COMP 2002 Final Report

Smart Scheduler for Electric Vehicle Charging

Team 9

May 12, 2021

Apollon Nakopoulos - psyan7 - 20188270

Dan Carlo Sioson - psyds9 - 20074594

Daniel Farquhar - psydf2 - 4328758

Danylo Szlachetko Blackburn - psyds10 - 20199767

Zain Rashid – psyzr2 - 20172929

Giovanni Besla - psygb2 - 20189870

Academic Supervisor

Geert De Maere

Sponsor company

ATOS

Industry Supervisors

Andy Wallace / Mike Smith

Documentation Links

Trello Board

https://trello.com/b/8CoekrPh/smart-scheduling-agile-board

Code Repository

https://projects.cs.nott.ac.uk/COMP2002/2020-2021/team9_project

Document Repository

Site

Meeting Minutes

https://uniofnottm.sharepoint.co m/:w:/r/sites/Team9groupproject/Shared%20Documents/Meeting%20Minutes.docx

Contents

Changelog

1.1 Addition of referencing to UK government policies on Electric Vehicle charging
1.2 Removal of objectives no longer being considered, regarding prioristing peak shaving/ valley
1.2 Removal of objectives, taking in to account individual driving patterns.
1.4 Analysis of related works and references

2.1 Illustration of initial requirements - Edited details to match present outlook
2.2 Specification and prioritising - Reorgansied prioritisation - Added priority categories - Removed and added some requirement points
2.3 Progression and reduction of scope - Expanded on the evolution of requirements throughout the project, reduction of scope, requirements implementation
3.1 Overall Management
• We reduced the number of scrums per week and slightly increased the meeting length to increase efficiency, as tasks started to become more challenging.
• We reduced the frequency of meetings with our sponsor, as the project became clearer, so the meetings' goal became more of a show of progress.
• We increased the frequency of meetings with our supervisor to maximise how we can benefit and learn from their expertise on the subject. 3.2 Documentation and planning
• Included more details about planning and how it changed.
• Included how and why documentation was used.
4.1 Technical and Realisation Reflections • Added CPLEX information and future aspirations
4.2 Project Management Reflections • Project Management Approach Evaluated

1 Introduction to the project

1.1 Project Background

In the effort of building a sustainable future, the transition from petrol and diesel vehicles to electric stands out as an achievable goal; supported by UK Government policy regarding a proposed ban on their sale from 2030 [1]. This approach comes as an important part of the solution to decarbonisation, but not without issues of its own. The large-scale introduction of electric vehicles (EVs) to the UK electricity networks would create new levels of demand which the networks are currently not designed to handle [2]. For the National Grid to cope, the implementation of smart charging technologies can help to manage the impact that EVs would have. Without any system in place, large investments would be required to support the additional infrastructure and power generation necessary for the use of new charging stations, resulting in further costs to the average consumer and environment just to meet these demands.

1.2 Primary aims

In view of this, the primary aim of this project is creating a smart system that schedules the charging of EVs such that the electricity networks do not become overloaded, whilst also maximising the use of renewable energy sources from the grid, ensuring that the needs of the vehicle owner are met, and minimizing the total charging cost. We consider these objectives to be appropriate, as aiming to make the most of renewable resources where possible comes in hand with the environmentally conscious proposal for new vehicles to be fully zero emission. Where on top of this, the user is also able to benefit by charging their vehicle for lower costs, with their requirements met.

To do this, we are aiming to incorporate factors of both User-managed charging (UMC) and Supplier-managed charging (SMC) into our system, allowing the user to be more involved in the scheduling process, whilst maintaining the advantages of receiving data about the grid. The user is able to input a preferred time period for their vehicle to be charged, which changes the strategy of the scheduler to allow for a more personalised schedule to be created, focusing more strictly on meeting the requirements.

1.3 Focus of requirements

One point focused on from the requirements of this project was to incorporate the simulation aspect with real time data from the National Grid. This was thought to provide the most accurate results that are representative of a real charging station, as realistic assumptions could be made for the simulation that would model a charging station, where renewable energy production is more varied. The use of APIs from the National Grid through Elexon would allow for the smart scheduling process to be applied to real data patterns regarding renewable availability and grid load, resulting in fewer assumptions needing to be made.

1.4 Related works and objectives

Furthermore, as the popularity of EVs has greatly risen in recent years and smart charging is viewed as a 'new and evolving market', there are limited commercial products that try to address the issues that come with them. This opens opportunity for innovation, as seen by the many existing studies that focus on varying aspects of the smart charging problem. These studies differ in what their objective goals/functions are, that being for example: from the perspective of an EV driver, a utilities owner of a charging station or energy supplier, where for each user they require a different aim from utilising a smart scheduling approach to charging.

Some examples of this from a driver's view might be to maximise user preference (that charging demands are fulfilled as requested), minimizing charging cost (to know the optimal times to charge their EV at lowest price), maximising profit (to be able to sell energy back to the grid, V2G) [3].

From the view of a utility owner and energy supplier, there is greater focus on maximising renewables [4], minimizing load variance, minimizing peak demand [5]. All of which are focused around balancing load on the electricity network.

1.5 V2G approach

One beneficial method to do this, is using EV batteries as storage to return excess energy to the grid (V2G). This approach is used by

sustainable energy supplier OVO energy [6], where their V2G chargers are connected to Kaluza [7] an 'intelligent software platform'. This platform provides the ability to set flexible charging schedules, with preferences for minimum charge levels, where live updates are given through a web application. The purpose being, to optimise EV charging in response to available generation, weather data and energy pricing, whilst allowing the user to save money through the additional V2G capabilities. This technology also has a large focus on maintaining a green future, through helping to stabilise the grid, reducing dependency on fossil fuels to meet demands, which closely relates to the goals for this project. The drawbacks of this however, is that most current EVs are not V2G-compatible, limiting the usage to very few vehicles, and that EV batteries using V2G degrade faster, which can incur greater costs in the future. It can be seen as an issue for scalability that this software is not widely compatible with most EV owners. Comparatively, this project aims to create a smart scheduling approach that can apply to any model of EV, whilst also able to maximise renewable resources and user needs.

1.6 Simulation approach

One further example of related work is that of EVLibSim [8], being a tool for simulating EV charging stations. This tool focuses on simulating the charging of large numbers of EVs, which can be used to test various Smart Grid techniques that help to minimise load on the grid, and to also maximise use of renewables within charging stations.

1.7 Summary

Having considered the various potential objectives and tools that exist to take on the smart charging problem, this project tries to take a slightly different approach through being more interactive. Where a user has more of an involvement, being able to request a time to charge, allocating them a charging station to be reserved for the duration required. With all of this being done through a simple mobile friendly web page, that clearly returns the scheduled time. This method we view to be important, as the closer connection with the user reinforces the idea to be conscious of EV charging, where most EV owners tend to charge their vehicles without regard for cost, energy usage and the environment. Additionally, the

accessibility of our smart charging system gives further incentive for EV owners to take the smart approach, without the requirement to take part in a V2G scheme for example, which may degrade the battery life of their vehicle.

In general, all these objectives have reason to be considered when forming the best solution to a smart scheduling system but seeing as they are only within the confines of limited trials, there is not enough recognition for the problems of greater significance. This factor gives a strong motivation for what objectives to focus on in this project, as have been mentioned previously, driving the group further to develop a strong solution.

2 Requirements and critical analysis

2.1 Illustration of initial requirements

Mapping out the requirements of the project consisted of a set of written and visualised examinations, beginning with a definition for the problem and a suggested solution in a short textual analysis. We were able to expand on our ideas by determining potential stakeholders and detailing how they should be influenced by our scheduling system. Fundamentally, our main targeted users, who are electric vehicle drivers/owners, should be informed of an optimal time to charge according to their needs whilst considering various factors such as the availability of nearby charging points. In addition, energy providers should be utilised, with this also applying to renewable electricity to ensure maximum renewable usage. Further users include manufacturers/providers of charge points/facility managers, and naturally, system administrators.

We analysed this more precisely through a user story table covering necessary procedures:

AS A	I WANT TO	SO THAT

Driver	Know when the best time is to charge my car battery	I do not run out of battery in the middle of a journey
Driver	Know where and when is the best time to charge my car battery	I can charge it off renewable energy
Driver	Know where and when is the best time to charge my car battery and current availability	I do not show up to a charging point that does not have availability
Driver	Know where the best place is to charge my car battery	I do not waste energy on the charging trip
Facility Manager	Know the peak hours and the dead hours for my charging ports	I can save energy.
Facility Manager	Know if I have enough charging ports in comparison with electric car owners in the area	I can add/remove charging ports
Admin	See Requests/messages logs	I can increase/decrease messages quota in my system
Admin	Know user's feedback to the charging times	I can improve and update the AI predicting model

Our high-level use case diagram aimed at illustrating the relationships between determined entities of the system, with which we clarify the flow of interactions and present solid and specific needs for our actors:

2.2 Specification and prioritising

We follow with a briefing text, making use of previous findings to list our initial requirements as features. We prioritise these from most basic, i.e.,

essential for the program to function minimally, to expected additions, and to lower priority extra components. The following list is the initial requirements of the system in ascending numerical order from most to least important and categorised accordingly.

Essential:
- The system must schedule the charging of electric vehicles while taking the user's driving patterns and information on the renewable mix of the National Grid into account- scheduling should also be prioritised at times to best balance load on the grid.
- There should be a (or multiple) model(s) that receive various input data to schedule optimal charging for electric vehicles.
- The system must satisfy charging requirements- charging must be complete as a priority.
- The system must be able to handle multiple scheduling requests at one time/ consider multiple users scheduling requirements.
- There must be a subsystem providing a (or multiple) user interface(s) that allows users to interact with and obtain information from the system.
- The subsystem providing the user interface must show electric vehicle owners the scheduled times to charge their vehicles.

Additions:
- Electric vehicle owners could enter information about their vehicles to be considered in the scheduling process- charging preferences/requirements (e.g., / state of charge at departure, expected time to unplug).
- The subsystem providing the user interface could be accessed through a mobile app.

Lower priority predictive aspect:

- There could be a machine learning model such as a LSTM neural network that will predict driving patterns of the electric vehicle drivers.
- The system must be able to predict electric vehicle owners' driving patterns including 1-3 days from any given day.
- The subsystem providing the user interface could show predicted heatmaps representing future usage information of charging points.
- The system could provide an API to allow other programmers to use the system from outside systems.

2.3 Progression and reduction of scope

As we made way with the project, at times our requirements had to change to fit newly assessed goals. Through going from research to development, the scope of our requirements reduced to building a solid but minimal core scheduler as a prototype. This also meant that they became more specific, expanding on the essential category to constitute smaller tasks in different components. To achieve a prototypical idea, the team would go through a number of re-evaluations in terms of the architecture of the system, the tools to be used, and in general relating to the scope of the requirements. Through progressing with the timeline and assessing the workload, it became clear that our prioritisation of developing strong and intricate foundations for the core scheduling component was correct, but with some underestimation regarding the complexity and time involved in accomplishing it. Our prototype ended up being expressed through getting our front-end web app up and running, which would act as a placeholder to help us better judge the requirements which would be built into it. In short, various reconsiderations of our system brought us to the conclusion that we had to let go of our initial lower priority plans (such as our predictive aspect or mobile application objectives) to fulfil our essential category requirements to a satisfactory degree within the allocated timeline.

3 Project Management & Progress

3.1 Overall management

Starting this project, we, as a team thought about the management ideologies we wanted to implement going further and have established them according to what would be most suitable for the project. However, as we dived deeper into the project, we started to make micro adjustments to better increase the efficiency of our work.

For instance, we started off relying heavily on an agile approach, including scrums. During the first part of the project, we used to run multiple short meetings every week for everyone to recap what they have achieved, if they faced any challenges, and what to do next. We, also, had a meeting with our supervisor and our sponsor every 2 weeks. The structure of

the scrum meetings was straightforward and concise where we start by everyone sharing what they have accomplished and what we need to get done for the next time frame. Whereas for the supervisor and sponsor meeting, we would discuss what the overall final project would look like, and how it would function.

Going forward, we followed the same main ideologies but implemented minor changes. For example, we still followed an agile approach of constantly updating and being flexible with the project structure. However, we reduced the number of scrums to one per week, increased our supervisor meetings to once per week, and reduced our meetings with the sponsor to once per month. While our scrum meetings now lasted longer than before, maybe not even being a scrum anymore, they still had the same structure, as it has proven efficient. Those changes were made for multiple reasons. One of them being the fact that with the progression of the team withing the project, everyone now had larger and more challenging tasks to do, as a result, regular small meetings became more redundant. For the same reason of tasks getting harder, we began to meet up with our supervisor once per week, to try and maximise what we learn from them and their expertise in the field. Lastly, meetings with our sponsor were reduced since the meeting goals themselves has changed. To elaborate, in the beginning, we were still agreeing on the final structure and state of the project, so it was beneficial to share what we are reaching and finding frequently. While on the other hand, towards the end of the project, everything was getting clearer so the goal of the meetings with our sponsor was merely to show progress.

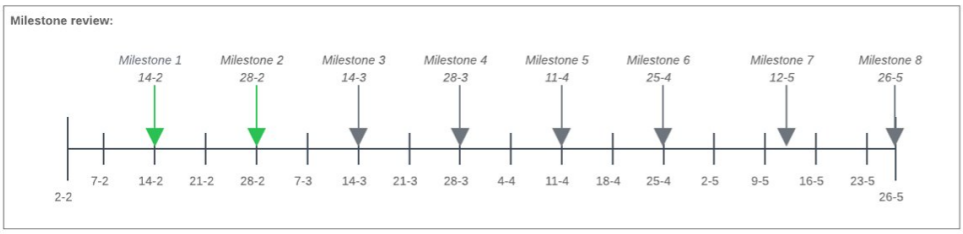
3.2 Documentation and Planning

In order to put down an effective plan to follow for the project, we started by agreeing on what the final project would look like when complete and when it needs to be done by. Then, we started working backwards in time thinking about how much time each component or level would take. The main split up was as follows; producing a functioning prototype by Christmas time to make sure all the different components are working together successfully. For this initial prototype, we used a primitive FCFS for the scheduling. This, surely, would not be the final working scheduler, but it was developed nevertheless to as a placeholder for the scheduling

component within the system, so that when the more complex scheduler is ready, we can simply swap them together.

After reaching that milestone, we had a clearer idea of what the final product would look like, so the following diagram with deadlines and milestones was produced.

Prospective Timeline



[1][2] - Front end, real time grid data, database connectivity
[3] - Extending simulation
[4] - GUI to manipulate simulation parameters,
[5][6] - Revised scheduling algorithm (To have something running with output for next meeting)
[6][7] LSTM predictive model
[7] - Testing and Report
[8] - Final Presentation

Further down the line, some minor changes were made since we realised that some components needed more work than we expected, so we had to redistribute some of the workload amongst us and let go of some minor components of the system.

Since we were tackling a large project, and different team members were responsible of different aspects and components, it was hard, yet crucial to make everyone have some basic level of understanding of all the components of the system and how they work. To tackle that, everyone was producing documentation files for their all their work in an extensive, yet concise manner. Not only that, but we also kept track of all the progress done in the meetings by documenting everything in our Meeting Minutes document.

3.3 Project Management and Tools

To communicate we used a mixture of Microsoft Teams and Slack. Slack would be used as our main source of communication allowing us to communicate and plan our workload in an organised and professional manner. We had multiple channels for the different sections making up our project. This allowed us to easily share work if need be and have multiple chats running simultaneously for when multiple teammates were working on different tasks at the same time.

While for our formal meetings we would use Teams allowing us to make use of features such as screen sharing to update and show our client progress made within our project. In Teams we implemented a SharePoint where all members could freely upload and edit files. This was important as it allowed for the possibility for multiple members to simultaneously edit the same file, giving the group more freedom when brainstorming ideas as we could all input our own thoughts to a joint file.

Trello was used as our planner letting us set and assign roles/tasks to each group member. We used an agile approach for our board creating multiple cards which related to a certain part of development such as a backlog, tasks in progress or issue tab; to name a few.

Outlook would be used to set meetings in the group and between our supervisors because of its built-in calendar feature. Letting us organise meeting in a more professional manner.

3.4 Team Skills

With a project of this scope our team had to make full use of our strengths to most efficiently breakdown the tasks. Each member was assigned a component of the project to specialise on:

- Zain – Database Management / Web Server – Previous MySQL and web app development experience
- AJ – Front End Django Application – Previous web app development experience

- Dan S – Scheduler – Most experienced with Python due to his knowledge of AI heuristics algorithms through him developing a game
- Danylo – Simulation - Previous Application Development Experience in C#
- Daniel F – Collection and manipulation of data
- Giovanni – Communication between components / extending and refactoring the simulation

4 Preliminary Reflection

4.1 Technical and Realisation Reflection

The biggest problem we ran into as a group was quite early on. As it had been suggested to us in the project brief, we had planned to use Unity Simulation to complete much of the project, due to the ability to create a 3D environment and include an interesting visual element to our simulation. On top of this, we would be able to generate data and run multiple simulations using the cloud.

However, after researching it further, it appeared that in order to create the environment for the simulation, we would require Unity Pro, which is an expensive subscription. We also found that there was limited community support for Unity Simulation, as it is relatively new, and we felt that due to the complex nature of the task and the fact that a lot of the themes in the project were new to us, we wanted to use software that had a good community support network to lean on to help with our learning. As a result, we looked for alternatives and found SimPy.

While SimPy did not offer the same visual element to the simulation, the simulation of the processes themselves can still be done, and our team had more python experience than C#, the language we would have used with Unity Simulation. Also, creating the simulation in SimPy took less time than in Unity Simulation. We used this time to create a more technical scheduler using IBM CPLEX.

IBM CPLEX had a steep learning curve, but we were able to get to grips with it through meetings with our group supervisor. Ultimately, CPLEX allowed us to achieve results we would never have been able to with python alone. Despite the initial challenge, the use of a Python API for CPLEX did make things easier to pick up, as it is generally used with Java.

Overall, on the technical side, the group worked through issues really well, either using online resources to figure out the solution, or finding a suitable alternative way of completing the task. The breaking down of the project early on really helped in this regard, as we were able to assign aspects of the project to team members that were most suited to them with their strengths.

In the future, a big idea we had but did not have time to implement was an interactive map for the user. This would show the locations of different charging points along with information about the user's vehicle that they had entered. We would have loved to have included this, however we decided other elements were more important to the thoughts behind the project regarding energy efficiency and helping the environment, where this feature would have mainly been to improve the experience of the user.

4.2 Management Reflection

Looking back on the start of the year, it is likely we did not fully anticipate just how challenging working as a group would be. With this project, almost every aspect of it was already brand new to us, the topic, the software and now due to the pandemic, working as a group exclusively online.

As a result, it was quite hard to pin down all the details of how we would go about completing the project early on and required a lot of research. We chose to take an Agile approach, which worked well to an extent, however looking back, we would likely have benefitted from having more certainty in our approach, so that we would have avoided spending time on certain resources that had very little to no effect on the project in the end.

The Agile approach did mean that all of our elements were very versatile and easy to use with whatever frameworks we decided to eventually use, which became vitally important when combining so many moving parts.

5 Conclusion

Along this project our team has learnt a handful of valuable skills, we have displayed resourcefulness having to piece together information from various sources. We persevered through the current unfortunate circumstances managing to keep punctuality high and workflow constant. The team thought ahead and were quick to adapt and overcome issues, implementing plans and roadmaps for how to tackle and breakdown the project. Working together to split work equally and fairly, the team made sure all work was completed on time even in circumstances where members may have been unable to work.

We are pleased that we were able to produce a working scheduler, with front end input that is all linked to a database. In a project that required at lot of research and was largely new to us it is very fulfilling to look back on how far we've come since the beginning. Although we were not able to implement everything we might have wished to, we are confident that with more time we would have been able to add these elements with our new knowledge as mentioned earlier in the report.

6 Bibliography

[1] O. f. Z. E. V. Department for Transport, »Consulting on ending the sale of new petrol, diesel and hybrid cars and vans,« March 2020.

[2] D. f. Transport, »Electric Vehicle Smart Charging,« HM Government, July 2019.

[3] E. S. a. M. A. El-Sharkawi, »Optimal Charging Strategies for Unidirectional Vehicle-to-Grid,« IEEE Transactions on Smart Grid, March 2011.

[4] E. L. D. R. C. S. a. B. W. A. Brooks, »Demand Dispatch,« IEEE Power and Energy Magazine, May-June 2010.

[5] W. S. a. K. C. F. Rassaei, »Demand Response for Residential Electric Vehicles With Random Usage Patterns in Smart Grids,« IEEE Transactions on Sustainable Energy, Oct. 2015.

[6] C. Topping, »Vehicle-to-Grid (V2G) explained: What it is and how it works,« OVO energy , 16 February 2021. [Online]. Available: https://www.ovoenergy.com/guides/electric-cars/vehicle-to-grid-technology.html.

[7] »SECURELY CONNECTING ALL DEVICES TO AN INTELLIGENT ZERO CARBON GRID,« Kaluza , [Online]. Available: https://www.kaluza.com/flexibility-platform/.

[8] S. N. S. E. Rigas, EVLibSim: A Tool for the Simulation of Electric Vehicles' Charging Stations Using the EVLib Library, 2018.

Software Manual

Glossary

DOcplex – IBM's optimisation problem solver API for Python; it stands for (IBM) Decision Optimisation CPLEX (Modelling).

LP (Linear Programming) – An exact mathematical optimisation method using linear functions to represent problem elements.

Decision variable – The name given to variables with values that are dynamically altered to obtain the optimal solution in a mathematical programming problem.

MILP (Mixed-Integer Linear Programming) – A type of linear programming technique involving both discrete and non-discrete decision variables.

LP Scheduler – Stands for Linear Programming Scheduler and refers to the scheduling component of the software.

Time slot – A discrete representation of a period of time; another term for time interval.

Time slot allocation – The process of assigning time slots to each electric vehicle during which they can be allocated charges in.

Time interval allocation strategy – A heuristic for limiting and relaxing the choice of periods for when each vehicle can get allocated charges.

Time Slot Optimiser – The software component that performs the MILP modelling and solving process, determining which time intervals to charge electric vehicles in and how much these charges should be.

Charge allocation – The process of determining how much energy (measured in kWh) is put into electric vehicles' batteries. Also referred to as 'time slot optimisation'.

Scheduling window – A sequence of time slots used in the MILP model by the Time Slot Optimiser.

Production – A term that refers to the generation of electricity in the context of the MILP model.

Consumption – A term that refers to the use of generated electricity in the context of the MILP model.

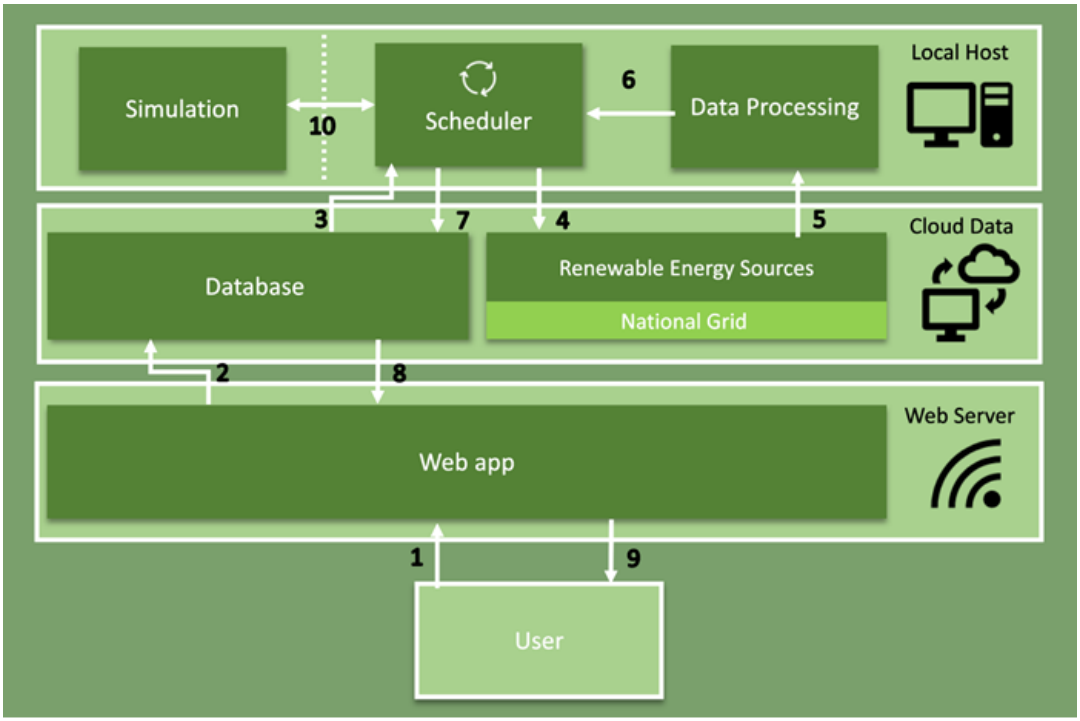
Auxiliary vehicle – A 'stand-in' vehicle representing the tweakable charge decision variables of vehicles that have already been scheduled.

kW – The rate of 1000 watts per second, or 1000 joules per second.

kWh – Equivalent to 3600 kilojoules—the amount of energy transferred in 1 kW in 1 hour.

Introduction to the problem

The smart scheduling problem requires a complex algorithm to factor in current fossil fuel – renewable mix percentages, energy consumption, environmental factors as well as the dynamic schedules of the electric vehicle drivers. With this problem the ideal goal would be to have a solution that took into consideration all these factors and used a self-learning AI to determine trends and predict the best times for users to charge based on their frequent schedules.



Overview of System Architecture

The system is a multi-tiered program made up of multiple components working in unison.

1. Firstly, the user will interact with the web application. Here they will create an account to store their personal details as well as charging preferences. They will then send a request to schedule through the front end.

2. Next, the user's details are validated in the back end to make sure they are compatible with scheduler. If so, a request is successfully sent and saved to the database under that user's account.

3. The scheduler constantly runs, repeatedly checking the database for new requests for users wanting to charge. Once the user request is received, the scheduler will import their details from the database and format it for usage with system.

4. After storing the user's inputs for use it'll request current API data on renewable sources and energy consumption in the area.

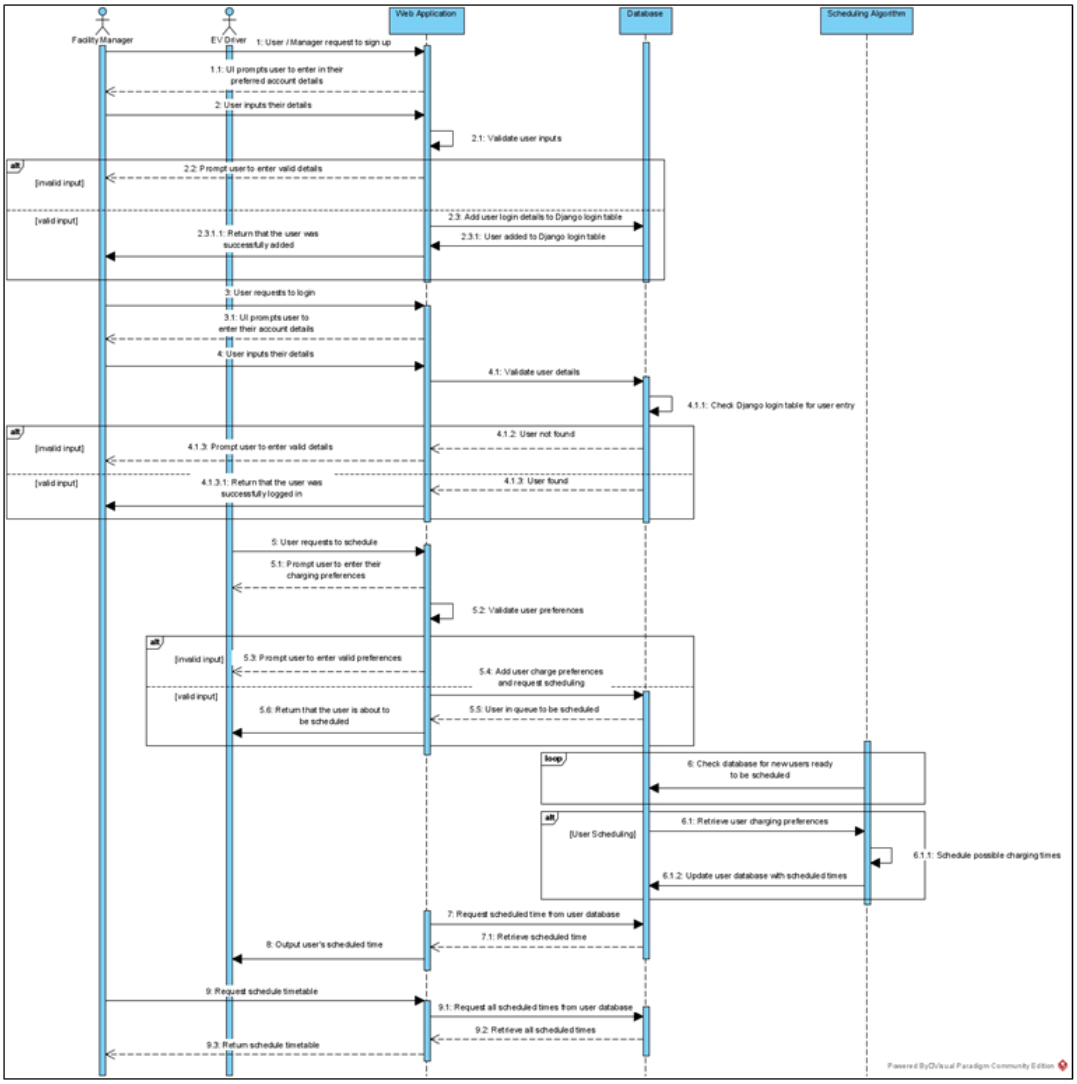
5. Again this new imported data has to be processed before it is handled by the scheduler.

6. Once processed the data is fed back to the scheduler and combined with the user inputs is used to schedule the user's new charging slot. The scheduler will generate a time based on the API data, user preferences and previously scheduled vehicles, to produce a slot around current energy figures.

7. After the vehicle has been scheduled or is unable to due to low energy supply and /or other factors; the scheduler output will be exported to the user database where it is again stored under the user's account.

8. While the scheduler is generating the new slot, the web app is constantly checking for an update in the database to know when the scheduling has been complete. The web app will showcase a loading screen guiding the user through the scheduling process. Once the scheduler output has been saved to the database, the web app imports this data and formats it for use.

9. Now the web app displays the allocated time slot to the user along with the location of the charging station through a Google Maps API.



Sequence Diagram representing the interactions the system will encounter when either a user or manager uses the web application.

10. The final component is the simulation which is run at regular intervals to test the scheduler is working as intended. However, isn't vital to the main process of scheduling a vehicle.

System Setup

Database

The only requirement to access the cloud-based database is that the host user has the **RDBMS** running.

To access **AWS** sign in as a temporary user with these **IAM** credentials:

- **Account ID: 449420458914**
- **IAM username: Team9**
- **Password: Password!**

With these details you can manually turn the database on and off as well as create new web app instances.

To view the database download the **MySQL Community Server** as well as the **MySQL Workbench.**

- No settings have to be tweaked in the installation.
- Once installed enter the **Workbench** and go to **Connect to Database.**
- From here you'll enter :

Hostname: schedulerdb.cv1vtvg9bql2.eu-west-2.rds.amazonaws.com

Username: admin
Password: password **Port**: 3306

- Now you should have full access to the database system

Showcases MySQL Workbench UI

Web application

To setup the web app the user must have access to the database as well as an **AWS account** and **Python 3.8**

- First download the **Django** code off the **Git repository**
- Next, install the dependencies from the **requirements.txt** file in the project.
- Now, if the database is running the site should work locally.

However, to upload it to AWS requires more steps:

- Make sure in the Django project in **settings.py**, **DEBUG** is set to **FALSE** when trying to deploy the site to the web. As otherwise a data breach could occur.
- Follow this tutorial: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html.
- Enter the **command prompt** inside your Django project's root directory.

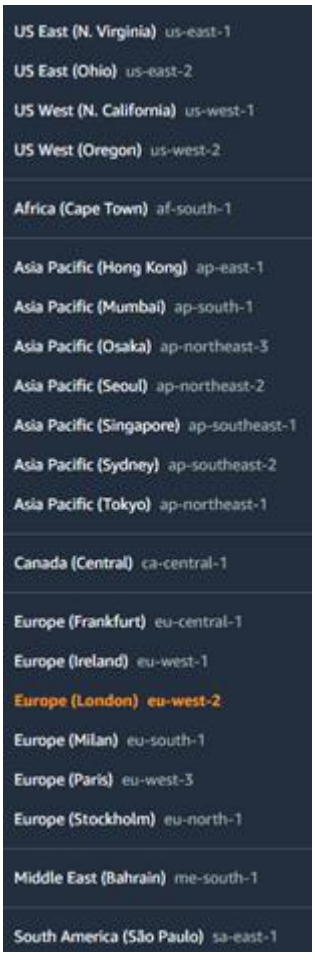
- Initialise your elastic beanstalk application using this command:

eb init -r eu-west-2 -p python-3.8 scheduler-application

Where:

eu-west-2 = what server you want to use to host your application.
3.8 = the Python version you'd like to deploy your code at.
Scheduler-application = whatever name you want to give your application

After entering that command you'll be prompted with a question, say **y** and select the keypair already created:



Select a server region from this list.

Do you want to set up SSH for your instances?

(y/n): y

Select a keypair.

1) **my-keypair**

2) **[Create new KeyPair]**

Now enter in the command line:
eb create ev-scheduler

Where: ev-scheduler = the domain name you'd like for your site

- The instance should now be deployed, however, first you have to add the new domain URL to your list of **ALLOWED HOSTS** inside your **settings.py** in your **Django** project.

- Since the application is already deployed you can now actively update this instance by typing:

eb deploy

- This will update the application with any changes you made to the code, in the case adding the new domain name as an accepted host.

- Once deployment is complete, the site should work!

Scheduler

For the scheduler to work you must have **CPLEX Optimization Studio** installed on your machine as well as **Python 3.8**:

https://www.ibm.com/docs/en/icos/20.1.0

With this project we require the **full version** of **CPLEX** not only the **Community Version.** This can be gotten through the student/ faculty partnership:

https://content-eu-7.content-cms.com/b73a5759-c6a6-4033-ab6b-d9d4f9a6d65b/dxsites/151914d1-03d2-48fe-97d9-d21166848e65/academic/home

- Once CPLEX is installed navigate to this directory on your machine:

C:\Program Files\IBM\ILOG\CPLEX_Studio201\cplex\python\3.8\x64_win64

- There should be a file called **setup.py,** this file must be run in the command line at that directory:

python setup.py install

- After running that command CPLEX should now be installed.
- Now, go to the **Git repository** and download the **scheduler project.**
- Install all the dependencies off the **requirements.txt** file.
- The scheduler should now be set up!

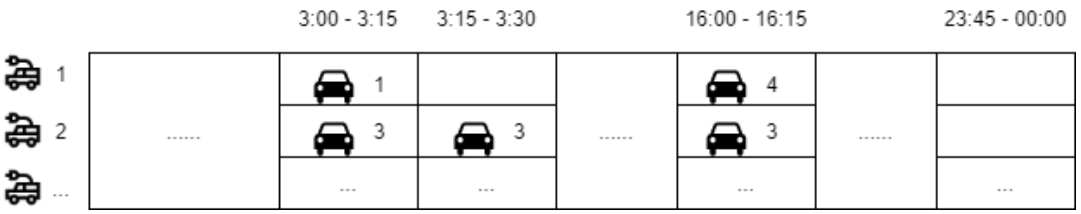
Run Project

To now run the project as intended you must have:

- A local host running the scheduler on their machine.

- The cloud database running.
- The application instance served to Elastic Beanstalk and live.

Scheduling



Visual representation of time intervals.

The main concept behind the scheduling process is to partition time into fixed-length intervals to reduce the computational requirements and provide reasonable approximations of a schedule. The software solution discretises time into 15-minute intervals, resulting in 96 time intervals in a single day. These time intervals function as time slots in a timetable where electric vehicles are assigned to, provided that the right conditions are met. Each assignment is associated with a charge value representing the raw kilowatt hour (kWh) transferred into each vehicle's battery.

The scheduler makes two (related) decisions—the charging period an electric vehicle is assigned to (that is, the time that it starts charging and the time it finishes charging), and how much electricity to put into the electric vehicle's battery.

The scheduling process makes use of various aggregated data from the electricity grid around a specific area in decision making. Due to the simulation aspect of the solution and the limitations of the BM Reports API, some information is simply generated, but treated as results of API calls, and all are processed to downscale the values appropriately to fit the simulation environment.

For each time interval, the following electricity-related information are used in the scheduler's decision-making process:

- Total generated electricity from traditional sources
- Total generated electricity from renewable sources
- Total electricity demand
- Maximum capacity of the electric grid

- Price tariffs for charging

Using the data about each time interval, the scheduler builds and solves a mathematical model to obtain the optimal allocation of charges to each vehicle.

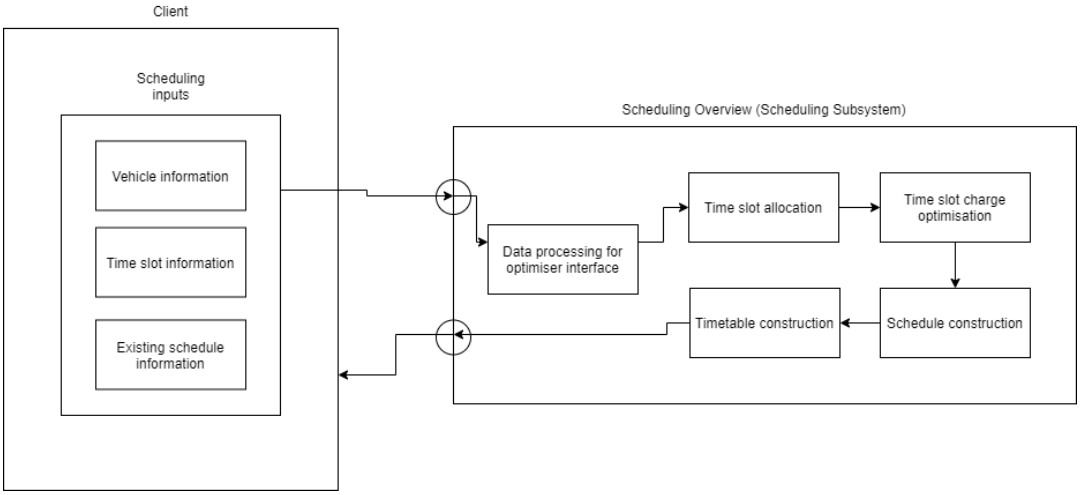
Decision Optimization CPLEX Modelling for Python (DOcplex)

DOcplex is an optimisation library for Python that makes use of the commercially available CPLEX mathematical solver. It has two components: one for mathematical programming (linear, quadratic, and so on) and one for constraints programming.[1]

DOcplex's mathematical programming component (DOcplex.MP) is used to model and solve the problem of determining how much charge is put into each vehicle in each time slot.

The use of DOcplex requires installation of IBM ILOG CPLEX Optimization Studio[2]. Alternatively, DOcplex may also be used through IBM Watson Studio Cloud[3], though because the scheduling process had been written with a local machine in mind, building and solving the model on the cloud may require significantly more additional steps to set up.

Overview of Scheduling



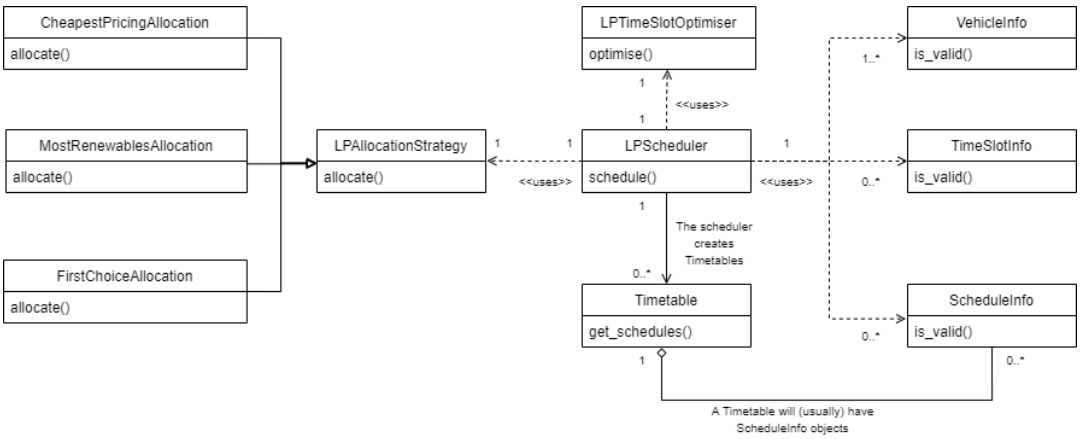
An overview of the scheduling process.

The scheduling process is requested by the daemon through the scheduling interface, called the LP Scheduler (Linear Programming Scheduler). This is done by wrapping related information into objects and passing these into the scheduler component. There are three categories of information that the scheduler takes in:

- Vehicle information – information about each individual vehicle, e.g., battery capacity.

- Time slot information – information about each time interval, e.g., total traditional production.
- Existing schedule information – information about vehicles that have already been charged in each time interval, e.g., how much charge is allocated to a particular vehicle in a time slot.

Scheduling Components



A simplified class diagram for the scheduling process with all main methods.

The LP Scheduler serves as the main component that ties every subcomponent of the scheduling process together. As part of its processing in preparation for the time slot optimisation, it automatically discretises times such that each time maps to the nearest 15-minute time interval.

LP Scheduler

The scheduler interface that ties everything together. All clients outside the subsystem must use this component and follow its interface to perform scheduling.

Vehicle Info

This parameter class can be instantiated multiple times as necessary to represent each vehicle to be scheduled. It encapsulates the following information:

- Vehicle ID.
- Arrival and departure date times.
- State of charge at arrival.
- State of charge demand.

- Battery capacity.
- Charger ID.

Time Slot Info

This parameter class can be instantiated multiple times as necessary to represent each vehicle to be scheduled. It encapsulates the following information:

- Date and time representing the time interval.
- Maximum load capacity for the time interval.
- List of consumption values.
- List of production values.
- List of renewable production values.
- List of available chargers during the time interval.
- Existing charging decisions (a list of Schedule Info parameter objects).
- Price tariff associated with the interval.

Schedule Info

This parameter class can be instantiated multiple times as necessary. Each Schedule Info represents how much charge is put into a time interval (e.g., in time interval 0, there could be two Schedule Info objects, representing two cars charging in that time interval).

This class encapsulates the following information:

- Vehicle ID.
- Charge (how much electricity is put into the vehicle, in kWh).
- Charger ID (the ID of the charger the vehicle uses).
- The scheduled time of arrival—the date time that represents the beginning of the 15-minute time interval when the vehicle is supposed to start charging.

- The scheduled time of departure—the date time that represents the beginning of the 15-minute time interval when the vehicle is supposed to leave the charging point.

LP Allocation Strategy

	$t_1 \vdash t_{49}$	t_{50}	t_{51}	t_{52}	t_{53}	$t_{54} - t_{95}$	t_{96}
v_1	...0...	1	1	1	1	...0...	0
v_2	...0...	1	1	1	1	...0...	0
v_3	...0...	1	1	1	1	...0...	0

A visual representation of a time slot allocation matrix. A value of 1 signifies that a vehicle can be charged during the relevant time slot, and a value of 0 signifies that the vehicle cannot.

There are three allocation strategies currently built-in for the scheduler to use. These are simple strategies that prioritise different factors, affecting the way time slots are allocated to each vehicle.

The first strategy is First Choice Allocation, which is the most basic allocation strategy that simply uses the preferred charging period directly.

The other two strategies, Most Renewables Allocation and Cheapest Pricing Allocation are similar strategies that prioritise time slots with more renewables and time slots with cheap price tariffs respectively. These have an offset setting that is used to move vehicles' possible assignment of sequence of time slots backward or forward within the scheduling window. By default, this offset is 10 time slots or 2.5 hours (assuming that there are 15-minute intervals).

Timetable

This class is the result of the scheduling process and represents schedules determined by the scheduler. The schedules are represented using a 2-

dimensional list (a timetable) of Schedule Info objects with an offset as the scheduling window provided. These can be obtained in an easier to process dictionary by using the method in the class.

Timetable	
Time Interval/Slot	**Schedule Info (Allocated Charges)**
...	...
60th time interval / 900th min (15:00)	EV 63 with charge 4.0, arriving at 14:30 and finishing charge by 15:30
	EV 20 with charge 1.0, arriving at 14:00 and finishing charge by 15:15
61st time interval / 915th min (15:15)	EV 63 with charge 6.0, arriving at 14:30 and finishing charge by 15:30
...	...

A visual representation of a timetable showing 2 electric vehicles' schedules.

LP Time Slot Optimiser

This class handles the allocation of charges, using the time slot allocation matrix as a guide in limiting the time slots in the scheduling window where each vehicle can be charged. It returns a charge allocation matrix.

Scheduling Pipeline

Any client to the scheduler is expected to supply implicitly, in order to request schedules, a scheduling window along with the vehicle information, time slot information and any other existing vehicle schedules. This scheduling window is determined by the time slots that the client supplies to the scheduler.

The scheduler then processes the data, discretising times and converting them into formats that the optimiser can process. It uses an allocation strategy to then acquire a time slot allocation matrix, which it then passes on to the optimiser.

The optimiser builds and solves the MILP model using IBM's DOcplex API, and returns a charge allocation matrix back to the main scheduler class.

The last task in the scheduling pipeline involves processing the charge matrix and creating a Timetable to pass back to the client.

MILP Formulation

The core of the scheduling lies within the Time Slot Optimiser, which performs the mathematical modelling and determines the optimal allocation of charges.

The scheduler passes on representations of all required data as well as the allocation of charges (which it will have obtained using one of the allocation strategies) to the optimiser; the optimiser subsequently builds a model, solves it, and serves back the result to the scheduler client.

	$t_1 - t_{49}$	t_{50}	t_{51}	t_{52}	t_{53}	$t_{54} - t_{95}$	t_{96}
v_1	…0…	3.0	2.0	3.0	3.0	…0…	0
v_2	…0…	3.0	2.0	3.0	1.0	…0…	0
v_3	…0…	2.0	1.0	2.0	1.0	…0…	0

A visual representation of a charge matrix, where the values represent the amount of electricity to be put into the vehicle by the end of each time interval.

A charge matrix (as shown above) represents the time slots in the scheduling window and the charges allocated for each vehicle in each time slot. This matrix is constructed and returned by the optimise method.

Decision Variables

The main concept for the time slot optimisation makes use of the idea of production and consumption—that is, the generated electricity and the electricity demand—per time interval. The consumption is represented as two decision variables: one representing the consumption of energy generated from traditional sources and the other representing the consumption of renewable energy. Other decision variables are used to quantify the optimisation elements—all decision variables are as follows:

- Vehicle charge ($Ch_{v,t}$) – Represents charge allocated for vehicle v at time slot t.
    - The charge must be an integer between 0 and the maximum charge $Ch_v^{max} = \left\lfloor \frac{CR_v}{cpi} \right\rfloor$ where CR_v represents the charge rate of the charger used by vehicle v per interval, and cpi is the charge portion per interval, calculated as $\frac{60}{l}$ where l is

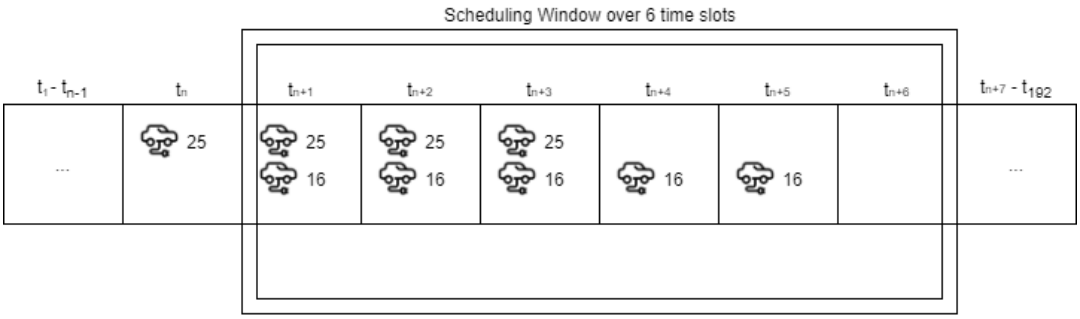
the interval length in minutes. For 15-minute intervals, cpi=4.

- Charge deviations ($D_{v,t}$) – Represents deviations for vehicle v such that $\sum_{t=1}^{t_{final}} Ch_{v,t}$ is equivalent to the charge amount needed to fulfil dv, representing the vehicle's demand, where t_{final} is the final time slot.

- Charge of vehicles that have already been scheduled ($Ch'_{v',t}$)– Represents charge allocated for the already scheduled vehicle v′ at time slot t in the scheduling window.
- Sink (S_t) – Represents the amount of production not 'consumed' in time slot t. This is an assumption that the electricity that is not used is sent somewhere for use, such as by storing it somewhere.
- Use of traditional production (CT_t) – Represents the 'consumed' amount of electricity generated from traditional sources (e.g., coal-fired power stations) at time slot t.
- Use of renewable production (CR_t) – Represents the 'consumed' amount of electricity generated from renewable sources (e.g., photovoltaic power stations) at time slot t.

Tweaking Scheduled Vehicles' Charge Allocations

In order to tweak the schedules of vehicles that have already been scheduled but still preserve their scheduled times, the optimiser adds these vehicles' charges as decision variables in the model ($C'_{v,t}$) whilst putting in constraints such that their total charge by the end of their charging time is the same as the total charge they have already been allocated. this should make sure that the optimiser can still alter these charges whilst not changing the users' charging times.



A visual representation of the scheduling window; V16 *is a vehicle that is being scheduled whilst* V25 *is a vehicle that had already been scheduled. In this case, the optimiser takes the charge values of* V25 *from* tn+1 *to* tn+3 *and adds them to the model as the decision variables of an auxiliary vehicle. Any vehicle charge allocation already in the overall timetable is added to the model as long as these lie within the scheduling window.*

Constraints

In solving the problem formulation, a number of constraints had been identified that optimisation process is subject to (see Decision Variables section above for variable names).

Let V and V′ represent the set of identifiers of vehicles' to be scheduled and the set of identifiers of already scheduled vehicles.

Firstly, to quantify the use of electricity, the consumption and the charges must be equal to the use of renewable and traditional energy. For all time slots t:

$$CT_t + CR_t = \sum_{v \in V} Ch_{v,t} + \sum_{v' \in V'} Ch'_{v',t} + C_t$$

Quantifying the charge deviations is simple—for any real number x and y, if $x \leq y$, then there exists a positive (or 0) real number k such that $x + k = y$. In this case, k is the deviation. For all $v \in V$:

$$d_v = \sum_{t=1}^{t_{final}} Ch_{v,t} + \sum_{t=1}^{t_{final}} D_{v,t}$$

The production and the consumption must be equal (energy cannot be destroyed). For all time slots t:

$$CT_t + CR_t = PT_t + PR_t + S_t$$

Where PT_t and PR_t represent the production of electricity from traditional sources and renewable sources at time slot t respectively.

The auxiliary vehicles' 'demand' (the total charge values they had already been allocated) must be equal to its total charge. For all $v' \in V'$:

$$d_{v'} = \sum_{t=1}^{t_{final}} Ch'_{v',t}$$

To use the time slot allocation matrix, constraints must also be set to prevent charges from being allocated in time slots that are not within the scheduling window or those that do not follow the allocation strategy used. For all $i \in V, i' \in V'$ and all time slots t, if row i and column j of the time slot allocation matrix is 0:

$$Ch_{i,t} = 0 \text{ and } Ch'_{i',t} = 0$$

The maximum grid capacity G_{max} must not be exceeded, so the total consumption must be restricted, as well:

$$G_{max} = \sum_{t=1}^{t_{final}} (CT_t + CR_t)$$

Objective Functions

These functions serve as the goals that the optimisation process tries to minimise and maximise. There are three objective functions in total, as listed below (in decreasing order of priority):

To make sure that the allocated charges meet each vehicle's demand as much as possible, the first objective function should be to minimise the charge deviations.

$$Minimise \sum_{v \in V} \sum_{t=1}^{t_{final}} D_{v,t}$$

To maximise the use of renewables, like the deviations, the optimiser maximises a linear expression that sums up the decision variables that quantify the use of renewables in each time slot:

$$Maximise \sum_{t=1}^{t_{final}} CR_t$$

Similar to the other objective functions, the optimiser creates a linear expression that represents the total charging cost. The pricing tariffs can simply be used as the decision variable coefficients to calculate the sum:

$$Minimise \sum_{v \in V} \sum_{t=1}^{t_{final}} Ch_{v,t} \, Pr_t$$

where Prt represents the tariff associated with time slot t (price per kWh).

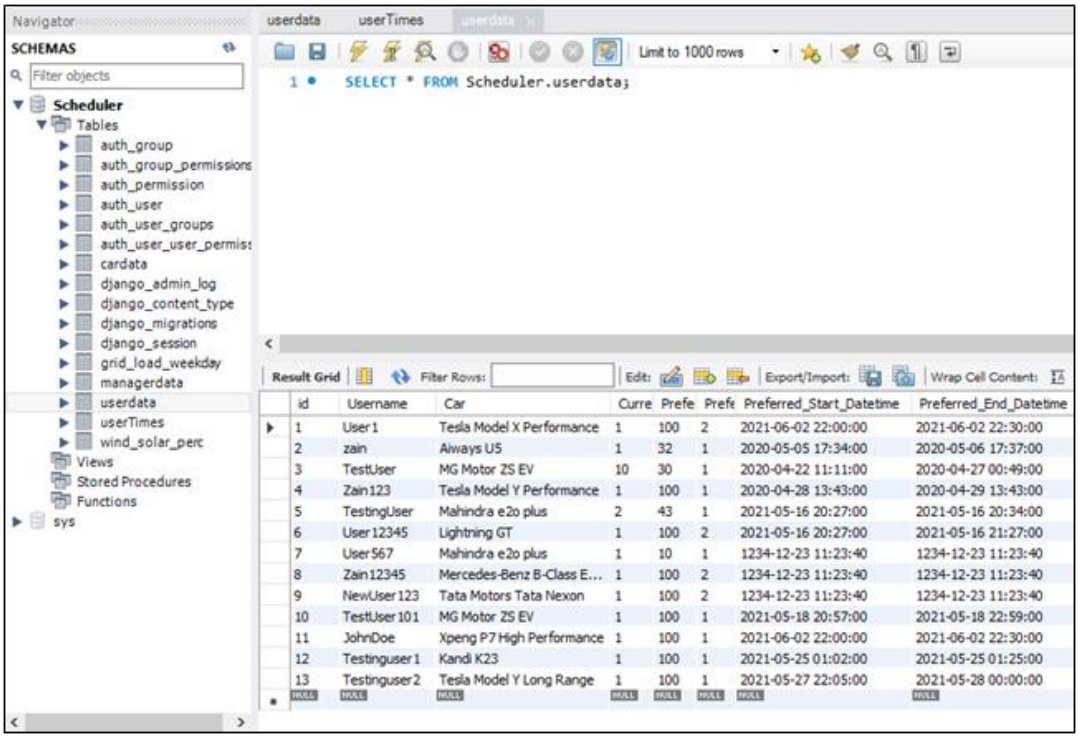
Scheduler Structure

There are only two relevant files required to run the scheduler: **simulation/scheduler/run_lp_scheduler.py** and **simulation/scheduler/lp_scheduler.py**. The former acts as a daemon of sorts that runs in the background, observing for a change in the database which signifies that a user is scheduling. The latter contains all components of the scheduler.

Data / Hosting Management

Type of Database System

In this project our data is primarily stored in an online cloud database. The system we opted for was a **RDBMS**. The specific RDBMS we chose was **MySQL**, even though **Django** is stated to be better integrated with a **PostgreSQL** Database. This is because PostgreSQL is an object-relational database meaning it contains more complex features such as table inheritance and function overloading. However, we decided ease of use and prior experience triumphs the upsides of PostgreSQL. Our group has used MySQL in past projects and thus find it easier to manage and work with. On top of this MySQL has an official Workbench which saves time as we visually see the data being handled and stored instead of through queries alone, helping with debugging and efficiency.



MySQL Workbench GUI – showcases database structure as well as table formatting.

Database Host

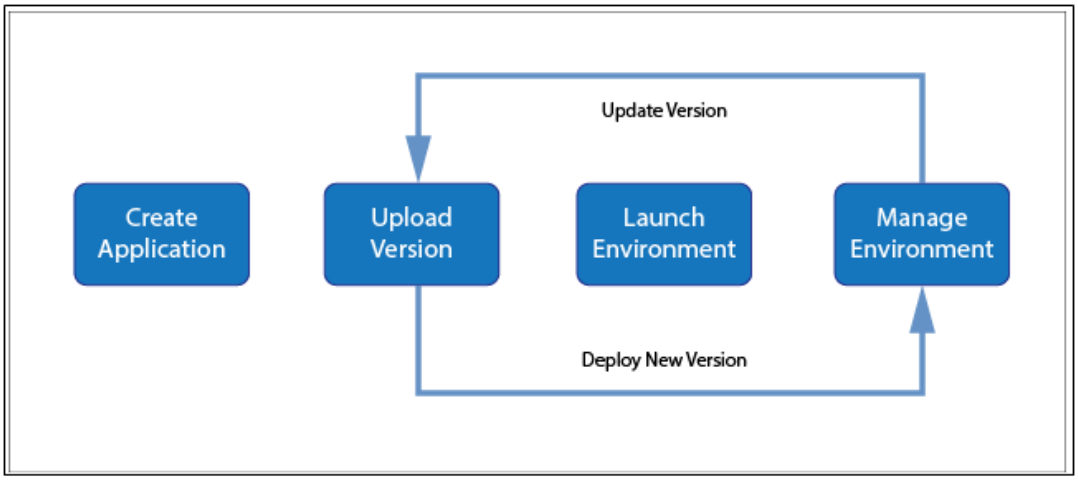
After deciding on using MySQL our group looked at options for hosting this database on a cloud platform. We didn't want a local database due to a cloud database being more portable and accessible. This would mean every member in the group had access to the same database to allow for easier managing of the system. In addition to this, we considered the end goal of our project and if we wanted a product which could be reused in a real-world setting, we had to make considerations on how easily we could transfer our code base and dependencies to other systems. Overall, leading to the choice of a cloud hosted database.

The two choices we settled with was either **Amazon Web Services** (AWS) or **Microsoft Azure**. These two choices both had their pros and cons with AWS utilising elastic cloud computing (EC2) allowing for the available resource footprint to shrink or increase based on demand, leading to greater scalability and a smaller performance load on average. However, Azure boasted great functionality and a simple, yet effective user interface. This meant in terms of ease of use and learning curve Azure came out on top. The deciding factor ended up being that Azure was restricted to the use of Microsoft SQL, which although similar to MySQL has restricted usage due to being a Microsoft product so cross compatibility could cause issues down the line. Leading us to go with **Amazon Web Services** in the end.

Web Application Host

As with the database system we also had to decide what platform to use to host a web application. Thankfully, both **Microsoft Azure** and **AWS** offered these services. AWS allowed hosting of sites through its **Elastic Beanstalk** service. This service let users package web applications made through a variety of languages and upload them straight to the web. Elastic Beanstalk handles every aspect in the cloud upload, including capacity provisioning, load balancing, scaling, and application health monitoring. Azure also boasts these features and since being a newer infrastructure is again more user friendly and understandable. Due to the database already being hosted on AWS we decided it was best that we used **Elastic Beanstalk** to upload our web application too. As it meant we could more easily link web application to our database. Furthermore, we believed the tougher learning curve of AWS would

allow us to better understand the infrastructure we are using, plus possibly helping in future cloud related projects.



Workflow diagram showcases how Elastic Beanstalk can actively replace and update the current environment in production.

Database Structure

The database is the centre of our project allowing us to link our different system components together. The database is responsible for:

Local Data

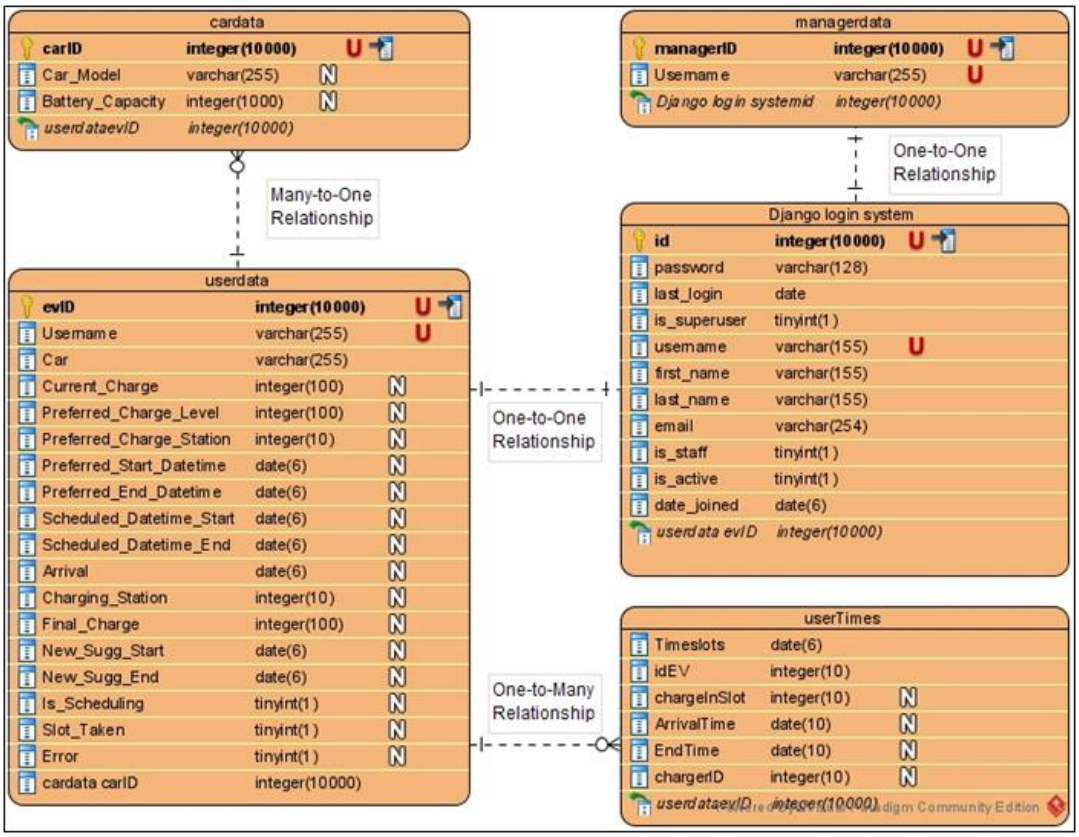
- Storing data on electric vehicles and their battery capacities.
- Storing offline weather and energy data for testing.

Web App

- Storing user and manager login details.
- Storing user charging preferences.
- Storing validation outputs.

Scheduler

- Storing feedback from the scheduler (incomplete scheduling, errors…).
- Storing scheduled slots returned from the scheduler.
- Storing alternative time slots for users.



Entity-Relationship diagram showcases links between different tables within the database.

From the diagram it can be seen that our core table is **userdata**, which is where our user's charging preferences and scheduler outputs are saved. This table uses the **cardata** table to extract the **battery capacity** of a user's given vehicle. As the battery capacity is required for scheduling an electric car. The **Django** table stores all the user/ manager login details for the web application. This is gets updated whenever a user or manager creates a new account and/ or changes their password. While the **userTimes** table is used to get the schedule info of currently scheduled vehicles at each 15-minute timeslot interval. These values are then passed to the scheduler to help calculate energy demand and clashing times.

Security

The password data imported to the **Django** table is converted to a **SHA256 hash** through the use of an algorithm. Meaning there is no way to access a password once it has been set and one must reset their password if forgotten. This feature is important as it makes sure the integrity of the application is up to industry standards. As the concern of a data breach is very valid even for a project of this scope. Django also implements password validators to make sure any password inputted follow certain guidelines to prevent easily breakable passwords.

When searching for data in the database and/ or validating user inputs, we followed strict syntaxes to make sure our data was secure from issues such as **SQL Injection.** This was done by making all search queries have their passed value validated and searched before being ran through the database. Furthermore, a reason we decided to use Django as our web framework is because of its excellent security and as such it has built-in **SQL Injection prevention**. This meant when importing and exporting data in the back end we didn't have to worry about any SQL breaches. However, when accessing the database from the **Scheduler** this had to be looked at carefully. In addition to this, **Django** also sports **Clickjacking Protection, SSL/HTTPS Security and Host Header Validation.**

Data inputs / API

As an input to the scheduler's timeslot info, a list of renewables and traditional fuel production values is produced using the BMRS API from Elexon.

This provides data from the National Grid regarding the predicted quantity of renewables (wind and solar) and traditional fuel types for a 48-hour period. The results returned from each request are saved as csv files, which are parsed to return a list of quantity values for both renewable sources and traditional fuel types, in megawatts, for each 15-minute timeslot of a 48-hour period.

To make use of the BMRS API, a user will need to register an account with Elexon, where there will be provided a unique API key, providing access to

the service. Following the BMRS API and Data Push Guide[4], the data available is documented, with instructions on how to register for the API, and search parameters for constructing any request.

Web application

Web Framework

Our group decided to use **Django** to produce our web application, this was a choice made due to the project implementation required. The scheduler was being coded in **Python**, as Python had strong libraries which had good support for **Machine and Deep Learning** which would help create a more robust scheduling system. Due to this, Django was the most obvious choice for our application as although it was a new framework for the group it allowed for better integration between the scheduler and back end. This reduced the uncertainty when it came to testing the integration as both systems were running the same code base.

On top of this, thinking of the project in the real-world, having the ability to package our scheduler within our back end allows for a more compact and portable solution. This would make it a more desirable and efficient product as it can be easily transferred to multiple systems with little hassle. Another benefit was the reusability of code throughout both systems, allowing the group to cut down on time that otherwise would have been spent on transferring over implementations from the opposite system into another language.

Django also excels at database integration because of its **ORM (Object-Relational-Mapping) layer** this means a developer using Django won't have to deal with using the right syntax for their specific database when querying, instead they are able to use Python code to search, update, insert and delete values from a database. This is a massive benefit as our project heavily relies on database queries.

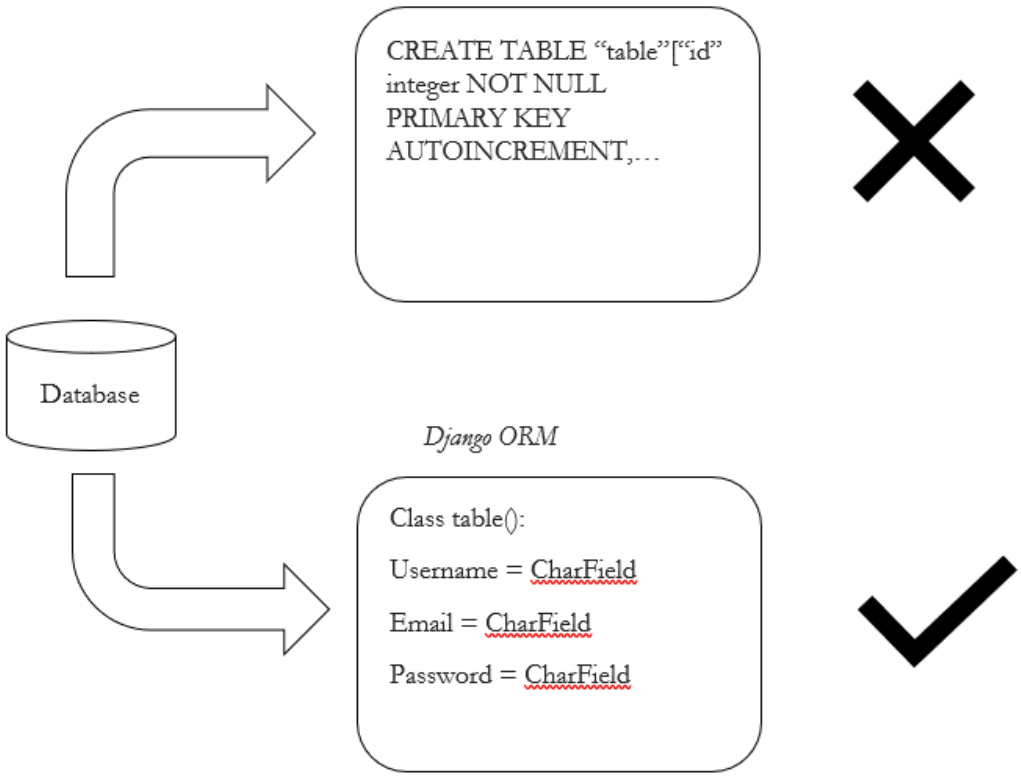
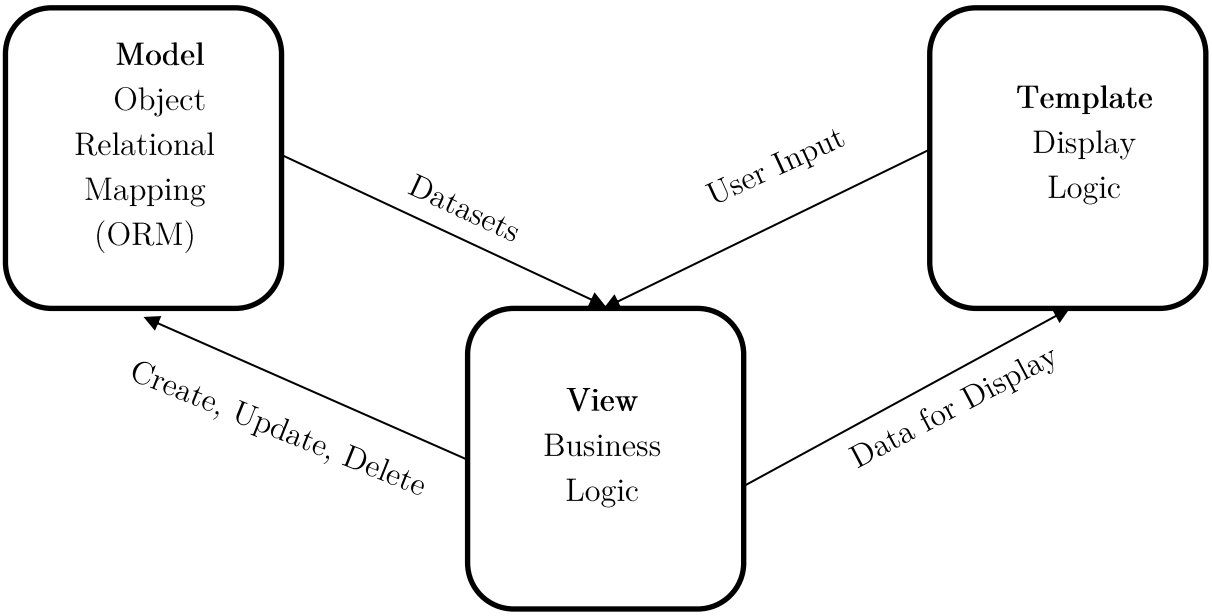


Diagram showcases simplicity of Django querying compared to using a database specific syntax.

Django Design Pattern

Django follows an MVT(Model-View-Template) architecture structure, this closely resembles the MVC (Model-View-Controller) pattern.

However, in the case of Django's MVT pattern the framework will take care of the Controller section (the code that facilitates the interactions between the Model and View), leaving us with the template. The template is a mixture of pure HTML file with Django's own template language (DTL).



MVT Pattern in Django

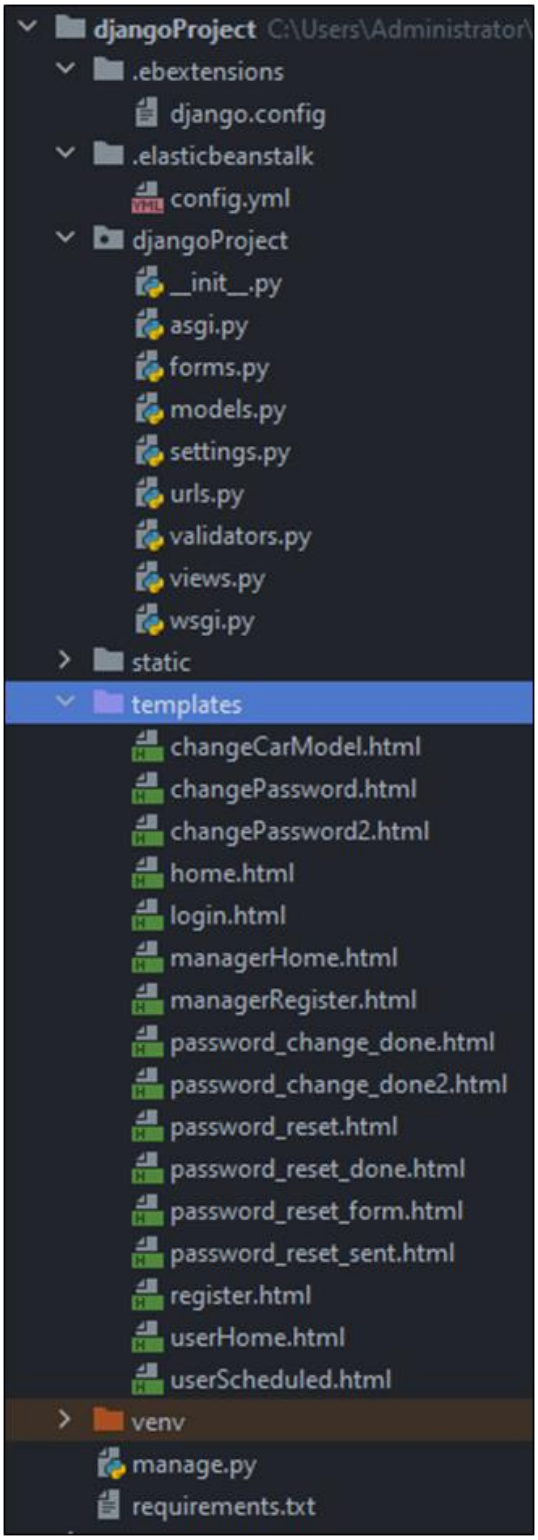
Web App Structure

Front end

- Bootstrap Template
- HTML
- CSS
- JavaScript

Back end

- Python
- SQL Queries
- Elastic Beanstalk Configuration



Django's root directory contains the back-end code used to run the web application. The files are each unique and have their own role in the framework.

___init.py:

Initialises the MySQL connector for use with our framework and tells Django the package **djangoProject** is an app.

settings.py:

This contains all the configuration settings for the project, here is where the database is passed in and set up to link with framework. Other settings include what hosts are allowed access, setup of the templates, and the validators to be used by Django for user input validation, and more.

urls.py:

This is the root URL configuration page for the project, here templates are linked to a corresponding id so they can be referenced around the application.

wsgi.py / asgi.py:

Used when deploying the application to the web, required to deploy using Python. Allows WSGI and ASGI compatible web servers to serve your project.

models.py:

Contains all the models used within the application, this data is used to fetch our database values in **views.py**.

forms.py:

Alters the default Django user creation form, adds an email field as well as sets validation for this newly added field.

validators.py:

Contains a custom validator to make sure the email field has a unique index assigned to it.

views.py:

This file is where most of the processing and querying occurs. In views user forms are validated and inserted into the database. Views also controls what data is shown by the template.

Package – static:

Static holds all styling assets for the templates, it is packaged here so all styling elements are in one location for when we upload our application to the server. Otherwise the styling may not load once it's served.

Templates:

Templates holds all the HTML files in the project.

Requirements.txt:

Contains all libraries used within this Django project, this means any user who runs this project can automatically install all dependencies through this file.

Simulation

One of the ways used to test the scheduler and observe its effect on a few charging stations, an offline simulation software was developed. This software was written in python using the python library SimPy to simulate the environments. Within said software, an environment can either be simulated with a scheduler or without. To further explain, the simulation can be set up to run with just the randomness of vehicles arriving to the charging stations, or it can be set up to run with the vehicles charging according to the scheduler. The two different simulation modes are each on a different file (unscheduled.py and scheduled.py), so that each could be run separately.

Unscheduled.py consists of 2 classes. The first class being Vehicle and the second class being ChargingStation. The Vehicle class follows the VehicleInfo class of the lp_scheduler.py, just used differently for the simulation. Hence it has all the same parameters in its constructor. Whereas the ChargingStation

class just takes a list of timeslots as a parameter. The ChargingStation class main job is the simulation of vehicles arriving, charging, and leaving the station at given times with different charging capacities, hence the name.

The ChargingStation class contains 2 methods that do all the logic behind the simulation, one calling the other. As one might think, those methods are going to be fairly simple and straightforward for the unscheduled.py simulation, as it is vehicles arriving, charging, and leaving. Those methods are named car and simcar. The car method takes in the following parameters:

Env	The SimPy environment
Name	The car name
Bcs	Battery charging station
Driving_time	The driving time till the charge station
Current_charge	Current state of charge of the car
Battery_capacity	The Car's battery Capacity

The car method adds all the attributes/calculations to the respective variable for each car. For instance, when the method is called (a car arrives), it tells the environment that a car has arrived during the timeslot n, and that it needs X time to charge, so it will leave around time slot n+X. The timeslots in the environment are thought of as 15-minute slots, 96 of them to be exact. To elaborate, if a car arrives at time slot 3 and leaves at time slot 5, staying for 2 timeslots, it means that the car stayed for 30 minutes. The car method is also sending text output to the user updating them with what is happening. For instance, when a car arrives at time slot 7, the car method calculates what time timeslot 7 is, and tells the user " CarName has arrived at Time with with X charge and Y battery capacity ". The method does the same for almost any action the car takes or undergoes, like leaving or calculating how much time is left till full charge. Additionally, the simcar method simply calls the car method a several times with randomised initial values at randomised times during the environment run to simulate several cars arriving at different times.

As for scheduled.py a very similar approach and structure were taken to build the simulation environment. Just like unscheduled.py, a class for the

charging stations was created, UChargingStation. The class followed the ChargingStation implementation of having the methods car and simcars to create the simulation functionality. However, the main differences here are the parameters the class for the constructor and the functionality of the member methods and, of course, the use of the lp_scheduler to assign cars time slots.

The constructor of UChargingStation takes a list of scheduledcars and a list of max loads a charging point can have at a given point as parameters. The car method has different parameters than the one in unscheduled.py as well. In scheduled.py it takes just the environment, a timetable, and the scheduled load. Unlike unscheduled.py, more functionality here is implemented in the simcars method rather than car method. All the car method does is update the load on the grid depending on how many cars are charging in a particular time slot. For instance, if 1 car is charging with charge rate of 22KwH, then the load on the grid is increased by 22 whilst its charging.

As for the simCars method, it simply loads renewable energy on the grid, maximum load for the grid at a given point, and the load on the grid from external generators. It then calls the scheduler, giving it the list of cars that requested charge, and then the scheduler assigns the cars a time slot, and the time slots are updated accordingly.

In both files, the environment is run until a pre-set number of timeslots is reached, each being a 15-minute time slot. The number of charging stations, cars to be charged can also be manipulated by adjusting the code minorly. The environment runs and calls the simCars method, that consequently calls the car method to replicate a car wanting to charge. Both files also output data that can be then extracted and put on a graph to show how the load on the grid is being affected by the scheduler and whether it surpasses or reaches the maximum load at any given point.

Maintenance and Extensions

Coding Styles and Conventions

The coding conventions used for the project[5] are (loosely) based on PEP8[6] and Google's Python Style Guide[7]. The conventions are chosen and documented such that a developer with not much experience in the language can easily follow them.

Scheduling

There are a few aspects where the scheduling can be improved through additions. The MILP formulation, for instance, might be one place to go about doing so. The optimiser (MILP modeller and solver) is separate from the main LP Scheduler interface that clients can interact with, so more factors can be added to help influence the optimisation decisions. Should new information be required to realise this, adding the information to the parameter objects as new inputs to the scheduler should suffice.

Any additions or changes to the model requires the use of the DOcplex API[8].

Equally, should a different method of optimising the charge allocation is required, a new optimiser class can be created as long as it supports the optimise method.

Another aspect where the scheduler may be improved is the collection of possible allocation strategies. The current solution provides only three strategies—one based purely on the user's preferred charging period, one based purely on the availability of production of renewables in each time interval, and one based purely on the total price of each charging period. In the same way that the MILP model can be modified and improved, any extra input requirements can also be achieved by adding to the parameter objects.

Optimising for Performance

The scheduler currently does not experience performance issues. However, these should be kept in mind when maintaining and extending the code base, particularly when it comes to the mathematical model.

There should be a balance of readable and easier to understand code with optimised code, with the former being the preferred target—that is, one should avoid prematurely optimising and attempt to make code as readable as possible.

Testing

Testing was performed using a mix of formal and informal testing. Formal tests were written using Python's unittest framework[9]. Due to the nature of the scheduler, it will be difficult to test for every case. Most of the testing had been done informally as new requirements and problems came along.

For future work on the system, tests for basic functionality and some correctness may be performed by writing unit tests with the framework. Moreover, any further work on the scheduling algorithm may benefit from using and potentially extending the simulation as required. A suggestion on how extensions to the scheduling may be 'tested' is to simply generate or create visual representations of the modelling result (e.g., a line chart) with random or manually designed pre-determined data.

Whether one is writing formal unit and integration tests or informally testing the code with manually designed inputs, adding more some form of documentation would be extremely helpful. Current documentation of unit testing is formatted as simple test plan tables to save time with a more agile approach[10]; each test case should have at minimum, at least an ID, a description and a pass or fail column.

For completion's sake, it would also be beneficial in the future to include some sort of test documentation with 'informal tests' that references the commit and had descriptions of what was tested and why in addition to how this

was done. These are not expected to be long because testing informally suggests a lack of time or resources to write proper tests; these suggested documentations are simply ways of aiding developers to remember what had been done. Should there be time for proper unit and integration testing, these should be done instead of writing these 'informal' test documents.

A suggested format[11] for such documents is outlined below:

1. Description of commit additions/extensions

2. Commit hash

3. Verbal description of testing, problems that arose, and how these were fixed (if unable to do so, then any notes that could help in fixing them).

The following is an example of a test plan table.

LPA 1.x

Test	Name	Description	Pass/Fail
LPS 1.1	Charges vehicles to demand	The scheduler should attempt to charge the vehicles up to their chosen charge level. Given that full charging is possible with the circumstances, by the end of each vehicle's charging period, they must have been charged to the charge they have chosen before scheduling.	Pass

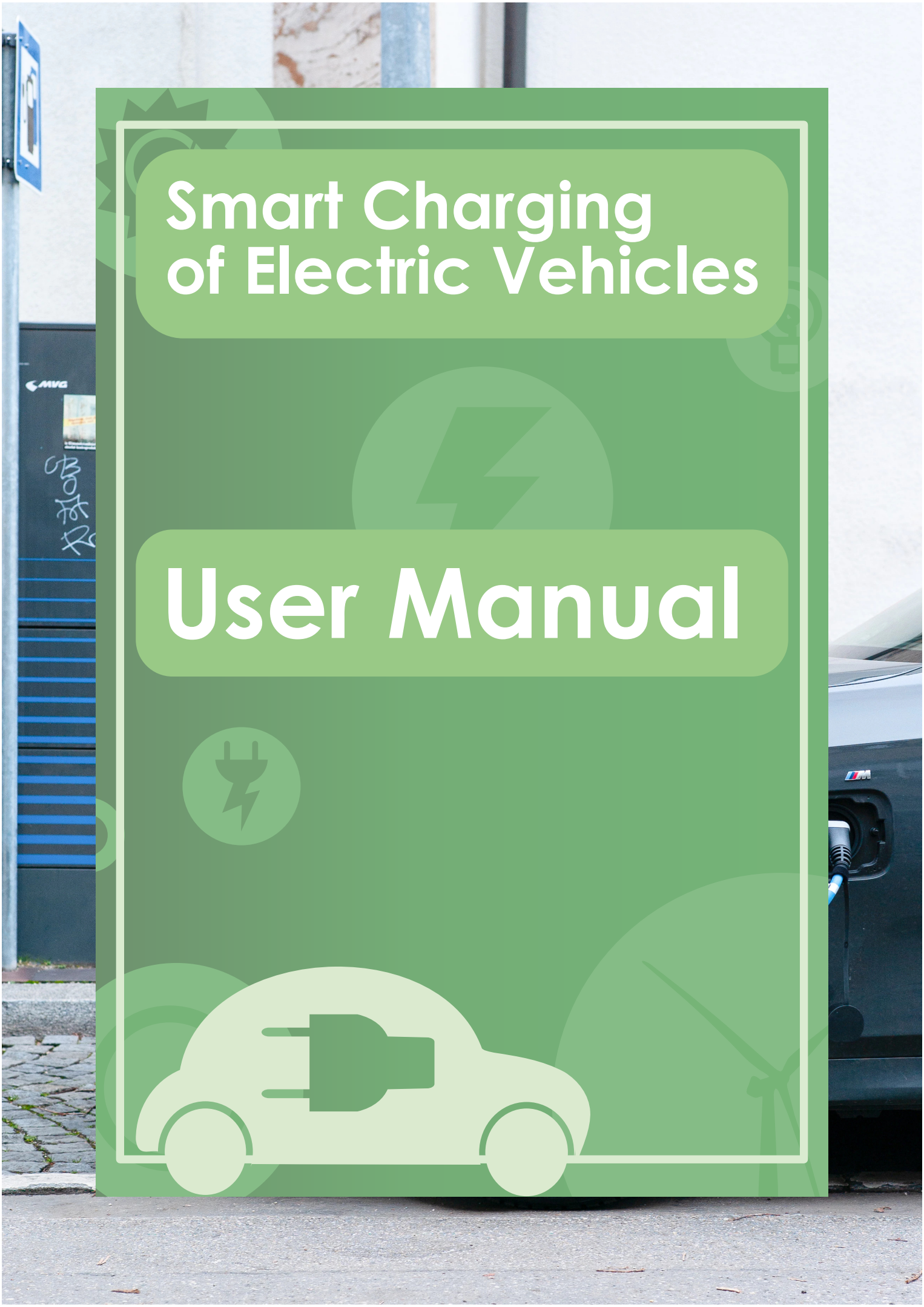
Conclusion

In summary, we believe we met most of the requirements we had set for ourselves at the beginning of this project and are happy with the outcome of our system. However, moving forward if our scheduler were to move to a

commercial setting, we have outlined essential features that will transform the project to a production standard. Firstly, to increase transferability across multiple users we would use a software container such as Docker. This would allow us to group our code under one system and set the requirements for installation. So, when a new user wants to access the scheduler, they can unpack this container, install the given requirements and be able to freely use our system. In addition to this, it would allow us to upload our CPLEX model to our web server instead of having to have the scheduler run locally. Another implementation would be to switch a self-learning model instead of a mathematical algorithm. This would increase the accuracy of the schedulers output as well as allow for the scheduler to evolve into predicting future charging times based on user's daily schedules

Smart Charging of Electric Vehicles

User Manual



What is a smart scheduler?

The smart scheduler is a system of software components that allows users to schedules their charging in advance or as they require it on the go.

To interact with the smart scheduler, users can use the interface provided by the web application (see below on how to access the website).

Requirements

The only requirement to access the site is a stable connection to the web with any of the common browsers, such as Google Chrome or Mozilla Firefox.

To connect, users can go to this website:

evscheduler.eba-65vkp7qm.eu-west-2.elasticbeanstalk.com

Users are also required to register to the website in order to use the scheduling system.

Please keep in mind that the following pieces of information is stored online in the software database:

- User's email address
- User's password for the website
- User's car model
- User's schedule information

Why use the smart scheduler?

The smart scheduler application is designed for two types of users: electric vehicle owners and facility manages.

Electric vehicle owners can use the website to request scheduled times during which they will be expected to charge their electric vehicle.

Facility managers are use the website to view which charging points in the area are being used; the system is designed to provide facility managers a way to analyse the use of charging points for use in identifying areas where their charging system could be improved and how this could affect other areas.

For users, smart scheduling is a way of getting 'smart' and functional schedules that help with reducing the load of charging vehicles on the grid whilst also helping the environment by making use of renewable energy on the grid that is often not fully utilised.

The smart scheduler also helps organisations and owners save money by making use of cheaper charging tariffs.

Quick Start (EV Owners and Facility Managers)

1. Sign up for an account on the smart scheduler website.

2. You are required to enter a unique username, an email address, and a password.

If you are an electric vehicle owner looking to use the system to request schedules, you will also have to enter the model of your car (this can be changed later).

3. Login to the website using the username and password that you have entered during registration.

4. As an electric vehicle owner, you can request a schedule by selecting the charging point you will be using, the time you will be available to charge the vehicle, your battery's state of charge at arrival and what charge percentage you would like to have after the charging is finished.

As a facility manager, you are able to view all schedules and which charging points cars are using.

For further details on the steps, please refer to the relevant manual pages.

How will my schedule be created?

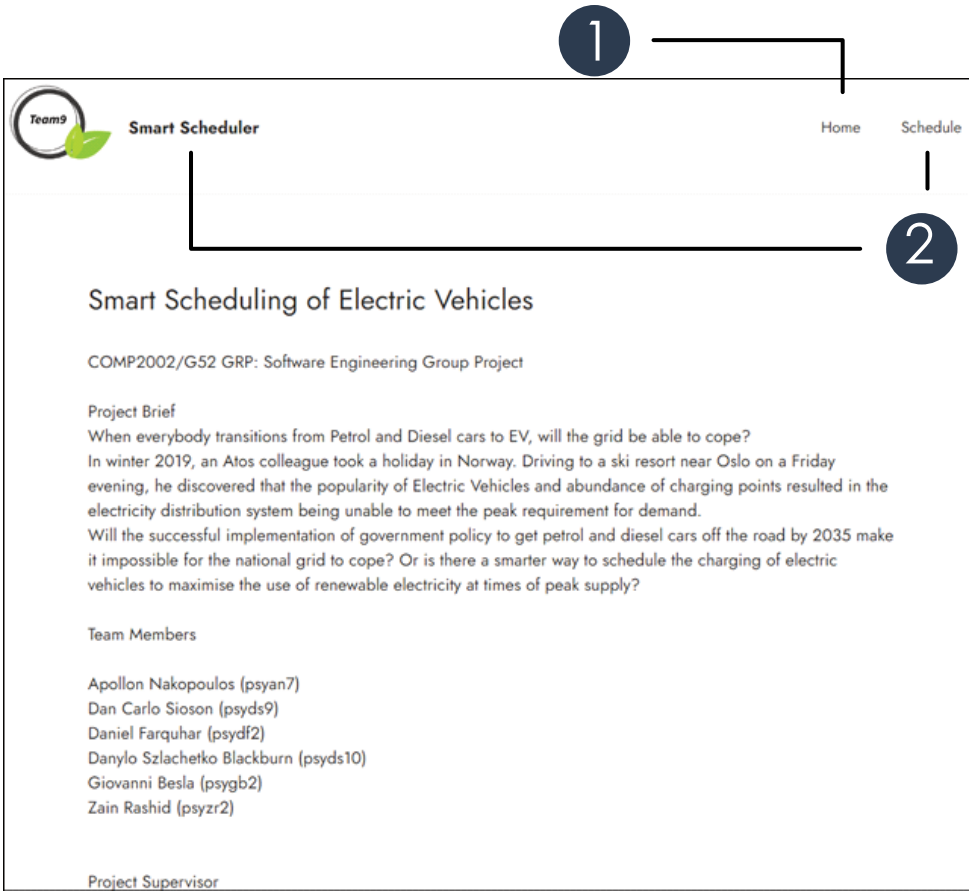
The scheduler uses services to connect to the grid to obtain data on the national grid.

This data is processed along with user information to create schedules that take into the account the following information:

- Use of renewable electricity in the grid
- Charging costs
- Availability of charging points
- Other users that are scheduled

Using these pieces of information, the scheduler uses algorithms to find schedules that not only meet electric vehicle owners' demand, but also schedules that prevent overloading of the national grid, are more environmentally-friendly, and save money.

Home Page



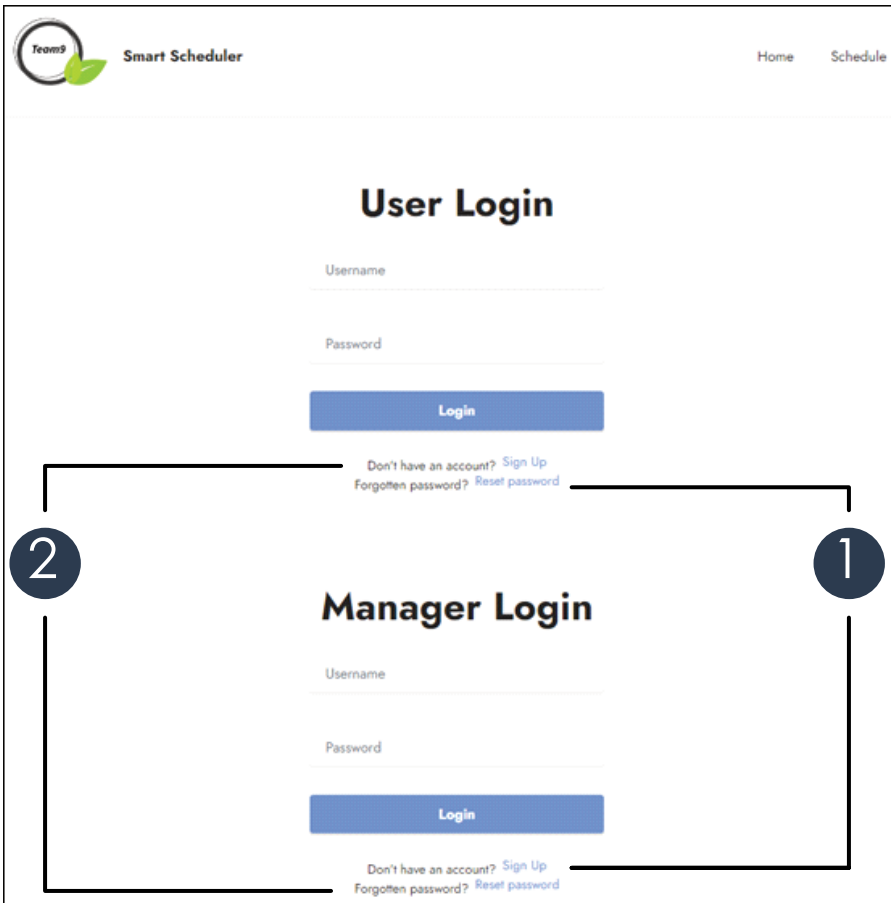
When users first enter the site, they will be greeted with a homepage displaying a rundown of the goal of the project and the parties involved.

(1) The "Home" option and the website name can be clicked, allowing users to return to the homepage if they wish to do so.

(2) To start the scheduling process, users can click the "Schedule" option along the navigation bar located at the top right portion of the page.

Users may register through the login page (see the next page of this manual).

Logging In



Once users access the "Schedule" page they are greeted with this login page. They will have the choice of either logging in as an electric vehicle owner to schedule their own electric vehicle, or a manager who has access to data of vehicles whom have already been scheduled.

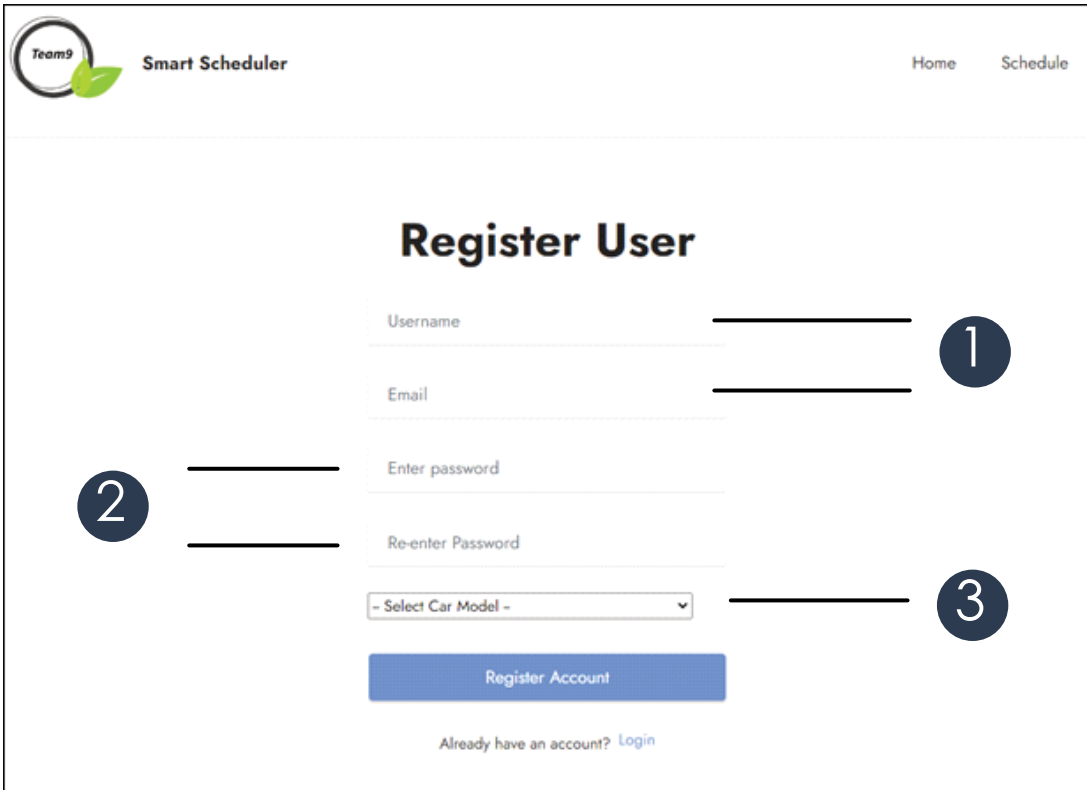
(1) If the user hasn't got an account, they can use the sign-up link to redirect them to a registration page.

This option is also available to managers.

(2) If the user has forgotten their password, they can request to reset their password through their email.

This option is also available to managers.

User Registration - Electric Vehicle Owners



Clicking the sign up link at the bottom of the login fields will bring up this page for users to sign up to the system.

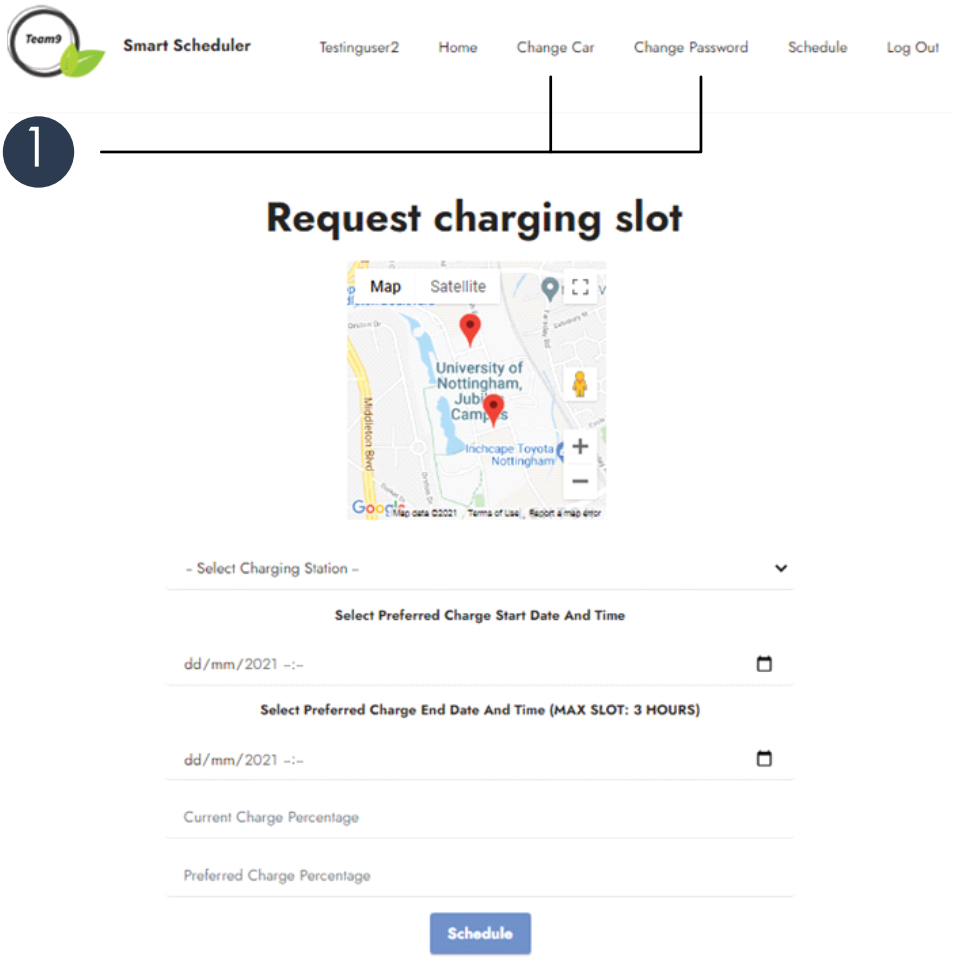
(1) Users must enter a unique username and email.

(2) Users must also enter a password. This password is validated to make sure that it is not a password that is easy to guess. Their preferred password must be entered twice to ensure that their password was not entered incorrectly.

(3) Finally the user will select what electric car they have ownership of, to help with scheduler calculations.

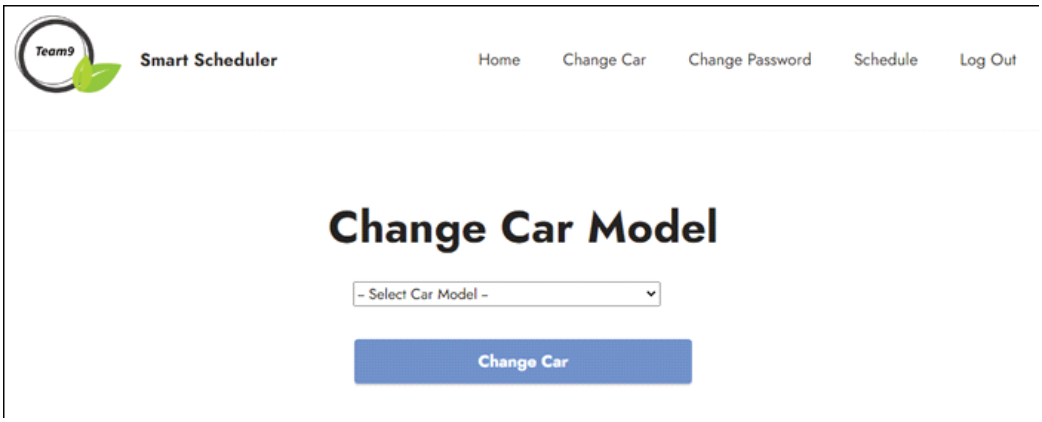
(4) Clicking "Register Account" will register the user once they have entered valid login information.

Changing User Information



(1) Users can change their car model or passwords using the navigation bar once they are logged in by clicking the appropriate buttons in the navigation bar located at the top of the page.

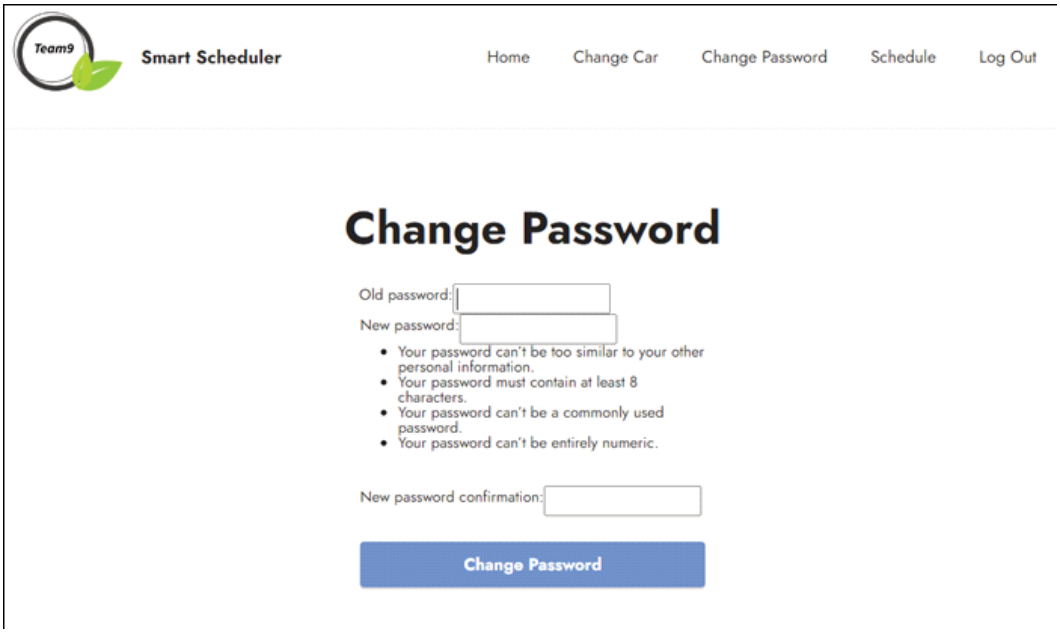
Changing Car Model



Vehicle owners can change the car model associated with their account should they switch vehicles.

Users can click the drop down list to see the list of all supported vehicles and choose the model of their electric vehicle.

Changing Password



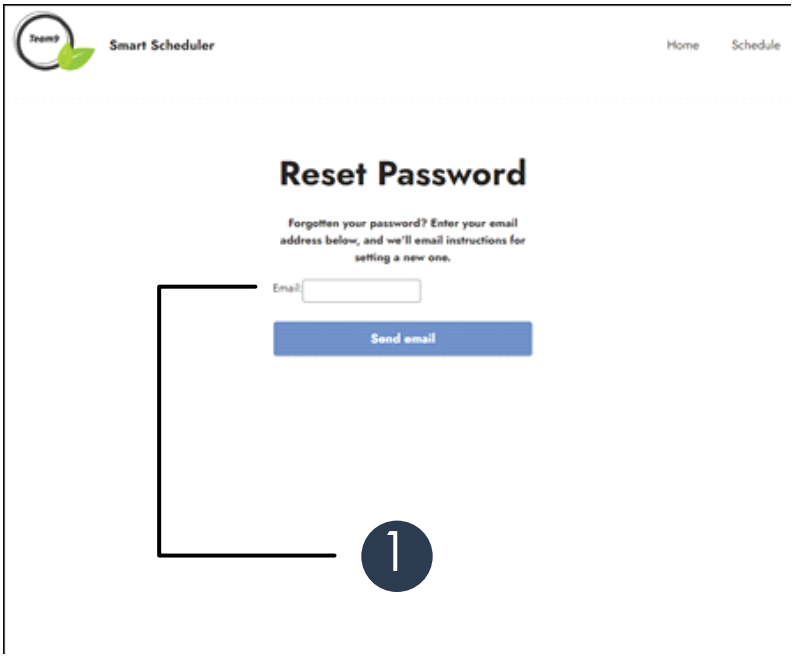
Both vehicle owners and facility manager can change their passwords in the same way.

They must first enter their original password before they are able to enter a new password.

This new password must be unique, it must not be the same as the previous password, and it must should not be a password commonly used (e.g., 'password' as password).

It is recommended to use strong passwords for user accounts (see *Keeping your Account Secure*).

Resetting User Password

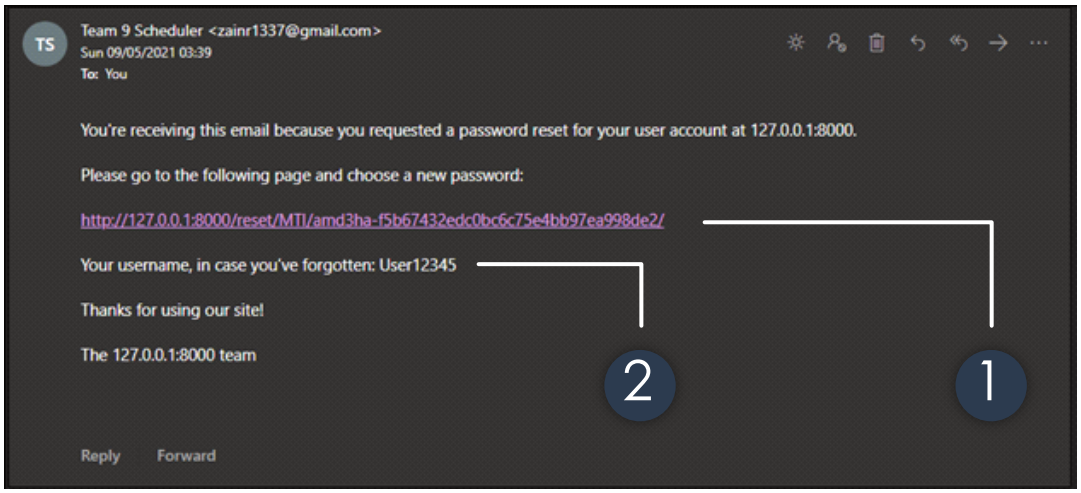


If users have forgotten their password, they will be redirected to this page.

(1) This prompts users to enter the email associated with their manager or electric vehicle owner account. This allows the site to email a reset link to the user for security reasons.

Users should wait for the email to appear on their account. This may take a long time depending on the status of the web server.

After an email has been entered, the user will be sent an email similar to the one below:



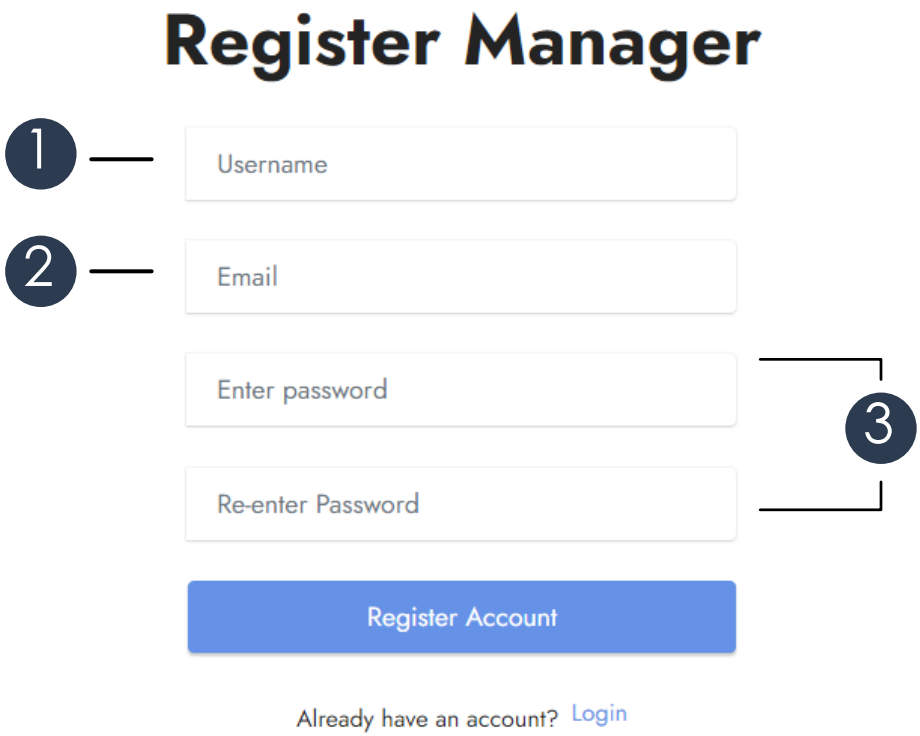
Receiving this email means that the system has received the user's request to reset their password and has provided them a way to do so.

(1) In this email the user will receive a uniquely generated address which will lead them to a page which will allow them to enter a new password.

(2) It will also remind the user of their username as well in case they forgot.

After the user's password has been reset using the link provided in the email, they must now login to the site using their new password.

User Registration - Facility Managers



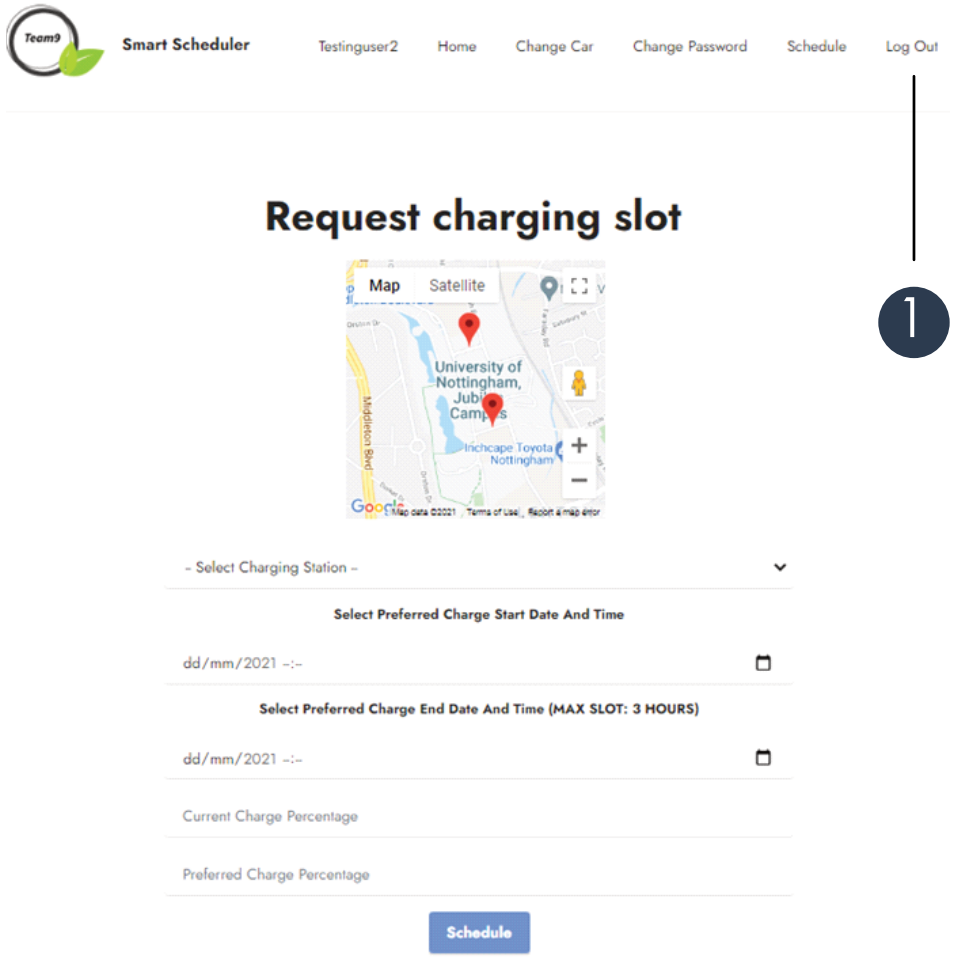
When a manager wants to register a new account, they are prompted with this page.

(1) Managers must enter a unique username.

(2) Managers must also enter their email address.

(2) Manager must enter a valid password. This password is validated to make sure that it is not a password that is easy to guess. Their preferred password must be entered twice to ensure that their password was not entered incorrectly.

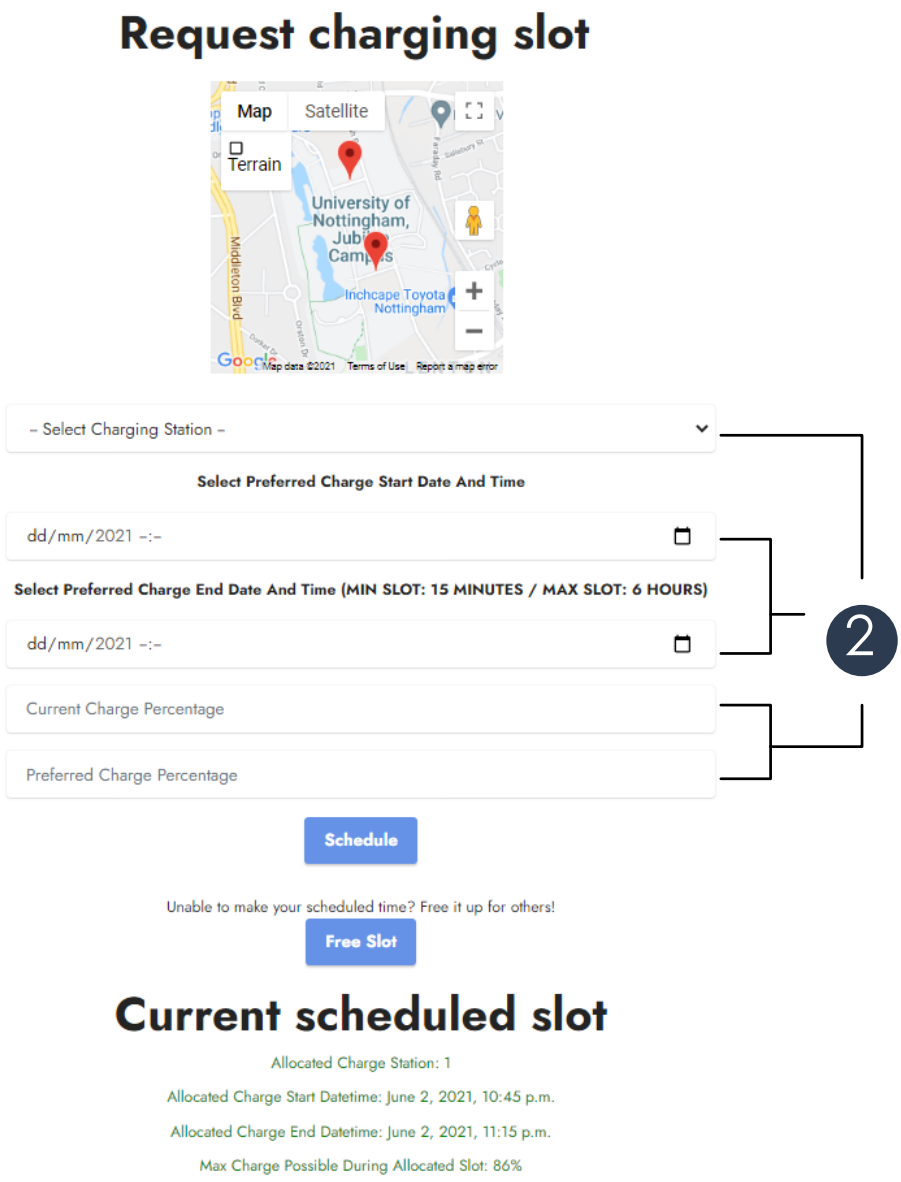
Signing Out



(1) Users can click "Log Out" to sign out of the website and be redirected to the login page.

Using the Smart Scheduler Web Application to Request Schedules

Requesting Schedules



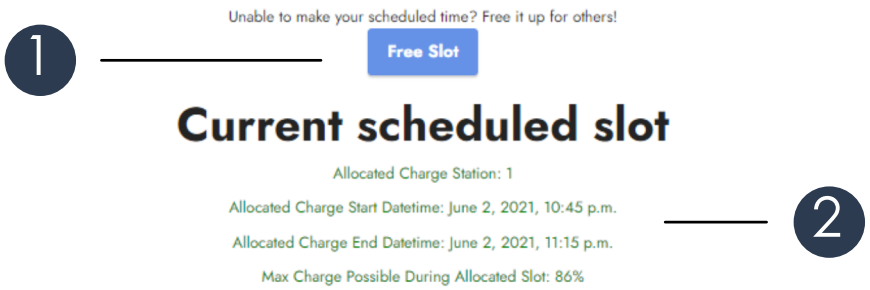
This page is presented once a user has successfully logged in.

It allows them to enter their charging preferences to schedule a charging slot.

(1) An interactive Google Maps section can be used to view the locations of charging stations.

(2) Users can enter the following information:

- Choice of Charging Station
- Available time period to charge
- Battery percentage at arrival
- Preferred battery percentage after charging



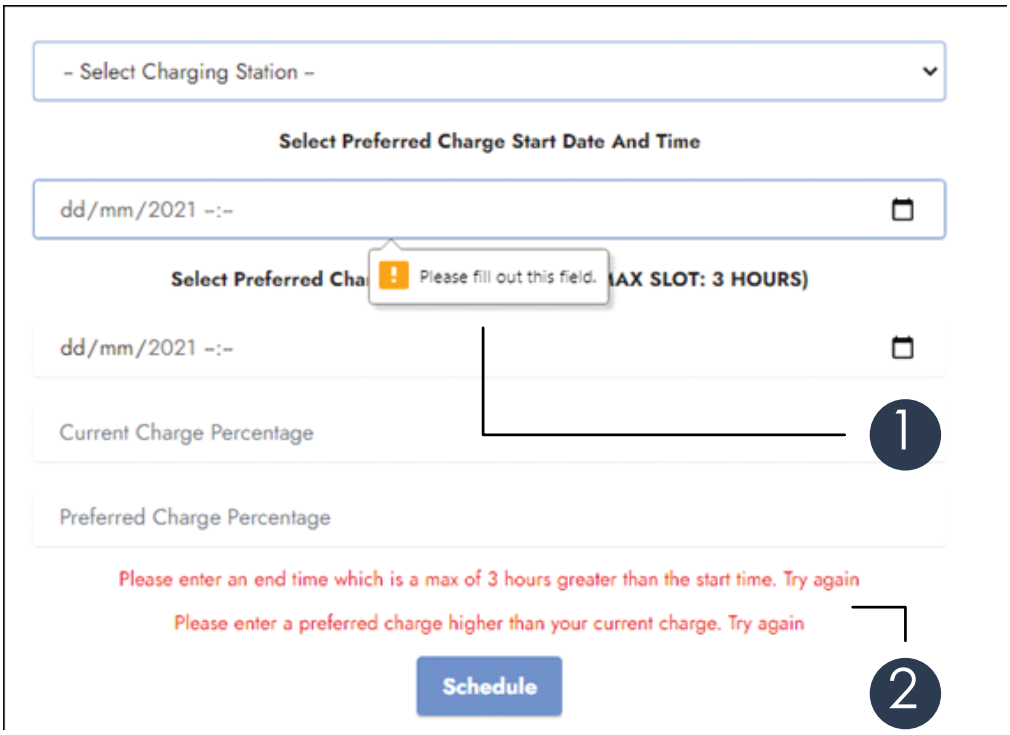
(1) Schedules can be freed up at the user's behest. Users must do this in order to

(2) If users have already attempted to schedule previously, this slot will be displayed in case they have forgotten the details of their charge schedule allocation.

If schedules have expired, they will no longer be displayed to users.

Entering Incorrect Details

Warnings and Entering Invalid Charge

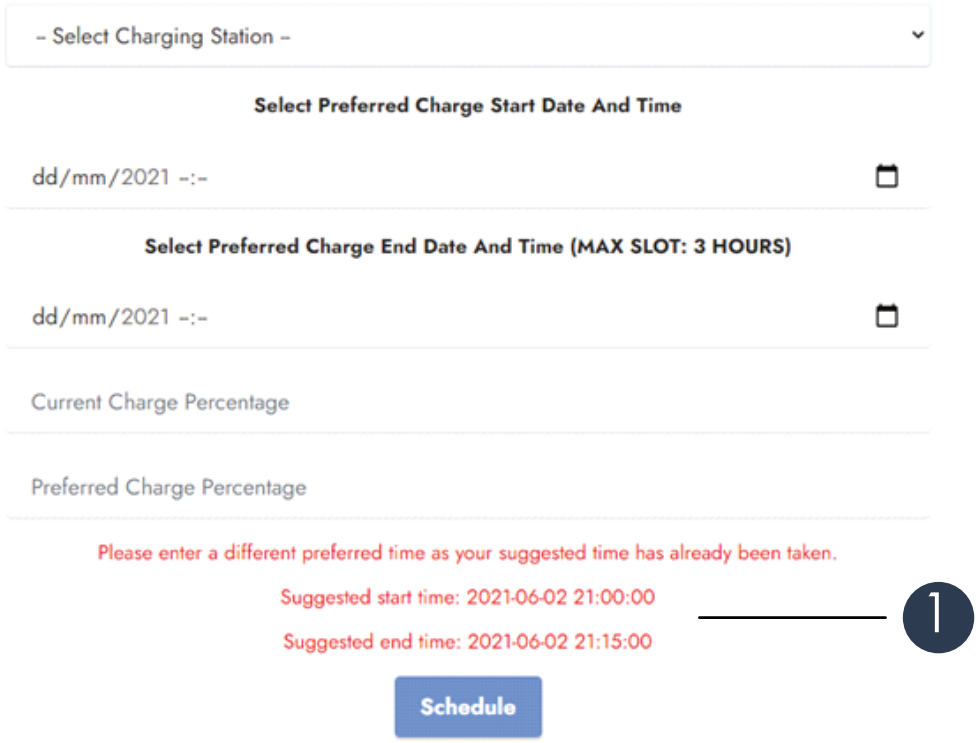


(1) If users forget to fill out a field, they will see a warning message to remind them to complete the form.

(2) Entering invalid information such as a charge percentage demand lower than their state of charge at arrival will also bring up a message.

These error messages will tell users what went wrong and what information the system requires to schedule them.

Scheduling with Invalid Time Period



Not every schedule can always be met. A schedule may be invalid because other people have already scheduled at the chosen charging points during the exact same period.

(1) If the user tries to schedule a time that has already been taken they'll get a prompt telling them this, in addition to suggesting a newly calculated time which is free and based around the users current preferences.

Scheduling Process

Loading Screen

Your car is currently being scheduled!

Processing user data...

Processing local energy and weather data...

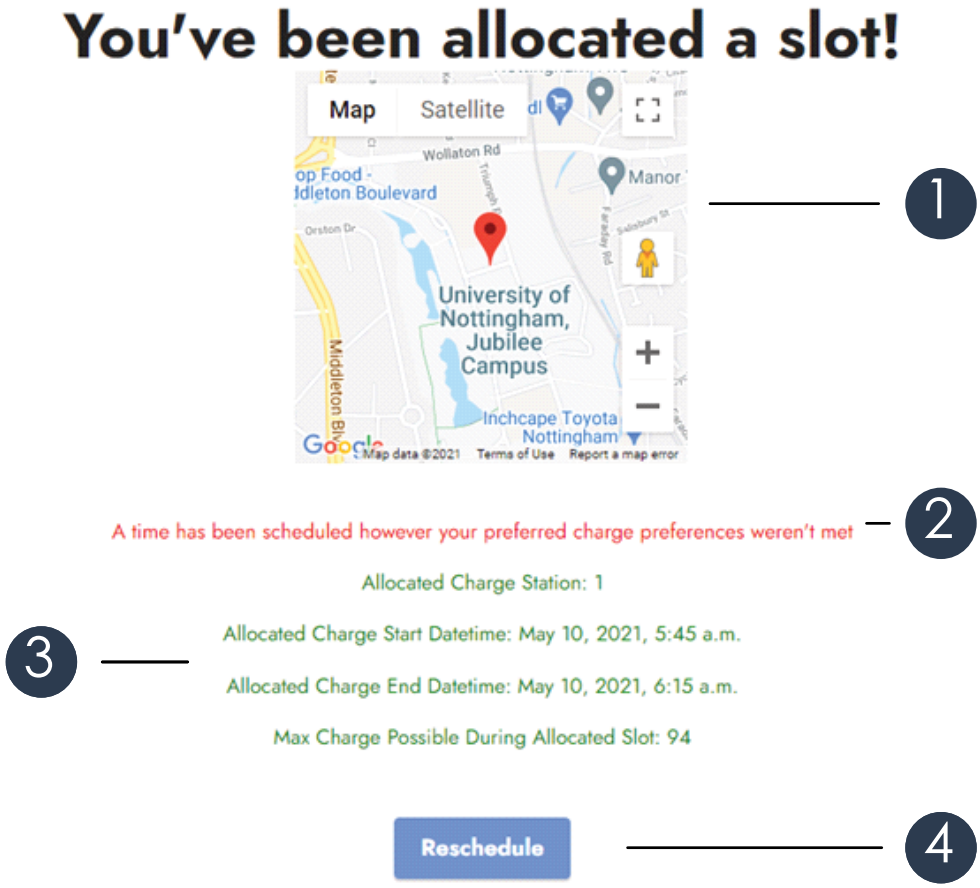
Checking for clashing times...

Scheduling slot...

Once a user has entered their preferred schedule details and these are validated by the system, the user will see a loading screen. This may take some time.

Once the system is done scheduling, the website will display the user's schedule (see next page).

After Clicking "Schedule"



(1) After the slot has been scheduled the user can see the position of their charging station again.

(2) Unfortunately, not every requirement of users' requests may be met. Should this happen, a message will be displayed to show this.

(3) The newly scheduled slot will be showcased here for the user showing the start and end date and time. As well as the allocated charging station and the max charge difference they can get from their current charge percentage.

(4) While the scheduler tries to prioritise users' requirements, the system is not always able to do so. Should users require want to reschedule with different times, they can click the "Reschedule" button that appears underneath the map and messages.

Using the Smart Scheduler Web Application to View Usage of Charging Points

Viewing Schedules as a Facility Manager

1

Charging station 1

Username	Current Vehicle Charge	Preferred Charging Level	Preferred Start Datetime	Preferred End Datetime	Charging Station	Allocated Charging Start Datetime	Allocated Charging End Datetime	Max Charge During Slot	Arrival Datetime
zain	1	32	05/05/2020 5:34 p.m.	05/06/2020 5:37 p.m.	1	05/05/2020 5:45 p.m.	05/05/2020 6:15 p.m.	32	10/04/2020 5:50 p.m.
TestUser	10	30	04/22/2020 11:11 a.m.	04/27/2020 12:49 a.m.	1	05/02/2020 5:30 p.m.	05/02/2020 10:45 p.m.	32	04/23/2020 10:49 p.m.
Zain123	1	100	04/28/2020 1:43 p.m.	04/29/2020 1:43 p.m.	1	05/02/2020 10 p.m.	05/03/2020 12:45 a.m.	32	04/27/2020 1:43 p.m.
TestingUser	2	43	05/16/2021 8:27 p.m.	05/16/2021 8:34 p.m.	1	05/16/2021 8:45 p.m.	05/16/2021 9 p.m.	32	05/02/2021 8:27 p.m.
User567	1	10	12/23/1234 11:23 a.m.	12/23/1234 11:23 a.m.	1	12/23/1234 11:23 a.m.	12/23/1234 11:23 a.m.	32	12/23/1234 11:23 a.m.
TestUser101	1	100	05/18/2021 8:57 p.m.	05/18/2021 10:59 p.m.	1	05/18/2021 9:15 p.m.	05/18/2021 10 p.m.	32	05/03/2021 8:58 p.m.
JohnDoe	1	100	06/02/2021 10 p.m.	06/02/2021 10:30 p.m.	1	06/02/2021 10 p.m.	06/02/2021 10:30 p.m.	32	05/06/2021 12:40 a.m.
Testinguser1	1	100	05/25/2021 1:02 a.m.	05/25/2021 1:25 a.m.	1	05/25/2021 1 a.m.	05/25/2021 1:30 a.m.	32	05/06/2021 1:02 a.m.
Testinguser2	1	100	06/02/2021 10:30 p.m.	06/02/2021 11 p.m.	1	06/02/2021 10:45 p.m.	06/02/2021 11:15 p.m.	86	05/12/2021 9:59 a.m.
Testinguser7	1	50	06/02/2021 10:30 p.m.	06/02/2021 10:45 p.m.	1	06/02/2021 10:30 p.m.	06/02/2021 10:45 p.m.	48	05/12/2021 3:04 a.m.
Testinguser8	1	50	06/02/2021 10:45 p.m.	06/02/2021 11 p.m.	2	12/23/1234 11:23 a.m.	12/23/1234 11:23 a.m.	44	05/12/2021 3:30 a.m.

Charging station 2

2

Username	Current Vehicle Charge	Preferred Charging Level	Preferred Start Datetime	Preferred End Datetime	Charging Station	Allocated Charging Start Datetime	Allocated Charging End Datetime	Max Charge During Slot	Arrival Datetime
User1	1	100	06/02/2021 10 p.m.	06/02/2021 10:30 p.m.	2	06/02/2021 10 p.m.	06/02/2021 10:30 p.m.	32	05/02/2021 2:04 a.m.

When logged in as a facility manager, users will be able to see a list of schedules and which charging stations are being used at specific times by clicking the “Schedule” button located on the navigation bar at the top of the page.

This page showcases all currently scheduled times for all users.

(1) User’s charging preferences displayed as well as their allocation details.

(2) Different tables for each charging point.

Facility managers will be able to see electric vehicle owners’ usernames, their charge at arrival, the charge they want when they finish charging, their preferred time period for charging, the charging station they use, their allocated scheduled times, and the total charge they will receive.

Frequently Asked Questions

The schedule I got was much shorter than the time I requested.

This is normal behaviour. The scheduler will give schedules that take the least amount of time whilst still meeting users' demand.

How does the scheduler determine my schedules?

This is detailed in the 'How will my schedule be created?' Section on page 4.

My electric vehicle model does not appear on the drop down list when selecting a model.

The system database on vehicle models is kept up to date by relevant personnel. Unfortunately, it may not always be possible to support electric vehicles as they come out. It is recommended to mention this to technical support as feedback.

I would like to delete my account.

Please contact the relevant personnel (see Data Handling section on page 27).

Keeping your Account Secure

Users should follow common practices on keeping their online login information secure.

When registering to the smart scheduler website, users should:

- Use strong passwords. Strong passwords typically contain a combination of letters, numbers, and symbols and are of the appropriate length (it is recommended to use at least a password of 25 characters).
- Not share their login information to anyone.

Using a password manager will help with keeping users' online login information secure.

Examples:

Great - C@xD@Kg44%Vgpu*M4zK4gXcw3

Good - berry_@spoon23batteryhair

Bad - evpassword

Data Handling

When users register, all data they have entered during the registration process will be stored in the software database. Rest assured, actions are taken to keep any sensitive and personally identifiable data are kept secure.

Please keep in mind that only the following pieces of information are stored online in the software database:

- Username
- User's email address
- User's password for the website
- User's car model
- User's schedule information

The software does not use this information for anything other than any processing required for the scheduling process.

Should users desire for their information to be removed from the database, their account must be deleted.

They must issue a request to do so to the system administrator, or those assigned to handle such matters in their organisation, or any relevant persons.