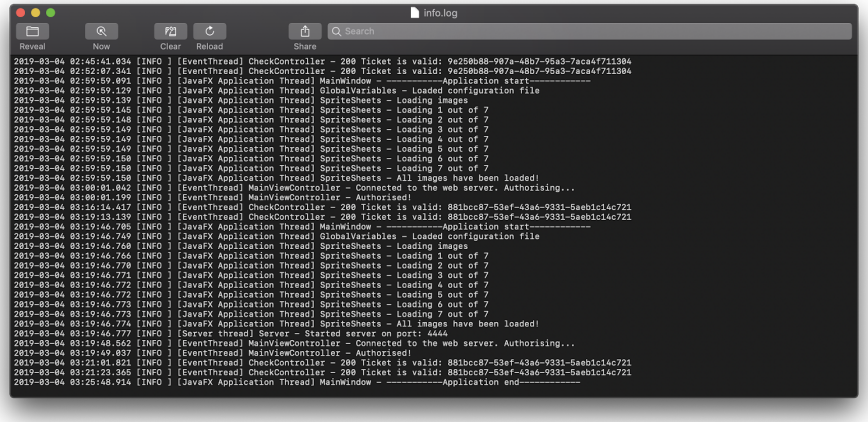


   a. Solution: make sure the following property exists in the configuration file, otherwise add it



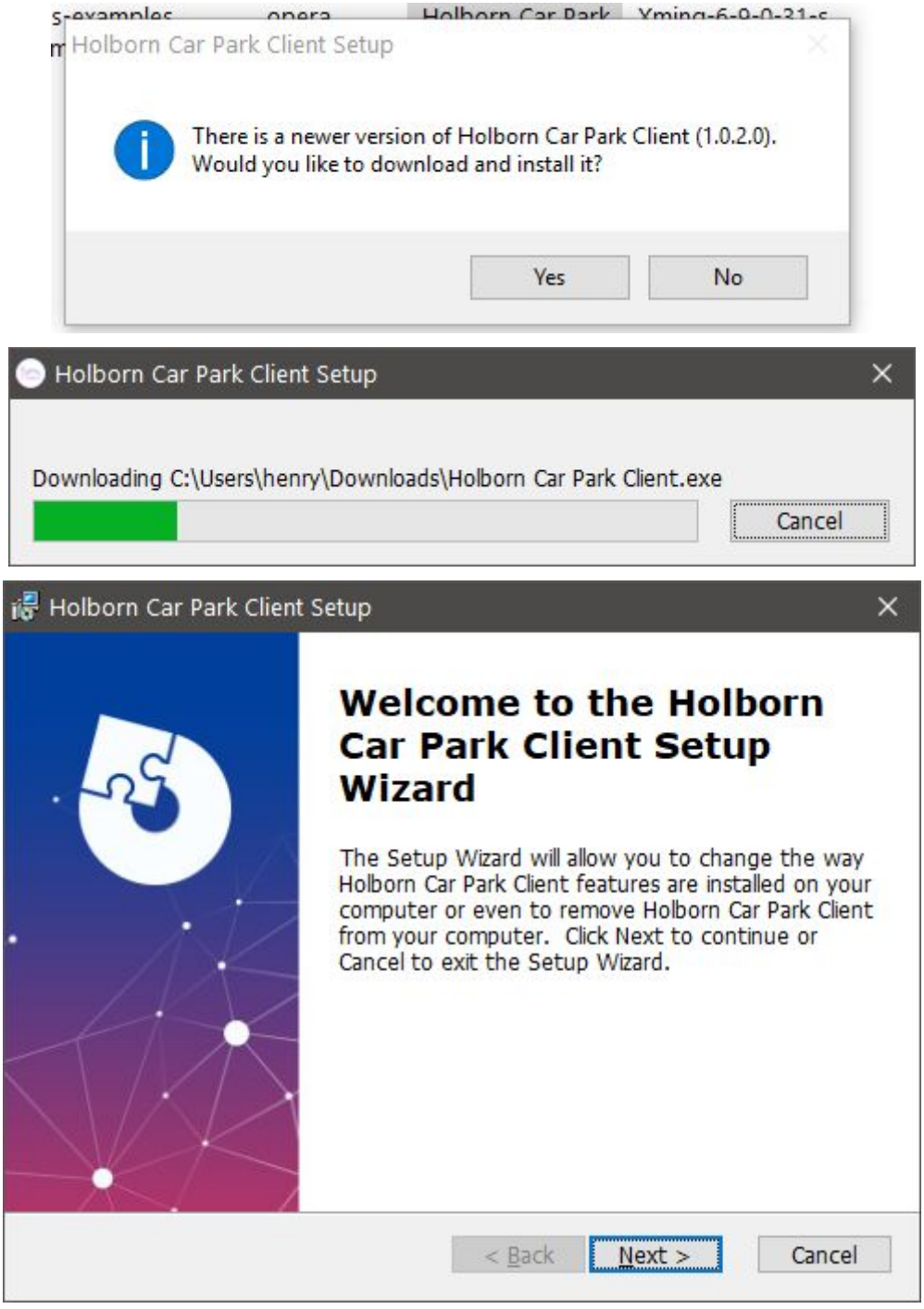
   b. Solution: make sure that the carpark_name and carpark_id are IDENTICAL to the ones stored in the webservice app

2. Application closed unexpectedly:
   a. Solution: please check the error and info logs in the installation directory. If any error or bugs have been encountered, please let us know



Updates

Holborn Car Park Client supports updates. Every time a new update is on, they will be automatically downloaded and after restart the application is up and running again



For the latest updates:https://holborncarpark.com/updates/client/download

Disclaimer: Some features may be missing or not work as intended in the pre-release candidate

// .. Teamwork:

This table shows the overall participation of all the group members broken down by each person's output and the complexity of tasks achieved.

Name	Planning	Setup	Coordination	Development	Documentation	Testing	Overall
Vlad	25%	20%	20%	30%	30%	25%	25%
Nanyo	25%	30%	40%	40%	15%	25%	25%
Henry	25%	20%	20%	15%	40%	25%	25%
Cameron	25%	30%	20%	15%	15%	25%	25%

Explanation and justifications:

Planning:

At the beginning of the project, we had several group meetups in the library during which we discussed the architecture and technology stack we wished to use and assigned + divide up tasks. As each group member attended, contribution here is even.

Setup:

Vlad: Setup maven, and configured it to work with javafx in java 11.0.2.
Nanyo: Setup the database on AWS, and created backend api with Postman.
Henry: Setup domain name and React front end.
Cameron: Setup sockets to communicate with the java barriers, ticket printing and barriers UI.

Coordination:

Vlad: Worked well as a team member
Nanyo: Divided and assigned tasks and took responsibility and leadership for the project. Managed the git repository and handled merge conflicts.
Henry: Worked well as a team member
Cameron: Worked well as a team member

Development:

(We all invested heavily in the project and spend many, many hours sacrificing sleep and sanity. But it is only fair to say who contributed the most)

Vlad: Created a very beautiful java application to simulate the pay terminal at a real life car park. Also made dark theme. (274+ commits)
Nanyo: Created a working backend and got it to work with the web a java clients. Had previous experience with this particular technology stack.(192+ commits)
Henry: Created the web manager console and login pages in React. (47+ commits)
Cameron: Created another java application to simulate barriers which communicates with the server through a socket and spawns tickets. (17+ commits)

Documentation:

Vlad: Created a user manual for the java payments terminal, and wrote comments
Nanyo: Code documents itself, some comments
Henry: Created this document and a user manual for the web console
Cameron:Wrote comments

Testing:

We all tested the code we wrote. Vlad did create some unit tests in java and we all put equal effort in the ensure the application runs with the element of least surprise.

Overall:

We all worked very hard on this project and put in a lot of effort.