Thrilltopia

Indoor & Outdoor

Fun & Exciting

Group & Individual

Activities:

Kayaking | Indoor Mini Golf | Indoor & Outdoor Tennis | Indoor Paintball | Climbing Wall | Paddle Boarding | Mountain Biking | & More

CFG Degree Project

Inês Branco Veiga | Vicki Gibbison | Kayla Hong | Paula Kuwałek | Sofia Leong | Marnie Thomson

Reserve Online Today

When weather is not on your side
We have your back!

INTRODUCTION

Thrilltopia is a reservation system for an imaginary activity centre based in Portugal. The aim of this project was to create an application, for the client 'Thrilltopia', which allows customers to explore and book a wide range of indoor and outdoor activities, whilst additionally including the forecasted weather over a 15-day period. The application provides the weather for the date and time the customers choose, so that considerations can be made without the need to for 3rd party websites to check the weather. This application will allow 'Thrilltopia' to operate more efficiently, by managing bookings and equipment hire. It will also ensure that activities are never over capacity. This product will put less strain on resources as the use of computerisation will mean less demand on staff to answer calls regarding bookings and general queries.

The home page will present the customer with a variety of activities, they can select an activity to read more information before selecting make a booking where they will be asked for details included their chosen booking date and time. A forecast for the weather conditions for that date using a publicly accessible weather API will then be presented then the customer can decide to proceed with the booking or select another date. The console application will allow us to display, update and delete a variety of functions, such as all reservations for a specific activity.

We have achieved our aim by the following objectives:

- ◊ The design of the 'Thrilltopia' web application using the Bootstrap framework.
- ◊ Created different sections of the application: reservation home page and an about us section.
- ◊ Connected our front-end to a publicly accessible weather API to present forecasted weather within a 15-day period.
- ◊ Unit testing to ensure the application ran as was expected.
- ◊ A comprehensive, multi-functional frontend console application

Background of our product

The main purpose of 'Thrilltopia' was to create an online reservation system which helps enhance the efficiency of its activity.

Reservation Home Page

When a customer goes onto the website, they will be presented with 12 activities to choose over a 15-date period, in a grid-format. Some of these activities can be done indoors or outdoors, individually or in a group setting, and customers will be given the choice of which they would prefer. Customers will also be presented with a difficulty level for each activity including a minimum age level, which will help them decide whether an activity is appropriate for them, minimising any phone calls between the company and customers. Customers will also be able to make changes to their reservation date, or if they so wish, cancel their reservation.

Weather

The 15-day weather forecast allows users to make decisions on whether they will do an outdoor activity taking into consideration weather conditions on their chosen date. This weather data will include severe risk, wind speed, temperature, and precipitation. This allows the customer to decide based upon the current weather conditions. Additionally, if person capacity has been reached for this specific date and activity then a warning message will be presented to the customer that they cannot book this activity. By default, the equipment will be provided to the customer for their booking, this also includes the necessary safety equipment for each activity. We have also included images, and a description of each activity to make it easier for the customer to decide if they are unsure about a type of activity.

Future Add-Ons

When customers book an activity, they will input their full name, phone number and the number of people on the reservation form. Currently this action will trigger an update on the database with the inserted reservation and customers' information. The intention is that this system is similar to a restaurant booking system, whereby we would look to develop this application further if more time is allowed, the customers will be sent a confirmation email when the reservation is made and a reminder text message 24 hours before their booking date.

SPECIFICATIONS AND DESIGN:

• Functional vs non-functional requirements

Functional requirements:

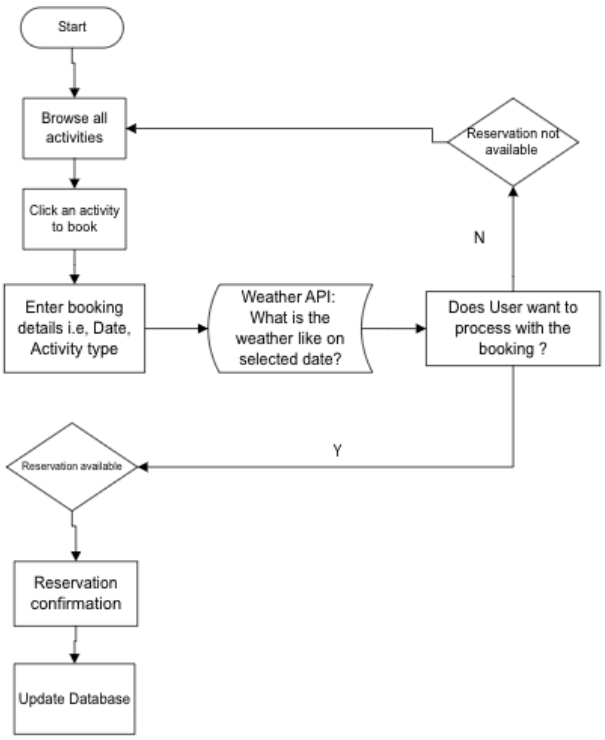
- ◊ A customer should be given a display of all activities 'Thrilltopia' has to offer, alongside their illustrative photographs, descriptions, prices and duration.
- ◊ A customer should be able to make a reservation himself by completing the reservation form.
- ◊ To make a reservation a customer should have to provide their personal details, such as: first name, last name and phone number, and choose a reservation date and time and number of people attending.
- ◊ Date, which a customer can choose must be in the range from reservation day to consecutive 2 weeks.
- ◊ After filling the reservation form, if the chosen activity is an outdoor activity, the customer shall be presented with a weather forecast for a chosen date in the form of a warning.
- ◊ There must be a client's application in the console app form.
- ◊ Should the customer call and wish to change or cancel their reservation it should be possible to do such operations.
- ◊ A client should also be able to search through activities by id, name, outdoor and indoor.
- ◊ A client must be able to see what kind of equipment should be given to the customer for which activity.
- ◊ There should be a possibility to sort activities by price ascending and descending.
- ◊ A client should be able to search through reservations by date and customer's last name.

Non-functional requirements:

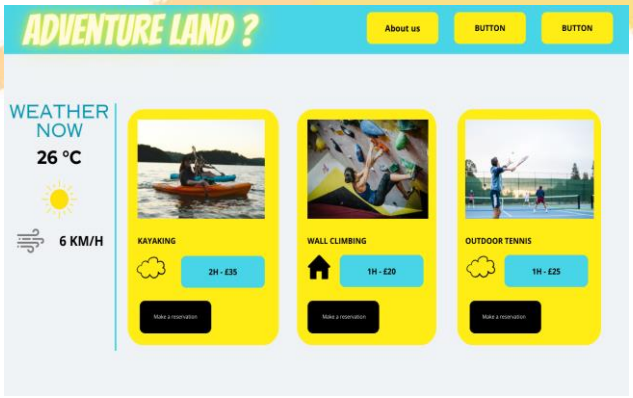
- ◊ The website should be compatible with Chrome, Firefox, Safari and Edge browsers.
- ◊ The console app should work on Python 3.11 and newer. The system should be capable of handling maximum 50 characters long customers first and last name input.
- ◊ The system should be built using licences free of charge.
- ◊ The system should hold up to adding a maximum of 100 reservations daily.
- ◊ The system should be able to handle up to 10.000 reservations altogether.
- ◊ The solution should follow a client-server architecture.

• Design and architecture

- Design and architecture



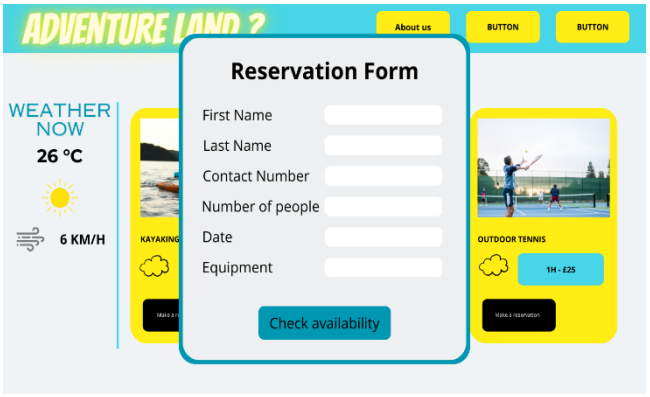
This diagram below shows the flow of the typical use case scenario. First the customer is presented with a display of all activities. After choosing one of the activities, the customer puts personal data into a form, chooses date and time. Then the request to the weather API is made and the customer is presented with a weather forecast for the chosen date. After receiving weather data, a customer can either proceed with making a reservation or close the form and go back to browsing the activities. If the weather is satisfactory and the customer would like to make a reservation, a request to the database will be made. Finally, the customer will be presented with the outcome of that request.



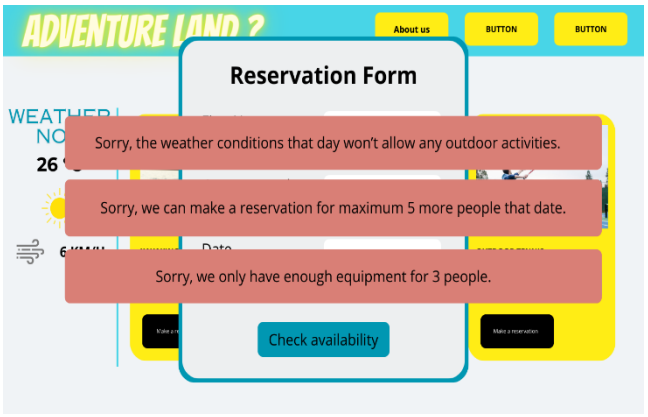
Our application will be divided into three parts: the frontend customer side, the console application client side and the backend service.

To fulfil functional requirements, we prepared a sketch of a customer interface visual design. The first picture presents the proposition of what a customer could see right after visiting the website: with activities, their photographs and a “make a reservation” button.

The second visual represents the reservation form that a customer would have to complete to make a reservation. The form input field validation is a valuable part of the web design.



The third picture is a proposition of how a customer could be presented with an alert, if for any reason, the reservation couldn’t be made.



To meet the functional requirement to present the customer with a weather forecast for the chosen day we will use a publicly accessible weather API. After checking if the chosen activity is a water activity or a different outdoor activity, we will check the wind speed, precipitation probability, temperature and severe risks.

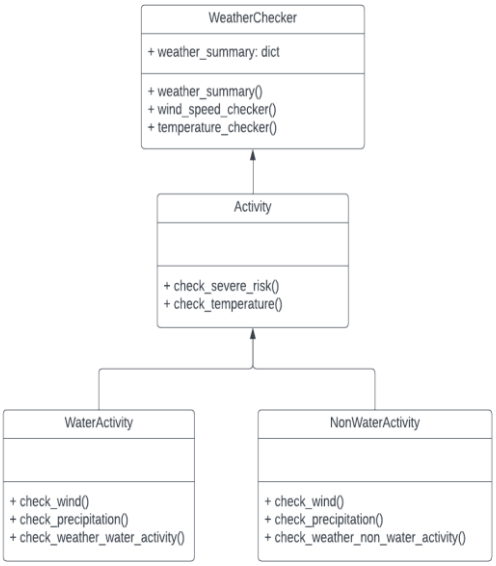
The backend part of the project will have following endpoints:

HTTP Verbs	Endpoints	Functionality
GET	/thrilltopia	the welcome screen
GET	/thrilltopia/activities	displaying all activities
GET	/thrilltopia/activities/indoor	displaying all indoor activities
GET	/thrilltopia/activities/outdoor	displaying all outdoor activities
GET	/thrilltopia/activities/price/asc	displaying all activities in ascending order
GET	/thrilltopia/activities/price/desc	displaying all activities in descending order
GET	/thrilltopia/activities/<id>	displaying activity with given id
GET	/thrilltopia/activities/name/<activity_name>	displaying activity with given name
GET	/thrilltopia/reservations	displaying all reservations
GET	/thrilltopia/reservations/date/<date>	displaying all reservations made on a selected date
GET	/thrilltopia/reservations/name/<last_name>	displaying all reservations made on a person with the given last name
GET	/thrilltopia/reservations/<activity_name>	displaying all reservations made for a given activity
POST	/thrilltopia/reservations	adding a new reservation
PATCH	/thrilltopia/reservations/update/<date>/<id>	updating a reservation date
DELETE	/thrilltopia/reservation/cancel/<id>	deleting a reservation of given id
GET	/thrilltopia/ equipment	displaying all the equipment
GET	/thrilltopia/activities/equipment/<activity_name>	displaying the equipment needed for a given activity
GET	/thrilltopia/weather/<activity_id>/<date>/<start_time>	getting the weather forecast for a selected activity, date and time

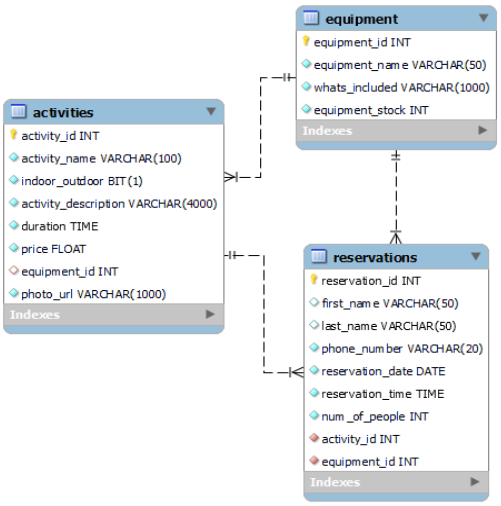
To support the operational tasks related to the use of the system we will prepare a console application. It will have following functionalities:

- ◊ displaying all activities, displaying all indoor or outdoor activities, sorting activities by price ascending and descending, searching activities by id and name.
- ◊ displaying all reservations, searching the reservations by date, customers last name and activity, adding new reservation, updating the reservation date, deleting a reservation.
- ◊ displaying all equipment and equipment for given activity.

Architecture of validating weather condition solution can be presented in the following UML diagram:



All the data provided by the client and added by customers will be stored in a database, which can be visualised by the entity relationship diagram below:



IMPLEMENTATION AND EXECUTION:

The team used an agile approach to project development. We initially brainstormed as a group and put our project ideas, and its elements, onto a Miro board. Here we started to divide up the different tasks. We also made use of a JIRA kanban board where we distributed the workload amongst the team and kept track of the progress of each task. We started by creating major stories on the backlog and issues were linked to allow team members to assign themselves to stories that they will undertake. Epics were created for the main area of the project design, then issues relating to these were created. All were given a priority order as some things needed to be done before others could be started, for example, the database needed to be created before we could create and test any functions. This system allowed team members to pick tasks based on priority and their own workload, giving us a clear view of how our project was progressing.

We have specific rules for our branch naming conventions to keep the project organised and make it easy to share tasks amongst team members. On the JIRA board, once a task has been executed it is moved to 'in review' status, allowing other team members to see that it needs to be reviewed on GitHub before the task can be closed.

• Development approach and team member roles

To keep our project running smoothly, we used a variety of tools designed specifically to make our work easier and more organised. At the start, we relied on a Miro board for teamwork, brainstorming, and sketching out our project plans. It was our go-to space for discussing ideas, creating visual representations, and mapping out the project's concepts and designs. Afterwards we relied on Jira as our main tool for managing the project. It helped us organise our tasks effectively through a board that displayed different stages like To-Do, In Progress, In Review, and Done. Tasks were divided into stories and added to our backlog. This system allowed team members to pick tasks based on priority and their own workload, giving us a clear view of how our project was progressing.

We used GitHub as our go-to system for managing different versions of our code and storing our project files. The code was organized by creating separate branches for each new feature or task. Before combining these changes into the main development area, known as the 'dev' branch, we made sure to review the code thoroughly. This review process, done through pull requests, helped us maintain high-quality code. Moreover, GitHub integration with Jira allowed us to easily track all the changes we made and link them to the specific tasks they were related to. Additionally, Slack was our primary communication channel, allowing real-time interaction among team members. It served as a platform for planned team calls, urgent communications, support requests, and general updates. We also used google meet and held regular meetings inline with our objectives and print goals and duration.

In addition to those project management tools, we also relied on an Excel-based activity log. This log helped us keep detailed records of specific tasks, milestones, and how our progress was coming along.

- Team Members Roles

Each team member played a significant role in contributing to the project, from brainstorming ideas and setting up essential project structures to actively participating in meetings, code development, and documentation. Their collective efforts were crucial in the successful execution of the Thrilltopia project. Everyone engaged in group meetings, providing valuable insights and support.

Inês:

⇒ Created the activity log Excel file.
⇒ Executed the Bootstrap grid and styling for the project's frontend.
⇒ Engaged in API endpoint creation.
⇒ Conducted unit testing for system functionality, ensuring robustness and reliability.

Kayla

⇒ Created the architecture diagram.
⇒ Engaged in API endpoints creation.
⇒ Contributed to unit testing for ensuring the functionality and reliability of the system's components.

Marnie

⇒ Engaged in API endpoint creation.
⇒ Contributed to unit testing for ensuring the functionality and reliability of the system's components.

Paula:

⇒ Led the setup phase, established communication channels, and created the Jira board.
⇒ Contributed significantly to frontend development, including creating forms, working on the user interface, and collaborating on reservation features.
⇒ Engaged in API endpoint creation.

Sofia

⇒ Engaged in API endpoint creation.
⇒ Played a key role in understanding and implementing the weather API, creating relevant functions to retrieve weather data and analyse it.

Vicki

⇒ Worked extensively on the database development including database creation and data population
⇒ contributed significantly in developing the backend files containing the API endpoints, functions and error handling
⇒ Played a key role in implementing the functions and routes in the main file, also the run function to run the application in the console.

• Tools and libraries

Tool Used	Purpose	Choice Justification
Jira	Project Management	Jira allowed us to build a backlog of work that needed to be completed to reach our team goals. It allowed us to see our individual and group workflow throughout the project so that we could see where we were up to at any point. It allows for ownership and accountability for individual tasks and has Github integration allowing us to track work cohesively.
Miro	Project Management	This was used to brainstorm ideas and create the basis of our project posting ideas, design, activities and routes all in 1 place. It was a great tool for interactive communication throughout the project.
GitHub	Project Management	This allowed us to create a team repository that supported collaborative working. It allows for version control meaning that more than 1 person could be working on the project at any time on their own branch which also allowed for any changes in direction

		as new code could easily be reverted to its old format. Finally it allowed for pull requests to be created which meant that team members could review each other's code before merging into the main branch.
Slack	Project Management	This was used as our primary method of communication, channels were always open meaning that there was almost always someone at the end of a message for help if needed. We also arranged all meetings via this platform.
Google Meet	Project Management	This platform was used for our team meetings as it is free and easily accessible to us all.
Pycharm	Application Building	This is one of the code editors that was used by some members of our team. It is a familiar code editor with features such as highlighting, coloured syntax and auto complete which helps with readability and understanding which were important due to the time constraints of the project.
VSCode	Application Building	This is one of the code editors that was used by some members of our team. It is a familiar code editor with features such as highlighting, coloured syntax and auto complete which helps with readability and understanding which were important due to the time constraints of the project.
MySQL Workbench	Application Building	This was the unified visual tool that was used to design and develop our database.
Python	Application Building	This was the main programming language used for the backend development of our project
JavaScript	Application Building	This was one of the programming languages used for the frontend development of our project.
HTML	Application Building	This was one of the programming languages used for the frontend development of our project.
CSS	Application Building	This was the style sheet language used to define the presentation of HTML elements of our project.
SQL	Application Building	This was the programming language used to create our database in MySQLWorkbench.
Bootstrap	Application Building	This was used in conjunction with HTML, CSS and JavaScript to create display aspects of the frontend of our project
Flask	Application Building	This is a Web Server Gateway Interface for python(WSGI framework) that is used for developing web applications. It allows web servers to pass requests to web applications or frameworks.
Flask-CORS	Application Building	This is an extension for flask to handle Cross-Origin resource Sharing(CORS), enabling cross-origin requests.
requests	Application Building	This is a python library used for making HTTP requests and handling responses.
datetime	Application Building	This is a built-in module used for handling date and time related operations.
json	Application Building	This is a built-in module for handling JSON data.
unittest	Application Building	This is a testing framework for writing and running unit tests in python

mysql-connector	Application Building	This is a library used to connect to a MySQL database.
Collections (OrderedDict)	Application Building	This is a module used for implementing specialised data types.

• Implementation process (achievements, challenges, decision to change something)

Due to work commitments, it has been challenging for all members of the team to be on the same call, as a result, instead of having one weekly meeting, we tend to hold 2 calls at different time within the week so that members can join when suitable. The team has also been utilising Slack effectively to communicate and update all team members of any project changes, or difficulties we encountered when working on tasks individually.

Initially, protections had been set up at the review stage of pull requests on GitHub, to ensure each pull request had to be reviewed and then approved by someone else in the team before it could be merged. However, in week 4 of the project, challenges started to arise whereby most of the team could no longer review someone's pull request. For some of the team these challenges were overcome when switching to different web browsers (e.g. safari to chrome). Nonetheless, the challenges kept arising limiting who could review and approve pull requests. We decided to remove these protections due to time constraints, ensuring we could get the project completed in a timely fashion.

• Agile development

The team followed an agile approach throughout this project. We planned the work to be completed on each iteration based on the Jira board that was created. We had meetings every 3 to 4 days giving us clear deadlines of when we would like work to be completed by, any issues that arose in the interim were communicated to the rest of the team on slack. This process allowed us to keep the project moving forward, resolving issues as we came across them rather than having large unproductive gaps in the workflow. Working this way also made tasks more achievable and less overwhelming, also minimising conflicts with other areas of the project.

Case 1 - API implementation

At the beginning stage of setting up the weather API, the codes implemented would allow us to retrieve all the available information from the API and that provided us more than 15 pieces of available information to develop our functions. Members then made use of that information to build up various functions that would display the relevant information when a user interacts with the application.

However, the team then realised not all the information fetched from the API would be relevant or useful for the users to decide if they would like to proceed to book the activities. The team then made use of the Miro boards, discussed on team meeting to brainstorm and redefine the criteria that would impact the users' decision and members then refactored the codes and functions accordingly to ensure only the essential codes should be kept in the application and the irrelevant ones should be removed, i.e., UV index.

Case 2 - GitHub rule

Initially, we added in a rule on GitHub to ensure each pull request is reviewed by another team member prior to it being merged into the dev branch. However, in the last week of the project, the group had been having difficulties reviewing pull requests on GitHub, a decision has been made to lift the rule so members can just merge their own codes to the dev branch after pushing to GitHub given that members should have tested the codes in their locals.

Although on occasions when team members assisted in correcting codes, the codes would still be reviewed before merging, for learning purposes. For instance, member B had identified errors in running the cancellation reservation endpoint on browser and console, member A had communicated on Slack that part of the code (to be run on console) had been refactored however there were still bugs in the remaining part (code to be run on browser). Then member B went on and picked up the task for reviewing and refactoring the code (for browser). Once the review and fix had been completed, member B had communicated on Slack to let the other members

including the original code writer - member C know a pull request had been raised but not yet merged, so that members could go in and review the correct codes as a learning opportunity.

• Implementation challenges

The implementation of our project faced a few challenges that impacted some of our initial plans and required us to adapt our approach.
Initially the group planned to utilise the SQLAlchemy Object Relational Mapper to connect with our SQL database table with our python classes, in the purpose of we were trying to use the open-source SQL toolkit, SQLAlchemy, as the object-relational mapper for our project, which will allow the mapping for the database. Due to time and resource constraints, we revisited the project requirements and decided to use SQL queries to gather our data.

One of the difficulties we encountered was handling merging conflicts within pull requests. Coordinating contributions from multiple team members led to conflicts in the code that required careful resolution to ensure no information would get lost in the way. The introduction of Jira, although a powerful project management tool, presented a learning curve for the team members unfamiliar with its functionalities. Adjusting to this new platform presented initial challenges in organising and managing tasks effectively.

Dependencies among team members' tasks occasionally led to waiting periods, where progress on certain parts of the project relied on the completion of work by others. For example, without the database being created it was difficult to start other parts of the project.

Another significant challenge was the time constraints. Balancing project commitments with individual work schedules posed some difficulties, limiting the time available for learning and implementing everything we had initially thought.
We also had some setbacks such as Inês experiencing a failure in her personal computer right at the end of the project.

Even though the project faced various challenges, our team adapted and found alternative solutions to ensure progress toward achieving our project goals.

TESTING AND EVALUATION:

• Testing strategy

To make sure our code is functioning as expected, we designed a testing strategy including functional and user testing, going through back end, front end and the connection to the database. We focused on testing small, isolated parts of the codebase, ensuring that each performs as intended so we avoid bigger problems at the end of the projects as well as conducting end-to-end tests to simulate real-world user scenarios from start to finish. Through end-to-end tests, we're ensuring that the system functions smoothly across different user interactions and interfaces.

• Functional and user testing

Starting with the creation of the database and connection to our backend, we used print statements with our queries to make sure we were connecting and getting the information from the database. Followed by the creation of the API endpoints, we used print statements inside the functions while constructing these and at the endpoint we checked the browser and compared with the database to evaluate if the endpoints were correct. For the functions used for the endpoints we used unit testing using the *unittest* module to ensure the reliability of functions and consequently, the endpoints. In these unit tests we focused on valid but also invalid inputs to make sure the functions for the endpoints were correct in case the user used right or wrong inputs. We have also covered some boundary tests using the date 0000 to assert that the functions run smoothly.

Regarding the weather API we tested the respective endpoint with different dates and times to verify that we were getting the correct information. Also, while coding the functions for the weather we used print statements before continuing to catch errors at an early stage. Concerning the API key we have tested using different keys with different members of the group to make sure that the API call was working.

When it comes to the front end, we also conducted end-to-end tests to simulate real-world user scenarios from start to finish. This involved testing the entire user flow, from browsing activities and selecting a date to completing a reservation. Through end-to-end tests, we have ensured that the system functions smoothly across different user interactions and interfaces. We have done integrated front end testing, by having different inputs in the activity's form, with different dates, times, emails, people joining the activity and made sure that each field hade the correct format imputed as well as the minimum and maximum date be between today and 14 more days, which is the availability of the weather API. For the pop-ups of the activity form we tested adding different activities in different days, and therefore different weather, to make sure we were getting the correct warning messages according to weather that day.

• System limitations

For the unit tests of the functions that were used in the endpoints it was difficult to get boundary tests due to the complexity of the functions. For the weather API we had to test with dates up to 14 days from now due to the weather API availability.

CONCLUSION:

Throughout this project we have aimed to create and develop a reservation system for an imaginary activity centre, 'Thrilltopia', based in Lisbon. The customer will be able to browse through a variety of indoor and outdoor activities, where they will be presented with predicted weather over a 15-day period. The app uses a Bootstrap framework to create the layout of the front-end, combining user data alongside a publicly accessible weather API to return information in a readily available, easy format. In the future we would look to create a mobile notification for the user, reminding them of their reservation 24 hours before their booking.

We believe Hack Street Girls have achieved their aim successfully due to the efficient working of the application, alongside the team's commitment to work cohesively as a group under Agile principles, despite challenges that arose.