

comp90082-2024-sg-koala	4
BACKGROUND	10
Product Overview	11
Stakeholders	12
Client Expectation	14
Industry Need	15
Scopes	16
Terminology	18
Choice of Technology	19
Constraints	20
Ethical Consideration	21
Cyber Security Consideration	22
Deployment Decision	23
ROADMAP	25
High-Level Plan	26
SPRINT-1	27
Sprint-1 Goal	28
Sprint-1 Burn Down Chart	29
SPRINT-2	30
Sprint-2 Goal	31
Sprint-2 Burn Down Chart	32
SPRINT-3	33
Sprint-3 Goal	34
Sprint-3 Burn Down Chart	35
SPRINT-4	37
Sprint-4 Goal	38
Sprint-4 Burn Down Chart	39
REQUIREMENTS	40
Goal Model	41
Personas	42
Prototypes	44
User Stories - Product Backlog	45
Provided Information - User Stories Support Details	54
RISK	56
Risk Registry	57
Review Meetings	58
COLLABORATION	59
About Us	60
Team Calendar	61
Agile Processes	62
Team Charter	63
Definition of Ready	65
Definition of Done	66
Agile Ceremonies	67
Sprint Planning - 1	68

Retrospective - 1	70
Review Meeting - 1	72
Sprint Planning - 2	74
Review Meeting - 2	76
Retrospective - 2	78
Sprint Planning - 3	80
Retrospective - 3	82
Review Meeting - 3	84
Sprint Panning - 4	85
Retrospective - 4	87
Review Meeting - 4	89
ARCHITECTURE	90
Database Model	91
API	93
Analytics	107
TESTS	110
Backend - Testing	112
Unit Tests - GET	114
Unit Tests - PUT	115
Unit Tests - DELETE	116
Unit Tests - CREATE	117
Unit Tests - Search	118
Unit Tests - Miscellaneous DB Functions	119
Unit Tests - Charts	121
Frontend - Testing	123
Unit Test - Dashboard	124
Unit Test - NewBooking	126
Unit Test - Templates	127
Unit Test - TemplateDetail	128
Unit Test - Calendar	129
CODE REVIEW	130
Frontend Code Review Sprint #2	131
Backend Code Review Sprint #2	137
Front-end check list before doing the merge request	139
Back-end check list before doing the merge request	140
AI Code Review	141
Merge Request Feedback 1	142
Merge Request Feedback 2	157
Merge Request Feedback 3	165
Merge Request Feedback 4	171
Merge Request Feedback 5	173
Merge Request Feedback 6	175
Merge Request Feedback 7	177
Merge Request Feedback 8	179
Merge Requests Feedback 9	183

Frontend Code Review Sprint #3	185
Backend Code Review Sprint #3	187
TEMPLATE	188
Template - Decision documentation	189
Template - Product requirements	190
Template - Meeting notes	192
LINKS & RESOURCES	193
PRESENTATION	194
PRODUCT USER MANUAL	195
SPRINT2 DEMO	196
FINAL DEMO	198



comp90082-2024-sg-koala

About

The project background for the CIS Booking System revolves around the needs of the Museums & Collections (M&C) Learning Team at the University Of Melbourne.

The existing booking system faces challenges due to the complexity of managing multiple venues, varying spaces within these venues, different staff rosters, diverse booking processes, and invoicing methods. Schools show interest in these programs through various channels, including direct negotiations, online bookings for self-guided tours, Microsoft Forms for workshops, and general inquiries via phone or email.

The process involves multiple steps such as recording requests, checking venue availability, managing staff schedules, confirming details with schools, booking buses if needed, issuing invoices for non-partner schools, and handling additional logistical requests. The system currently utilizes tools like Planner, Priava, Microsoft Calendar, and Excel to manage these tasks, but it lacks integration and automation, leading to potential inefficiencies and inaccuracies in booking management.

The intent to upgrade to a new system arises from the need for a more cohesive and efficient approach to managing these complexities. The new system aims to streamline the booking process, enhance organizational efficiency, and improve the overall management of educational excursions by addressing the existing system's limitations and incorporating a more integrated, user-friendly solution.

Status

We are working on this!

[Expand all](#) [Collapse all](#)

- ✓ **BACKGROUND**
 - * Product Overview
 - * Stakeholders
 - * Client Expectation
 - * Industry Need
 - * Scopes
 - * Terminology
 - * Choice of Technology
 - * Constraints
 - * Ethical Consideration
 - * Cyber Security Consideration
 - * Deployment Decision

- ✓ **ROADMAP**
 - * High-Level Plan
- ✓ **SPRINT-1**
 - * Sprint-1 Goal
 - * Sprint-1 Burn Down Chart
- ✓ **SPRINT-2**
 - * Sprint-2 Goal
 - * Sprint-2 Burn Down Chart
- ✓ **SPRINT-3**
 - * Sprint-3 Goal
 - * Sprint-3 Burn Down Chart
- ✓ **SPRINT-4**
 - * Sprint-4 Goal
 - * Sprint-4 Burn Down Chart

- ✓ **REQUIREMENTS**
 - * Goal Model
 - * Personas
 - * Prototypes
 - * User Stories - Product Backlog
 - * Provided Information - User Stories Support Details

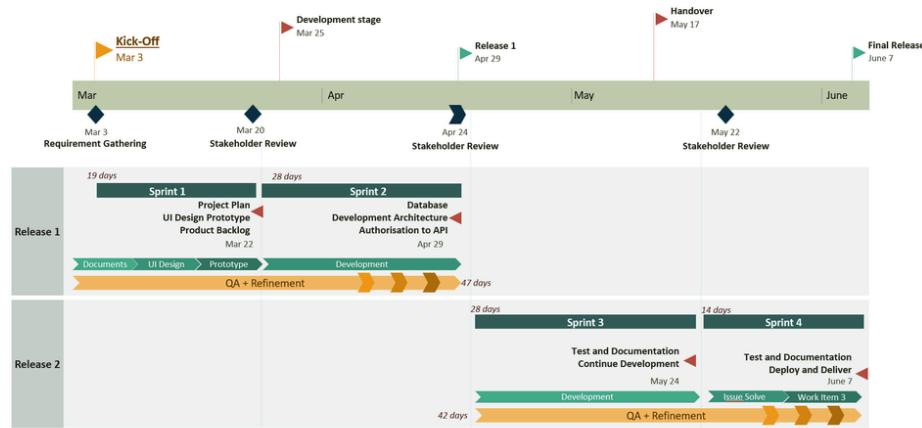
- ✓ **RISK**
 - * Risk Registry
 - * Review Meetings

- ✓ **MEETINGS**
 - * The 1st Client Meeting
- ✓ **Team Meetings**
 - * Team Meeting - 0
 - * Team Meeting - 1
 - * Team Meeting - 2
 - * Team Meeting - 3
 - * Team Meeting - 4
- ✓ **Supervisor Meetings**
 - * Supervisor Meeting - 0
 - * Supervisor Meeting - 1
 - * Supervisor Meeting - 2
 - * Supervisor Meeting - 3
 - * Supervisor Meeting - 4
 - * Supervisor Meeting - 5
 - * Supervisor Meeting - 6
 - * Supervisor Meeting - 7
 - * Supervisor Meeting - 8
 - * Supervisor Meeting - 9
 - * Supervisor Meeting - 10

- ✓ **COLLABORATION**
 - * About Us
 - * Team Calendar
- ✓ **Agile Processes**
 - * Team Charter
 - * Definition of Ready
 - * Definition of Done
- ✓ **Agile Ceremonies**
 - * Sprint Planning - 1
 - * Retrospective - 1
 - * Review Meeting - 1
 - * Sprint Planning - 2
 - * Review Meeting - 2
 - * Retrospective - 2
 - * Sprint Planning - 3

- * Retrospective - 3
- * Review Meeting - 3
- * Sprint Planning - 4
- * Retrospective - 4
- * Review Meeting - 4
- ▽ ARCHITECTURE
 - * Database Model
 - * API
 - * Analytics
- ▽ TESTS
 - ▽ Backend - Testing
 - * Unit Tests - GET
 - * Unit Tests - PUT
 - * Unit Tests - DELETE
 - * Unit Tests - CREATE
 - * Unit Tests - Search
 - * Unit Tests - Miscellaneous DB Functions
 - * Unit Tests - Charts
 - ▽ Frontend - Testing
 - * Unit Test - Dashboard
 - * Unit Test - NewBooking
 - * Unit Test - Templates
 - * Unit Test - TemplateDetail
 - * Unit Test - Calendar
- ▽ CODE REVIEW
 - * Frontend Code Review Sprint #2
 - * Backend Code Review Sprint #2
 - * Front-end check list before doing the merge request
 - * Back-end check list before doing the merge request
 - ▽ AI Code Review
 - * Merge Request Feedback 1
 - * Merge Request Feedback 2
 - * Merge Request Feedback 3
 - * Merge Request Feedback 4
 - * Merge Request Feedback 5
 - * Merge Request Feedback 6
 - * Merge Request Feedback 7
 - * Merge Request Feedback 8
 - * Merge Requests Feedback 9
 - * Frontend Code Review Sprint #3
 - * Backend Code Review Sprint #3
- ▽ TEMPLATE
 - * Template - Decision documentation
 - * Template - Product requirements
 - * Template - Meeting notes
- ▽ LINKS & RESOURCES
- ▽ PRESENTATION
- * PRODUCT USER MANUAL
- * SPRINT2 DEMO
- * FINAL DEMO

Roadmap





BACKGROUND

1. Product Overview:

- The **core functionality** and **purpose** of the product.
- How the product **addresses specific problems**.
- **Key features** of the product.

2. Stakeholders:

- Lists **key stakeholders** involved in the project, including both internal and external parties.
- Describes the **role, level of influence, impact** and **communication plan** of each stakeholder on the project.

3. Client Expectation:

- Specific **requirements** and **expectations** of the client regarding the product.

4. Industry Need:

- How the **product fits** into **specific client needs**.

5. Scopes:

- Defines the **scope of the project**.
- Project **objectives** and **functionalities**.

6. Terminology:

- Professional **terms** and **definitions** used in the project, and product.

7. Choice of Technology:

- Reasons for selecting **specific technologies or platforms**, including software, hardware, development tools, and frameworks.
- Technical **dependencies** and integration needs.

8. Constraints:

- Limitations faced by the project.

9. Ethical Consideration:

- Consideration of **issues and decisions** in the project

10. Cyber Security Consideration:

- Consideration of **security issues** in the project

11. Deployment Decision:

- **Deployment decision** of backend and frontend

Product Overview

Introduction:

Science Gallery Melbourne introduces multiple unique tours and workshops for secondary school group visits. With a high volume of booking requests throughout the year, the current use of multiple systems including Microsoft Office tools and Priava for space and time management has become cumbersome.

The primary challenge is the complexity and inefficiency of using multiple systems to manage bookings. This multi-step process requires constant updates across several platforms when any changes to bookings are made, involving a significant administrative effort and coordination among team members.

We seek to develop a Unified Booking Management System that simplifies and automates the booking process. This solution will integrate booking, calendar scheduling, and space and equipment management into a single, user-friendly platform. It will help the staff of the Learning Team add and update bookings quickly, check for any scheduling conflicts and store all the tours to the database automatically.

Key Features:

1. **Unified Booking System:** A centralised platform that integrates booking requests, calendar management, and resource allocation, ensuring a seamless process from initial inquiry to final confirmation.
2. **Dynamic Scheduling & Resource Management:** Checks for space, staff availability, and equipment requirements to prevent double bookings and ensure that all necessary resources are allocated to each event.
3. **Bulk Editing & Updating:** Offers the capability to modify and update multiple bookings simultaneously, facilitating easy adjustments to recurring events or changes in program schedules.
4. **Comprehensive Database Integration:** Confirmed and completed bookings are systematically recorded in a database, enabling efficient tracking, reporting, and analysis of past events for continuous improvement and planning.
5. **Interactive Weekly Dashboard:** Provides a real-time overview of upcoming programs, including detailed information on the schedule, assigned staff, and preparation requirements, aiding in efficient staff rostering and program delivery.

Benefits:

Efficiency: Automates and simplifies the booking and event management process, saving time and reducing manual errors.

Flexibility: Easily accommodates various program types, booking requirements, and resource constraints.

Reduce Human Error: Minimises the likelihood of mistakes in booking arrangements and resource allocations, ensuring accuracy and reliability in every event organised.

Stakeholders

Stakeholders	Description	Level of Interest	Level of Influence	potential Impact	Communication Plan
Science Gallery Learning Team	Main end user of the product, responsible for organising and managing educational programs and workshops at the Science Gallery Melbourne.	High, as the product directly affects their work processes and efficiency.	High, given their direct involvement in defining requirements and using the system.	High, as improvements in the system can greatly enhance their operational efficiency and program quality.	In person meeting after each sprint.
Partner/Non-partner school teachers	Teachers from schools that collaborate with the gallery for educational tours and workshops.	Low	Low	Low, their booking process will not change.	N/A
Project team	The group of individuals tasked with developing and implementing the project.	High, as their role is to deliver a successful product.	Very high, as they make the crucial decisions on the system's features and functionality.	High, as the project's success directly correlates with their performance.	Meet once or twice a week in person or through zoom.
University of Melbourne	The university associated with the Science Gallery Melbourne, potentially providing resources or support.	Moderate	High, the booking process need to use specific tool.	Moderate	N/A
Learning team temporary employee	Short-term staff hired to assist with the increased workload or specific tasks within the Learning Team.	Moderate	Low	Moderate, in terms of testing and providing feedback on the system's usability and efficiency.	N/A
Bus company BusCharter	The service provider responsible for transporting students to and from the gallery.	Low	Low	Low	N/A
Interpreters	The service provider responsible for translating for students	Low	Low	Low	N/A

Unimelb business service	The business services division within the University of Melbourne that responsible for paying additional fees (bus and interpreters).	Moderate , as they need accurate information sent from the system	Moderate	Moderate	N/A
Secondary school students	Students that participate in tours and workshops.	Low	Low	Low	N/A
IT department	The team responsible for maintaining the technological infrastructure.	High, as they need to ensure the system's reliability and security.	High, due to their role in troubleshooting, maintenance, and potentially implementing technical improvements.	High, as the system's functionality and user experience heavily depend on their efficiency and effectiveness in addressing technical issues.	Communicate using email.

Client Expectation

1. **Unified Booking System:** A centralised platform that integrates booking requests, calendar management, and resource allocation, ensuring a seamless process from initial inquiry to final confirmation.
2. **Dynamic Scheduling & Resource Management:** Automatically checks for staff availability, and equipment requirements to prevent double bookings and ensure that all necessary resources are allocated to each event.
3. **Bulk Editing & Updating:** Offers the capability to modify and update multiple bookings simultaneously, facilitating easy adjustments to recurring events or changes in program schedules.
4. **Comprehensive Database Integration:** Confirmed and completed bookings are systematically recorded in a database, enabling efficient tracking, reporting, and analysis of past events for continuous improvement and planning.
5. **Interactive Weekly Dashboard:** Provides a real-time overview of upcoming programs, including detailed information on the schedule, assigned staff, and preparation requirements, aiding in efficient staff rostering and program delivery.
6. **User-Friendly Interface:** The system should be intuitive and easy to navigate for all users.

Industry Need

The project, while focused on enhancing the booking and event management processes for the Science Gallery Melbourne, taps into a broader industry need within large educational and cultural institutions like the University of Melbourne. The specific challenges related to efficiency in such expansive organisations underscore a critical industry demand. These challenges are often compounded by the size and complexity of the organisation, leading to inefficiencies in communication, coordination, and resource management.

Scopes

Inner Scope	Form for new bookings	<p>Objectives: A form for staff to enter booking details when clients book via phone or email.</p>
		<p>Functionality: The form should contain all the necessary details and specific request. A booking ID will be sent to client email after form is completed.</p>
	Tag pending application	<p>Objectives: Tag pending application depends on the programs applied.</p>
		<p>Functionality: Staff will be able to tag pending application to generate checklist using different template.</p>
	Checklist	<p>Objectives: A checklist shows following steps that need to be complete.</p>
		<p>Functionality: Staff will use the checklist to ensure all necessary steps are taken, such as availability confirmation, payment processing, final confirmation with clients.</p>
	Calendar	<p>Objectives: A calendar shows all the confirmed bookings.</p>
		<p>Functionality: Confirmed bookings will be visible on calendar, with accompanying staff and number of students.</p>
	Database	<p>Objectives: A database will store all details of the bookings.</p>
		<p>Functionality: Staff will be able to add or edit new data. Generates analytic reports for future reporting.</p>
Outer scope	Secondary school application form	<p>Partner/Non-partner school teacher apply tour/workshop by filling online form.</p>

	Priava venue	Staffs check and book venue on Priava.
	Microsoft Calendar events	Add or update events on Microsoft Calendar, accompanying teacher will be assign to events.
	Payments	Pay necessary fees such as bus or interpreter fees.

Terminology

Term	Definition
Booking System	A digital platform used to manage reservations, appointments, or event participations, allowing users to schedule visits or services in advance.
Resource Allocation	The process of assigning available resources in an efficient, effective, and orderly manner. In the context of this project, resources can include staff, rooms/spaces, equipment, and workshop materials.
Dashboard	A visual interface that displays key information and metrics at a glance, allowing users to monitor and manage activities, resources, or performance indicators effectively.
Database	A structured set of data held in a computer, especially one that is accessible in various ways.

Choice of Technology

Choice of Technology	
Database	MongoDB
Front-End	React Framework
Back-End	Django Python

Client Requirement : Any technology is feasible, as long as it does not affect priava's calendar.

- **Database:** MongoDB

- what: MongoDB is an open-source NoSQL database that stores data in the form of documents (usually in JSON or BSON format). Compared to traditional relational databases (such as MySQL or PostgreSQL), it offers higher flexibility and scalability.
- why: For this project, given the limited budget, using the free version of MongoDB is a relatively good choice, as it can largely meet our requirements.

- **Front-End:** React Framework

- what: React is an open source plug-in JavaScript library for building user interfaces. Since It has become one of the most popular and widely used tools in plugin development. The main goal of React is to improve development efficiency, enable fast UI rendering, and help developers build large, dynamic web applications.
- why: Our team members are essentially experienced with React, which makes it very easy for us to get started. We can begin developing our website earlier, saving some time for learning and adaptation.

- **Back-End:** Django Python

- what: Django is an advanced Python web framework designed to make developing complex, database-driven websites easier and faster.
- why: Django's comprehensiveness can cover all our needs, and some of our team members also have rich experience using it, so we choose to use Django.

Constraints

- This product is required to use the Priava booking system for venue booking
- This product is required to use Outlook Calendar to invite facilitators and manage booking calendar
- Zero budget on deployment, database

Ethical Consideration

Privacy and Data Security:

- **Personal Information:** Ensuring that personal information provided by users is protected from unauthorised access and misuse is crucial. The system should comply with relevant data protection laws and use secure data encryption methods.
- **Access Controls:** Ensure that only authorised staff have access to sensitive customer data and that access levels are appropriately assigned based on job roles.
- **Data Handling:** Training staff on secure data handling practices is essential, as human error is a common security vulnerability. Additionally, the organisation should establish a robust data breach response plan to quickly address and mitigate the impact of data leaks.

Transparency and Honesty:

- **Error Reporting:** Systems should be transparent in terms of functionality and reporting, enabling staff to easily report any issues or inaccuracies in bookings. This system should be easy to use and integrated into the daily operations, ensuring that errors are corrected before they can do any serious impacts.
- **Communication:** Policies should be in place to ensure that all staff members communicate consistently and openly with customers, especially regarding cancellations, changes, or errors. This transparency builds trust and prevents misunderstandings. Training should emphasise the importance of clear and honest communication, reinforcing the company's commitment to customer service.

User Friendliness and Accessibility

- **Interface Design:** The design of the booking system interface should prioritise simplicity and efficiency, reducing the potential for user error and enhancing the overall user experience for staff such as clearly labeled functions and a logical navigation layout.
- **Accessibility Features:** Consider features that make the system accessible to staff with disabilities, such as screen readers or alternative input methods.

Technical Support

- **Responsive Technical Support:** A dedicated technical support team should be available to address system issues swiftly, minimising downtime and ensuring that the booking process is smooth for staff. This support should be proactive, identifying potential problems before they affect system performance.

Solution: apply these considerations in every pull request and merge and code review to reduce unethical decision.

Cyber Security Consideration

Secure Data Transmission and Storage:

Strong encryption protocols should be implemented for data at rest and in transit. For data in transit, TLS (Transport Layer Security) should be standard to secure communications between users' browsers and the server. For data at rest, robust encryption algorithms should be used to protect it from unauthorised access if the data storage is breached.

Solution: use a secure hosting server (vercel) and database (MongoDB)

Access Controls:

Ensure that only authorised staff have access to sensitive customer data and that access levels are appropriately assigned based on job roles.

Solution: use secured database (MongoDB) and add user authentication.

Security Audits:

Conduct regular security audits to assess the effectiveness of the security measures in place. These audits should check for vulnerabilities in both software and hardware components of the booking system.

Solution: check and update security concerns every sprint

Penetration Testing:

Employ ethical hackers to perform penetration testing. This involves attempting to exploit system vulnerabilities to determine where the system can be fortified. It helps identify weaknesses before they can be exploited by malicious actors.

Solution: check and test the system security every sprint, and document any concerns.

Regular Backups:

Regularly back up data and ensure that backups are stored in a secure location. Backups can be vital in recovering from a data breach, ransomware attack, or other types of data loss incidents.

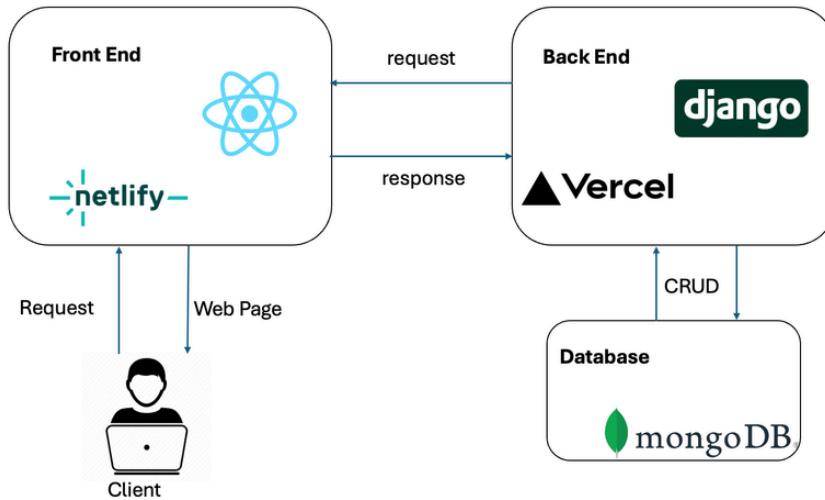
Solution: use Mongodb Database to backup data and user version control (github) for code backup.

Regular Updates:

Implement a strict schedule for updating all software, including the operating system, applications, and all third-party software components used in the booking system. This helps protect against vulnerabilities that could be exploited by attackers.

Solution: update system application with auto script to update any new application or packages.

Deployment Decision



Deployment Tool Options:

1. Netlify:

- **Description:** Netlify is a popular hosting and automation platform for static websites.
- **Pros:** Seamless integration with Git, continuous deployment, serverless functions, and CDN.
- **Cons:** Limited support for backend services compared to full-fledged cloud providers.

2. Vercel:

- **Description:** Vercel is a cloud platform for static sites and serverless functions, with a focus on speed and simplicity.
- **Pros:** Built-in support for serverless functions, CDN, and seamless integration with Git.
- **Cons:** Limited scalability and flexibility compared to full cloud providers like AWS or Azure.

Evaluation Criteria:

- **Ease of Use:** How easy is it to set up and use the tool?
- **Compatibility:** Does the tool support your technology stack and deployment environment?
- **Scalability:** Can the tool handle your current deployment needs and scale with your project?
- **Community Support:** Is there a strong community around the tool for support and updates?
- **Cost:** What is the cost of the tool and is it within your budget?
- **Features:** What specific features does the tool offer that are beneficial for your project?

Decision Criteria:

- **Technical Fit:** Do the tools meet your technical requirements for frontend and backend deployment?
- **Team Expertise:** Does your team have experience or expertise with Netlify and Vercel?
- **Cost-Effectiveness:** Are the costs of using both tools justified by their features and benefits?
- **Scalability:** Can Netlify and Vercel scale to meet your future deployment needs?
- **Support and Maintenance:** Is there adequate support and maintenance available for Netlify and Vercel?

Decision:

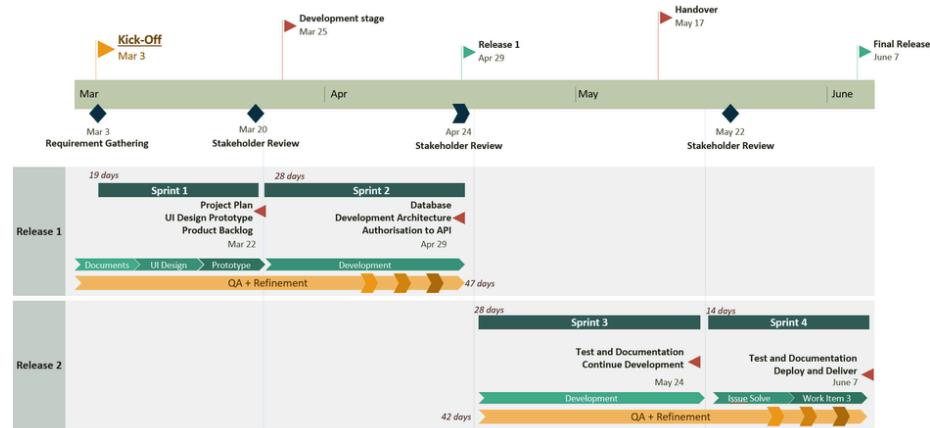
- **Selected Frontend Tool:** Netlify
- **Selected Backend Tool:** Vercel
- **Reasons for Selection:** Both Netlify and Vercel have free tier that the team can use in the development stage

Risks and Mitigation:

- **Risk:** Potential compatibility issues between Netlify and Vercel.
- **Mitigation:** Thorough testing and coordination between frontend and backend teams.

Stakeholder Communication:

- **Communication Plan:** Notify client about the decision to use Netlify and Vercel for frontend and backend hosting.

 ROADMAP


High-Level Plan

Sprint 1 - planning

- design prototype
- determine the choice of technologies
- generate product backlog
- estimate story points and priorities
- allocate user stories for sprint 2
- allocate development roles
- generate risk management strategy

Sprint 2 - development

- client review on sprint 1
- update requirements based on client review
- Initialise development architecture, front end and back end
- design database structure
- enable authorisation to external API
- complete user stories allocated
- monitor risk
- update risk management strategy if detect new risk
- update development speed

sprint 3 - development

- client review on sprint 2
- update requirements based on client review
- continue development on allocated user stories
- update risk management strategy if detect new risk
- test and document completed features
- complete user stories allocated
- update development speed

sprint 4 - deployment

- test and document all features
- client review on sprint 3
- final adjustment based on client review
- deploy and deliver

1 SPRINT-1

Sprint-1 Goal

- collect client information for user stories
- generate product backlog
- generate sprint 2 goals and burn down chart
- generate goal model
- confirm the choice of technologies
- confirm communication frequency
- initialised risk management strategy
- complete design prototype

Sprint-1 Burn Down Chart

Since Sprint 1 is the inception and design phase, there is no need for a burn-down chart of user stories.

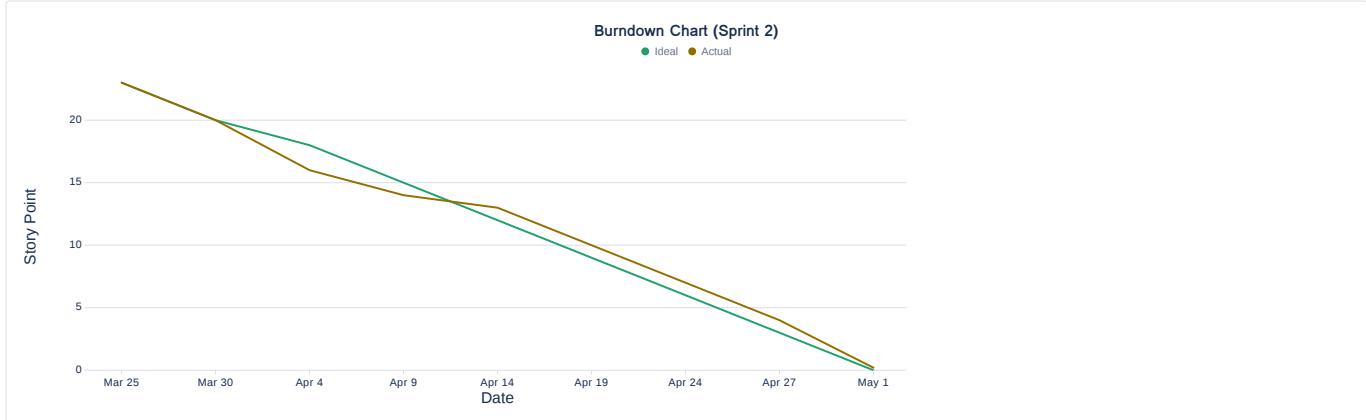
2 SPRINT-2

Sprint-2 Goal

Sprint goal	List of feature user stories
Implement school teacher booking process	<p>US-0.1 see User Stories</p> <p>As Tony Lee, I want to book a workshop through any method listed on the current website so that I can visit the place with students.</p> <p>Epic Feature 1 see User Stories</p> <p>As Emily Zhang, I want to input phone/ email bookings for clients so that I can process the booking.</p>
Implement learning team booking management	<p>Epic Feature 1 see User Stories</p> <p>As James Parker, I want to manage all pending bookings so that I can check with the booking checklist.</p>
Implement connection with Priava venue booking system	<p>Epic Feature 2 see User Stories</p> <p>As James Parker, I want to check a booking time with Priava venue booking so that I can manage the availability of the venue.</p>
Initialise database connection and storage	<p>Epic Feature 4 see User Stories</p> <p>As James Parker, I want to see all the booking information so that I can update the details.</p>

For detailed User Stories assigned to Sprint 2, See [Sprint-2 Burn Down Chart](#)

Sprint-2 Burn Down Chart



Task In Plan	Story Point (Total/Finished)
US 0.1	1 / 1
US 1.1	3 / 1.8
US 1.2	3 / 3
US 1.4	3 / 1.5
US 1.5	3 / 1.5
US 2.1	5 / 5
US 2.2	3 / 3
US 4.1	3 / 1
Total Story Point	23 / 16.8

Other User Story Finished	Story Point (Total/Finished)
US 0.2	3 / 3
US 1.3	3 / 3

Estimate Velocity= 23

Actual Velocity=22.8

US 1.1, 1.5, 4.1 are still under development as the team faces some technical issues, these user stories will be completed in the next sprint.

Task	Date	Ideal	Actual
Start	Mar 25	23	23
	Mar 30	20	20
	Apr 4	18	16
	Apr 9	15	14
	Apr 14	12	13
	Apr 19	9	10
	Apr 24	6	7
	Apr 27	3	4
End	May 1	0	0.2

3 SPRINT-3

Sprint-3 Goal

Sprint goal	List of feature user stories
Finish Epic feature 0	<p>Epic Feature 0, See User Stories</p> <p>As Tony Lee, I want to book a workshop and get a booking ID so that I can update/cancel any booking later through email or call using this booking ID.</p>
Implement Microsoft Calendar management system	<p>Epic Feature 3, See User Stories</p> <p>As James Parker, I want to update booking information to Microsoft Calendar so that I can invite and roster facilitators.</p>
Store and update booking details in Database	<p>Epic Feature 4, See User Stories</p> <p>As James Parker, I want to stored updated booking information in database so that I can use them for analysis.</p>
Enable updated booking details	<p>Epic Feature 1, See User Stories</p> <p>As James Parker, I want to update the current booking detail so that I can have the up to date information for reporting.</p>
Implement accessibility arrangements including interpreter or transportation	<p>Epic Feature 5, See User Stories</p> <p>As James Parker, I want to book additional arrangements so that I can satisfy clients' requirements.</p>
Implement update for risk assessment	<p>Epic Feature 6, See User Stories</p> <p>As James Parker, I want to send and update workshop risk assessment for a school so that they can prepare for this visit and I can keep a record of the risk of this event.</p>
Implement updates for medical information	<p>Epic Feature 7, See User Stories</p> <p>As James Parker, I want to gather and manage medical information, allergies, or accessibility needs for the visiting groups so that I can ensure a safe and accommodating visit for all attendees.</p>

For detailed User Stories assigned to Sprint 3, See [Sprint-3 Burn Down Chart](#)

Sprint-3 Burn Down Chart



Task In Plan	Story Point (Total/Finished)
US 0.2	3
US 3.1	1
US 3.2	3
US 4.3	1
US 5.1	1
US 5.2	3
US 6.1	1
US 6.2	1
US 7.1	3
U.S 4.2	3
U.S 8.1	3
U.S 9.1	1
Task Left from Sprint 2	
US 1.1	1.2
US 1.4	3.5
US 1.5	1.5
US 4.1	2
Total Story Point	31.2

Estimate Velocity= 22.8

Actual Velocity=30.2

Task	Date	Ideal	Actual
Start	May 2	31.2	31.2
	Mar 5	27	28
	Mar 8	23	25
	Mar 11	19	21
	Mar 14	15	17
	Mar 17	11	13
	Mar 20	7	5
	May 22	3	3
End	May 24	0	1

4

SPRINT-4

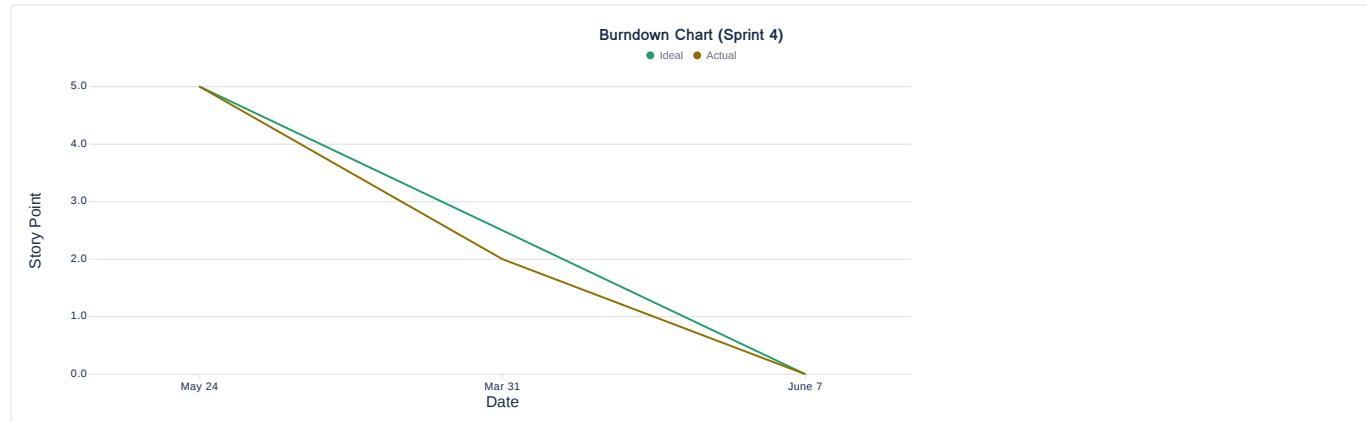
Sprint-4 Goal

Sprint goal	List of feature user stories
Finish U.S 6.2 from	<p>Epic Feature 6, See User Stories</p> <p>As James Parker, I want to confirm booking details with teachers two weeks before their scheduled visit, so that any necessary adjustments can be made on time.</p>

Refine UI design and test functionalities:

- Clear all “console.log” and “print” in code
- Improve user experiences
- Improve UI design
- Ready for Deployment trial

Sprint-4 Burn Down Chart



Task In Plan	Story Point (Total/Finished)
US 6.2	1
Finiliza Code and Refine UI Design	2
Testing	2
Total Story Point	5

Estimate Velocity= 30.2

Actual Velocity=5

Task	Date	Ideal	Actual
Start	May 24	5	5
	Mar 31	2.5	2
End	June 7	0	0



REQUIREMENTS

1. Goal Model:

- Outlines the **main objectives and goals** of the project or product.

2. Personas:

- Represents **fictional characters** created based on the research to represent the **different user types** who might use the product.

3. Prototypes:

- **Early models** or mockups of the product to explore ideas, design concepts, and usability testing.

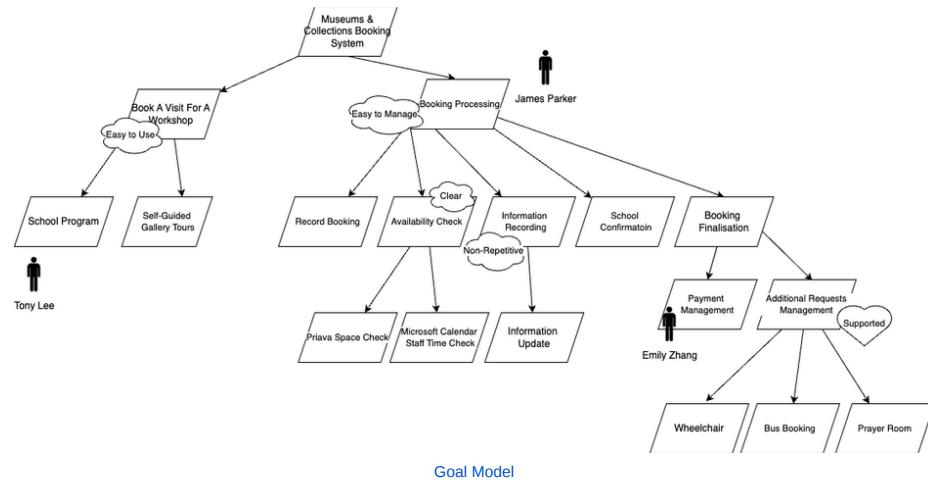
4. User Stories - Product Backlog:

- User stories describe the **features** and **functionality** of the product from the **end user's perspective**.
- The **product backlog** is a **prioritized list of user stories** that provide a comprehensive view of all the work needed for the project.

5. Provided Information - user stories support details:

- Includes **additional details** and information that support the user stories.

Goal Model



Personas



Image:generate By AI

James Parker
Booking manager

Age : 38
Gender : Male
Address : Melbourne

.....

Major Responsibilities

- Managing and overseeing the booking system for various trips within the Science Gallery.
- Ensuring a smooth and efficient booking process for both individual visitors and group tours.
- Coordinating with other departments to update trip schedules, availability, and special events.
- Handling customer inquiries and issues related to bookings.

Physical, Social, and Technological Environment

- Works in an office within the Science Gallery but often moves around the gallery to coordinate with different departments.
- Interacts with a diverse team, including exhibitions coordinators, teacher, and school representatives.
- Uses a variety of technology solutions, including a booking management system, Phava system.

Goals and Tasks

- To simplify the booking process for all users and make sure that it is user-friendly and accessible for all the user.
- To increase the efficiency of managing bookings, including ease of updating trip schedules and availability.
- To improve customer satisfaction.

Quote

"Our main duty is to let the client can book the journey of discovery efficiently and easily. To achieve this goal, my mission is to make the process as seamless as possible, ensuring that every guest are happy with the whole process."

SKILLS

Managing	★★★★★
Problem Solving	★★★★★
Teamwork	★★★★★

PERSONALITY

Personal Thinking	85%
Open Minded	75%



Image:generate By AI

Tony Lee
School Teacher

Age : 38
Gender : Male
Address : Melbourne

.....

Major Responsibilities

- Organize educational outings for students, such as visits to the Science Museum.
- Book a trip to the Science Museum for students of all ages.
- Coordinate with the Science Museum to ensure the itinerary is both educational and fun.
- Manage all trip-related logistics and safety measures.

Physical, Social, and Technological Environment

- Work within the school and communicate frequently with teachers, parents and students.
- Use your school's educational technology resources and external booking platforms to organize your trip.

Goals and Tasks

- Find and book educational Science Gallery tours that inspire students' curiosity and passion for learning.
- Make sure the booking process is quick and easy so that more time can be spent preparing educational content rather than administrative work.

Quote

Our goal is to create a lively learning experience for students through the exploration and understanding of different knowledge.

SKILLS

Teaching	★★★★★
Problem Solving	★★★★★
Teamwork	★★★★★

PERSONALITY

Personal Thinking	85%
Open Minded	75%

42



Image:generate By AI

Major Responsibilities

- Answering phone calls from schools and educators interested in booking educational programs.
- Processing booking requests, including entering data into the booking system, scheduling, and confirming bookings.
- Coordinating with other departments to ensure availability and readiness of programs.

Physical, Social, and Technological Environment

- Works in a busy office environment, part of the larger educational outreach department.
- Regularly interacts with educators, school administrators, and internal staff from various departments.
- Uses a specialized booking system, email, and phone as primary tools for communication and booking management.

Goals and Tasks

- To efficiently process all booking requests, ensuring schools receive prompt responses and high satisfaction with the booking experience.
- To manage the booking calendar effectively, maximizing the utilization of available programs and resources.

Quote

"Every call is an opportunity to make a positive impact on students' learning experiences. My goal is to ensure every school finds the perfect program for their needs."

SKILLS	PERSONALITY	
Time management	★★★★★	85%
Problem Solving	★★★★★	Personal Thinking
Teamwork	★★★★★	75%
		Humor

Emily Zhang
Booking Specialist

Age : 28
Gender : Female
Address : Melbourne

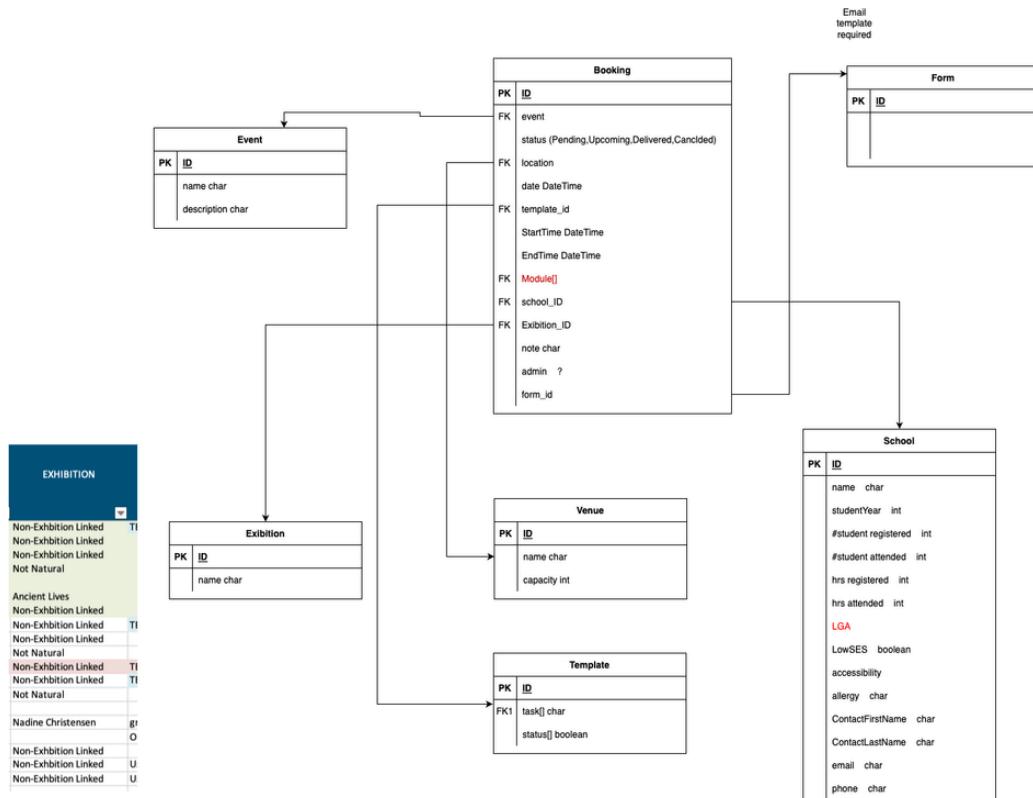
.....

Prototypes

Figma

View our design via [Booking Management System](#)

Database ER Diagram Prototype



User Stories - Product Backlog

The following user stories are based on the requirements given by the client.

Acceptance criteria is confirmed with client

- **Size Estimation/Story Points** estimate using the Fibonacci sequence, where 1 is 12 hrs, 3 is 48 hrs, 5 is half a week, 8 is a week
- **Priority** - must have, should have, could have, will not have
- **As** - reflect to [Personas](#)

Story Points	Estimated Working Hours
1	12 hours
3	48 hours
5	84 hours / half a week
8	168 hours / a week

Priority	Justification
must have	The user story is needed to satisfy client requirements and is included in Inner Scope
should have	The user story is an ideal way to satisfy client requirements and not necessarily be included in Inner Scope
could have	The user story is not essential to satisfy client requirements and is not included in Inner Scope . It is beneficial to have to improve user experience
will not have	The user story is not needed to satisfy client requirements and is not included in Inner Scope

Epic Feature 0: School teacher booking								
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
0.1	Tony Lee Client suggest that he wants to manually copy and past the booking details in the booking system. In this case 0.1 has been removed	fill in a booking form	I can book a visit for a workshop	<ul style="list-style-type: none"> • Display booking form • Must have required field of input • Let user see booking completion notification 	Must have	±	N/A	Size estimation: The current team relies on Microsoft booking form. We can adjust it to fit it in our user stories. Priority estimation: We need a booking form before

								processing the booking information by learning team member.
0.2	Tony Lee	be able to see a booking ID in my email inbox	I can update/cancel any booking later through email or call using this booking ID	<ul style="list-style-type: none"> Send email to client inbox Display booking ID 	Should have	3	0.1	<p>Size estimation: We will need to extract client email and generate unique booking ID.</p> <p>Priority estimation: Is good to have a booking ID for client. This allows the learning team staff to track booking easily by referencing the booking ID when update/cancel booking.</p>

Epic Feature 1: Learning team member booking checklist								
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
1.1	Emily Zhang	manually enter or copy the booking information if client booked by telephone/email	I can keep track of the client details and process booking.	<ul style="list-style-type: none"> Display form input field Allow user to tag the booking (based on checklist template type) Allow user to see completion notification A text field for user to paste the whole message including Program, Date, 	Must have	3	N/A	<p>Size estimation: A simple form for input and creating a record in the database</p> <p>Priority estimation: It is important that we have this method so the staff can process these booking made by phone call or email.</p>

				School, Time... <ul style="list-style-type: none">• Extract the client details from specific structure				
1.2	James Parker	tag all pending booking by checklist template type	I can keep track of all bookings process based on the template checklist	<ul style="list-style-type: none">• Display a list of pending bookings• Allow user to distinguish between different tags	Must have	3	N/A	<p>Size estimation: Only require to use distinguishable tags such as different colour and reflect on the different template</p> <p>Priority estimation: Need this for going through checklist for different types of workshop booking</p>
1.3	James Parker	update checklist template	I can have a more up to date checklist template for tracking booking progress	<ul style="list-style-type: none">• Allow user to see current template by name• Allow user to only update the required field	Could have	3	N/A	<p>Size estimation: Update all booking checklist with the same tagged template, need to search and update all relevant bookings in database.</p> <p>Priority estimation: not the most important, but it will be flexible to have it.</p>
1.4	James Parker	see all bookings and see each booking checklist progress	I can track progress of all bookings and individual one's	<ul style="list-style-type: none">• Allow user to see all booking details through a calendar like timetable• Allow user to search for target booking	Should have	5	1.1	<p>Size estimation: Need searching for one individual booking for convenience.</p> <p>Priority estimation: Learning team member should be seeing this if they want to track the</p>

									progress using the checklist
1.5	James Parker	update booking details	I can track the more up to date booking information.	<ul style="list-style-type: none"> Allow user to see all booking details Allow user to search for target booking Allow user to only update the required field 	Should have	3		1.1, 1.4	<p>Size estimation: Need searching for target booking for convenience. Need update target booking details in all other platform</p> <p>Priority estimation: This is a necessary functionality, but automation can be develop later in the development.</p>

Epic Feature 2: Connection and negotiation with Priava venue booking system								
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
2.1	James Parker	check Priava venue availability	I know if there is space available for a booking	<ul style="list-style-type: none"> Allow user to login to Priava Allow user to see Priava information inside the booking system 	Must have	5	N/A	<p>Size estimation: Need to use Priava API to embed in our booking system, setup Authentication to use this API</p> <p>Priority estimation: Priava is essential in this booking system</p>
2.2	James Parker	have booking details copied into Priava if there is availability	I can have the venue booked	<ul style="list-style-type: none"> Allow user to submit booking details to priava 	Must have	3	1.1, 1.2, 2.1	<p>Size estimation: Need to linked to Priava API for automation</p> <p>Priority estimation: This is required for venue booking and booking information need to be shown in Priava as well.</p>

Epic Feature 3: Microsoft Calendar management								
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
3.1	James Parker	add Priava booking information to the shared Microsoft Calendar	we can provide facilitators with immediate access to venue availability	<ul style="list-style-type: none"> Display correct time slot Information should be same as one given from Priava 	Must have	1	2.1,2.2	<p>Size estimation: Need to linked to Microsoft API for automation</p> <p>Priority estimation: This is required for venue booking and booking information need to be shown in Microsoft Calendar</p>
3.2	James Parker	invite facilitators to assist with scheduling	we can ensure the schedules are processed and audited efficiently	<ul style="list-style-type: none"> if casual facilitator required, roster via When I Work 	Must have	3	3.1	<p>Size estimation: Need to associate with When I work</p> <p>Priority estimation: facilitator is essential process and audit the tasks</p>

Epic Feature 4: Store booking details in Database								
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
4.1	James Parker	see a list of booking details in the booking system	I can select a booking and update the booking information	<ul style="list-style-type: none"> Display grouped data Allow user to update booking details Allow user to search a booking by booking id 	Must have	3	1.1, 1.2	<p>Size estimation: Link database to the booking system. Data should be stored once booking details was entered by client or learning team members.</p> <p>Priority estimation: Required for learning team member to track historical bookings.</p>
4.2	James Parker	see analysis of the booking details	I can report to the team	<ul style="list-style-type: none"> Display visual analysis 	Could have	3	1.1, 1.2, 4.1	<p>Size estimation: Need to process booking data and need to use</p>

				<ul style="list-style-type: none"> Display data trend 					packages for analysis.
4.3	James Parker	update a booking information	I can have the up to date information for reporting	<ul style="list-style-type: none"> Display most recent booking details Allow user to see update success notification 	Should have	1		4.1	<p>Size estimation: Need to update data in the database and show pop out video for update success notification.</p> <p>Priority estimation: Not necessarily needed in the booking system, but good to have for analytic task.</p>

Epic Feature 5: Accessibility Arrangements for booking									
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification	
5.1	James Parker	record if needed to book an interpreter for the client's school	deaf or hard of hearing students can fully engage with the visit.	<ul style="list-style-type: none"> Option to select the need for an Auslan interpreter during booking Link to Auslan booking service website 	Could have	1	3.1	<p>Size estimation: System should able to connect to Auslan Services including the process for requesting and confirming interpretation services.</p> <p>Priority estimation: Ensuring accessibility is very important for inclusive</p>	

								educational experiences.
5.2	James Parker	record bus booking details for partner schools using BusCharter.com.au	bus payment details are included for cost calculation and reporting	<ul style="list-style-type: none"> Option to request bus transport during the booking process. Interface to book buses directly or send requests to Bus Hire - Low Cost Bus Charter & Coach Hire in Australia 	Should have	3	3.1	<p>Size estimation: Add a link in checklist to connect to BusCharter.com.au and record booking details in booking system.</p> <p>Priority estimation: Promote school participation and make the whole process easier.</p>

Epic Feature 6: Risk								
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
6.1	James Parker This is not done through the booking system as the client need to contact their customer directly.	send-prewritten risk assessments to schools for all our programs	teachers can prepare and address any concerns prior to the visit.	<ul style="list-style-type: none"> Automated distribution of risk assessment upon booking confirmation. 	Could have	±	1.1, 1.2	<p>Size estimation: Enable automatic sending of risk assessments</p> <p>Priority estimation: It is very important for schools to understand possible risks before visiting and to promote a safe environment for students.</p>
6.2	James Parker	confirm booking details with teachers two weeks before their scheduled visit	any necessary adjustments can be made in a timely manner.	<ul style="list-style-type: none"> Automated reminder to send confirmation emails two weeks before the booking. 	Could have	1	3.1	<p>Size estimation: Enable automatic reminder and confirmation email sending</p> <p>Priority estimation: Reduces the potential for</p>

								misunderstandings is very important.
--	--	--	--	--	--	--	--	---

Epic Feature 7: Health and Safety Information

ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
7.1	James Parker	gather and manage medical information, allergies, or accessibility needs for the visiting groups	we can ensure a safe and accommodating visit for all attendees.	<ul style="list-style-type: none"> Secure form for collecting sensitive medical and accessibility information during the booking process. 	Must have	3	3.1	<p>Size estimation: Collecting sensitive medical and accessibility information requires keeping the data secure and private and the data can be easily accessible and manageable by internal personnel.</p> <p>Priority estimation: It is vital to prepare and provide a safe, inclusive visiting environment that meets the needs of all participants.</p>

Epic Feature 8: Post-visit Reporting

ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
8.1	James Parker	accurately record and report the number of students who attended each program in our database	we can evaluate our outreach and report on our educational impact.	<ul style="list-style-type: none"> Interface for entering attendance numbers into the LER database. 	Must have	3	N/A	<p>Size estimation: Implement an interface to enter and update attendance numbers in the database</p> <p>Priority estimation: Providing valuable data to assess program effectiveness and</p>

								plan future delivery is a must.
--	--	--	--	--	--	--	--	------------------------------------

Epic Feature 9: Financial Processing								
ID	As	I want to	So that	Acceptance Criteria	Priority	Size Estimation	Dependency	Justification
9.1	James Parker	record payment details for workshops, interpreters and buses.	I can confirm the payment has been received or made by the team.	<ul style="list-style-type: none"> Record invoice number, payment confirmation 	Must have	1	3.1	<p>Size estimation: Need to shown in booking details with option to update the payment information</p> <p>Priority estimation: This is part of booking update and create process.</p>

Provided Information - User Stories Support Details

Topic		
Planner	The staff member copies the booking details into Microsoft Planner/Tasks into the checklist template so they can track the steps of the booking process.	
Priava	The staff member checks Priava (venue booking system) that the workshop space or gallery space is available to be used, then copies the booking information into a Priava booking. If there is not sufficient space, they will negotiate with the school around an alternative date and time	
MSC + Invites	Once the Priava booking is confirmed, the staff member will add the booking to a shared Microsoft Outlook calendar, once again copying over the booking information. Outlook is used as the main tool by our team as it allows us to invite facilitators to the programs to assist with their own scheduling. If casual facilitators are required, they will also be rostered on here using When I Work.	
LER + update status	This is the stage where the staff member will enter the data into our excel spreadsheet database of all the programs we have run, manually entering all the data that we capture from the booking process. We also use this data for our reporting. For example, this would be (how many students from X school attended in term 1, and how many total hours did they spent here).	
Additional resources - booking is also recorded in database link to the target workshop booking	<i>Book interpreters</i>	Some of our schools have deaf/hard of hearing students that require interpreters. These are booked through Auslan Services.
	<i>Book bus if needed</i>	If a partner school would like to have a bus to our venues, we will book a bus through Bus Hire - Low Cost Bus Charter & Coach Hire in Australia .
	<i>Paid for interpreter/bus</i>	Our staff member will process the payments for buses and interpreters through the university system. This is currently done through Themis but the university is changing this system some time in the year.
Update information in database	<i>Risk assessments</i>	We have prewritten risk assessments for all our programs that we send out to schools.

<i>Confirm with teacher</i>	the staff member will send out a confirmation email two weeks before the booking to make sure the details are still correct.
<i>Medical information</i>	the staff member will obtain any medical information, allergies or accessibility needs for the group and update the Outlook calendar with the information.
<i>Numbers on the day</i>	We need to enter how many students attended into our database spreadsheet (the LER). This is for reporting purposes.

 RISK1. **Risk Registry:**

- Includes information such as the **description of the risk, its impact, likelihood, mitigation strategies**, and responsible person for managing the risk.
- The registry is regularly updated to reflect new risks, changes in risk status, and the effectiveness of risk response strategies.

2. **Review Meetings:**

- Scheduled meetings to **discuss and review the current risk status**, effectiveness of mitigation actions, and identification of new risks.

Risk Registry

Risk ID	Description	Category	Probability	Impact	Risk Level	Mitigation Plan	Owner	
0	API failure leading to interruption in service delivery	Technical	Medium	High	High	Implement alternative APIs; Regular API health checks; Develop a rapid response plan for API failure	@Guixian Li	
1	Software updates to the platform or database causing incompatibility issues	Technical	Medium	High	High	Test any software updates in a controlled environment before rolling them out	@Yuanbo Xu	
2	Data leakage in database due to unsecured data	Security	Low	High	Medium	Preprocess data before saving to ensure no sensitive information is stored	@Yuanbo Xu	[Date]
3	Inaccurate project time estimation leading to delays	Management	High	Medium	High	Implement more rigorous time estimation techniques	@Guixian Li	

Review Meetings



COLLABORATION

1. About Us:

- Provides an **overview of the team**.

2. Team Calendar:

- A shared calendar that **tracks all team-related events, meetings, deadlines**, and important milestones.

3. Agile Processes:

- Describes the **specific Agile methodologies and practices** adopted by the team.

4. Agile Ceremonies:

- Refers to the **essential meetings or events** in Agile methodologies that help teams to implement Agile practices effectively.
- Includes ceremonies like **Sprint Planning, Sprint Review, and Sprint Retrospective**.

About Us

Team Info

NAME	ROLE	CONTACT
@Yujun Yan	Mentor	?
@Guixian Li <i>Leona</i>	Product Owner	guixianl@student.unimelb.edu.au
@Daniel Su <i>Daniel</i>	Developer	dss1@student.unimelb.edu.au
@Yuanbo Xu <i>Chris</i>	Developer	yuanbo@student.unimelb.edu.au
@Pangfeng ZHENG <i>Jack</i>	Scrum Master	pangfenz@student.unimelb.edu.au
@Yongli Qin <i>Matt</i>	Developer	yongliq@student.unimelb.edu.au
@Yun-Chi Hsiao <i>Jim</i>	Recorder	yunchi@student.unimelb.edu.au

External

- GitHub <https://github.com/COMP90082-2024-SM1/SG-Koala> Connect your Github account
- Trello [COMP90082_2024_SM1_SG_KOALA](#)
- Slack Channel
 - #comp90082-2024-sg-koala
 - #client-meeting
 - #general
 - #meeting

Team Calendar

Meeting Type	Meeting time	People Involved
Sprint Plan Meeting	Before each Sprint start	All Team Member
Product Backlog Refinement Meeting	Before each Sprint start	All Team Member
Weekly Stand-up Meeting	Twice a week (Monday, Wednesday)	All Team Member
Supervisor Meeting	Weekly (Wednesday)	All Team Member, Supervisor
Review Meeting	After a Sprint finished	All Team Member, Client
Retrospective Meeting	After a Sprint finished	All Team Member

Agile Processes

Team Charter

Team Purpose and Mission

- Aim to significantly reduce workload and enhance user experience.
- Develop a user-friendly, efficient software application to improve the Science Gallery's booking system.

Scope of Work

- Design, develop, and test the application.
- Interface with the Science Gallery's existing systems and databases.
- Ensure accessibility and ease of use for various user groups.

Roles and Responsibilities

- Project Manager: Manages project schedules and liaises with the Science Gallery.
- Scrum Master: Guides and assists the Scrum team, ensuring Scrum practices are followed, resolving obstacles, and promoting a productive team environment.
- Recorder: Documents the development process comprehensively, including creating user guides.
- Developers: Responsible for coding and building the software application, including writing, testing, and debugging code, and collaborating on design and implementation decisions.

Procedures and Processes

- Weekly team meetings to discuss current work.
- Weekly meetings with Supervisor for progress reviews.
- Sprint sessions include planning, retrospectives, and client reviews.
 - Planning: Organising and defining tasks and objectives for the sprint.
 - Retrospective: Reflecting on the past sprint to identify improvements.
 - Review with Clients: Discussing progress and feedback with clients.
- Use of collaboration tools.
 - Trello for task management.
 - Slack for communication.

Resources

- Team members will independently research and document resources, compiling them in a shared document [🔗 LINKS & RESOURCES](#) for easy access and reference.
- Additionally, the team will coordinate with the client to obtain a testing account and necessary data for effective system testing and validation.

Performance Metrics

- Completion of development milestones on schedule.
- Reduction in booking system workload as reported by the Science Gallery.
- User satisfaction and feedback post-implementation.

Conflict Resolution Strategies

- Encourage open and respectful communication to address issues as they arise.
- Implement regular check-ins to identify and resolve emerging conflicts early.
- Facilitate mediation sessions with a neutral party for unresolved conflicts.

Duration and Timeframes

The team will operate over the duration of the semester, with the project structured into four distinct sprints:

- Sprint 1: 3 weeks.
- Sprint 2: 4 weeks.
- Sprint 3: 4 weeks.
- Sprint 4: 2 weeks.

The project is set to conclude by the end of the semester, aligning with these sprint durations for effective planning and execution.

Definition of Ready

The "Definition of Ready" in software development and Agile methodologies refers to a set of criteria that a user story or task must meet before it can be considered ready to be worked on in a sprint or development cycle. This concept ensures that work items are adequately prepared and understood before the team starts working on them, promoting efficiency and reducing the risk of misunderstandings or rework. Key aspects of the Definition of Ready typically include:

- **Clear Acceptance Criteria:** Specific conditions that must be met for the work to be considered complete.
- **Dependencies Resolved:** All dependencies on other tasks or external factors are identified and resolved.
- **Estimated:** The task has been sized or estimated by the team.
- **Feasible:** The team agrees that the task is possible to complete within the sprint or cycle.
- **Prioritised:** The task fits within the team's priorities and product roadmap.
- **Understood by the Team:** Everyone involved has a clear understanding of what needs to be done and why.

The Definition of Ready is a crucial component of Agile project management, ensuring that tasks are well-prepared before development begins, thus improving the team's efficiency and effectiveness.

Definition of Done

The "Definition of Done" in Agile and software development is a critical agreement that determines when a task or user story is considered complete. It serves as a checklist that ensures quality and completeness. Key aspects typically include:

- **Quality Standards Met:** The work meets predefined quality criteria, such as code quality, design standards, and performance benchmarks.
- **Acceptance Criteria Fulfilled:** The task satisfies all the conditions specified in the user story or requirement.
- **Code Review and Testing:** The code has been reviewed, and all necessary testing (unit tests, integration tests, etc.) has been successfully completed.
- **Documentation Updated:** Relevant documentation (technical, user manuals, etc.) has been updated to reflect the changes.
- **Approved by Stakeholders:** The work is reviewed and approved by the relevant stakeholders, such as product owners, clients, or team leads.
- **Potentially Shippable:** The increment is in a state where it could be released to users, even if it won't be shipped immediately.

The Definition of Done ensures transparency and a shared understanding of what it means for work to be complete, helping to avoid ambiguities and ensuring that each product increment is of high quality.

Agile Ceremonies

Sprint Planning - 1

Date & Time

13:00 Mar. 13th, 2024

Participants

- @Yuanbo Xu
- @Pangfeng ZHENG
- @Yongli Qin
- @Daniel Su
- @Yun-Chi Hsiao
- @Guixian Li

Goals

- Determine the specific roles for each team member.
- Allocate specific tasks to each team member.

Discussion Topics

Time	Item	Presenter	Notes
10 mins	Roles	@Guixian Li	Discuss each team member's personality and capabilities to assign roles effectively.
30 mins	Tasks	@Guixian Li	Compile the tasks for Sprint 1 using the checklist provided on the LMS.

Action Items

- Determine the specific roles for each team member.
- Allocate specific tasks to each team member.

Decisions

👉 @Guixian Li will be the **project owner**.

👉 @Yuanbo Xu is the **scrum master**.

👉 @Yun-Chi Hsiao is the **recorder**.

👉 @Guixian Li @Yuanbo Xu @Yongli Qin are responsible for **REQUIREMENTS** section.

👉 @Pangfeng ZHENG takes responsibility for overseeing **Trello** and planning **SPRINT-1**.

👉 @Yun-Chi Hsiao manages **Confluence**, **GitHub**, and **COLLABORATION** content.

 @Daniel Su is responsible for the  BACKGROUND section.

Retrospective - 1

Date & Time

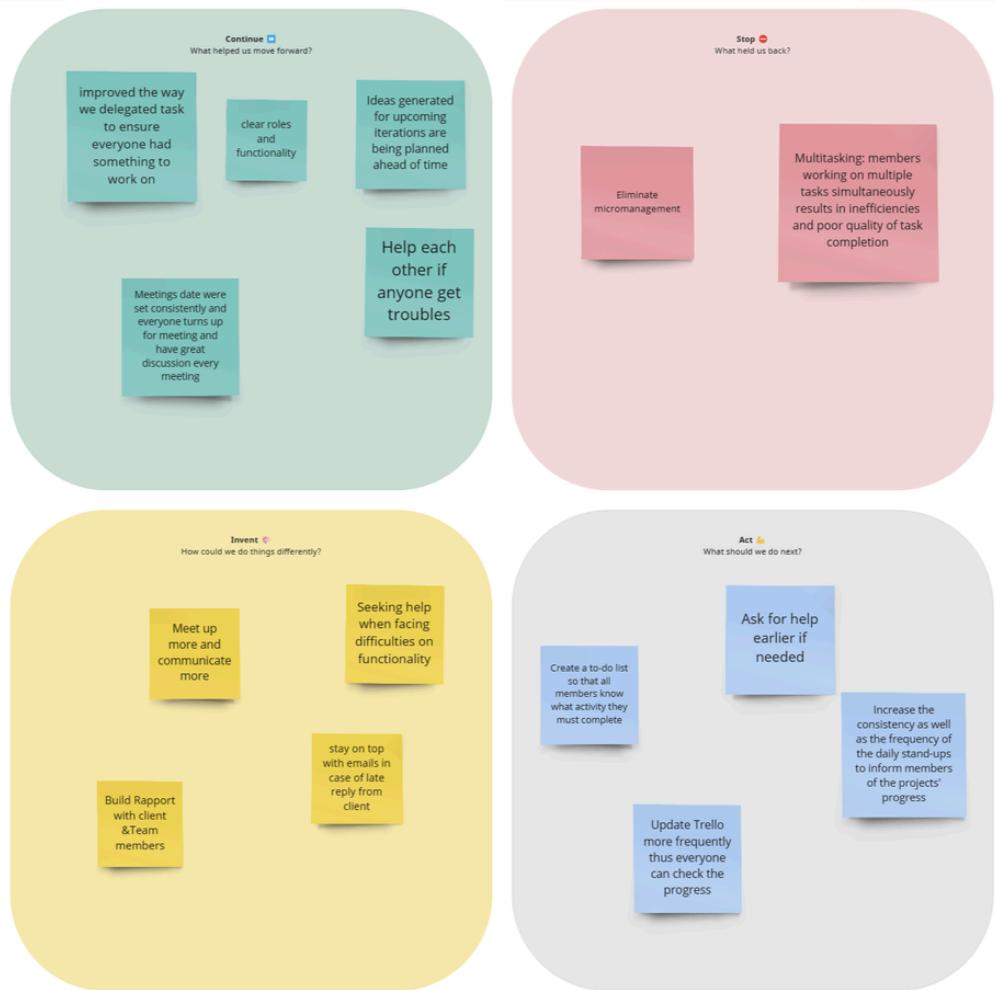
11:00 Mar. 27th, 2024

Participants

- [@Yuanbo Xu](#)
- [@Pangfeng ZHENG](#)
- [@Yongli Qin](#)
- [@Daniel Su](#)
- [@Yun-Chi Hsiao](#)
- [@Guixian Li](#)

Goals

- Reflection and evaluation: Review all aspects of the last Sprint together, including problems encountered when performing tasks, team collaboration, and achieving goals.
- Identify successes and challenges: Identify which practices are successful and should be maintained and which challenges or problems require improvement.



Review Meeting - 1

📅 Date & Time

09:30 Mar. 27th, 2024

👥 Participants

- @Yuanbo Xu
- @Pangfeng ZHENG
- @Yongli Qin
- @Daniel Su
- @Yun-Chi Hsiao
- @Guixian Li
- The client

📋 Goals

- Review current design.
- Questions for the client.

🗣 Discussion Topics

Time	Item	Presenter	Notes
30 mins	Figma Review	The client	<ol style="list-style-type: none">1. Overall, it's well-designed.2. The search bar should enable users to find both school and non-school partners.3. A textbox for staff to paste booking details from emails should be included.
20 mins	Discussion	@Guixian Li @Yuanbo Xu @Yongli Qin @Daniel Su @Pangfeng ZHENG @Yun-Chi Hsiao	<ol style="list-style-type: none">1. Chris will supply the email template.2. When clicking a hyperlink, it will open in a new page rather than being embedded in the current page.3. Consideration of whether different clients should use the same account.

✓ Action Items

- ✓ Let's review our Figma designs and address the questions that arose during our design phase.
- ✓ We'll undertake a review of the Figma designs to discuss any issues we encountered in the design process.
- ✓ Update Figma and User Stories according to client needs. @Yongli Qin @Yun-Chi Hsiao @Guixian Li
- ✓ Improve Confluence structure and improve document quality @Yuanbo Xu @Daniel Su @Pangfeng ZHENG
- ✓ @Pangfeng ZHENG Update Trello according to user stories

Decisions

- 👉 The same account can be accessed by multiple staff members on various devices.
- 👉 Priava, Microsoft, and the bus booking system all incorporate hyperlinks.
- 👉 Incorporate a textbox for entering an email address into the form creation process.
- 👉 **Chris says we will get H1!!!**

Sprint Planning - 2

Date & Time

13:00 Mar. 27th, 2024

Participants

- @Yuanbo Xu
- @Pangfeng ZHENG
- @Yongli Qin
- @Daniel Su
- @Yun-Chi Hsiao
- @Guixian Li

Goals

- client review on sprint 1
- update requirements based on client review
- Initialise development architecture, front end and back end
- design database structure
- enable authorisation to external API
- complete user stories allocated
- monitor risk
- update risk management strategy if detect new risk
- update development speed

Discussion Topics

Time	Item	Presenter	Notes
10 mins	Roles, Idea	@Guixian Li	Discuss each team member's personality and capabilities to assign roles effectively. Discuss each team member's idea on Figma
30 mins	Tasks	@Guixian Li	Compile the tasks for Sprint 1 using the checklist provided on the LMS.

Action Items

- Determine the specific roles for each team member.
- Allocate specific tasks to each team member.

Decisions

👉 We split features between members based on Trello - see [Trello](#)

👉 Update prototype design based on client feedback - add a new text field in Booking form allowing clients to paste booking details and auto matching into our booking form

👉 Confirm typography

Review Meeting - 2

📅 Date & Time

15:00 May. 7th, 2024

👥 Participants

- @Yuanbo Xu
- @Pangfeng ZHENG
- @Yongli Qin
- @Daniel Su
- @Yun-Chi Hsiao
- @Guixian Li
- The client

📋 Goals

- Review current design.
- Questions for the client. [📝 Sprint-2 Goal](#)

🗣 Discussion Topics

Time	Item	Presenter	Notes
30 mins	Product Review	@Guixian Li	<ol style="list-style-type: none">1. Overall, it's good.2. Booking form.3. Search bar.4. Template.5. Login page.6. See 📝 Sprint-2 Goal
20 mins	Discussion	@Guixian Li @Yuanbo Xu @Yongli Qin @Daniel Su @Pangfeng ZHENG @Yun-Chi Hsiao	<ol style="list-style-type: none">1. Booking form fields.2. Booking status.3. Analytics content.

✓ Action Items

- Assign tasks to members accordingly
- Resolve bugs in the last sprint
- Analyse deployment plans
- prepare data samples

Decisions

-  User stories are allocated to frontend and backend members
-  Reestimate leftover user stories
-  Confirm newest internal sprint deadline

Retrospective - 2

Date & Time

11:00 May. 15th, 2024

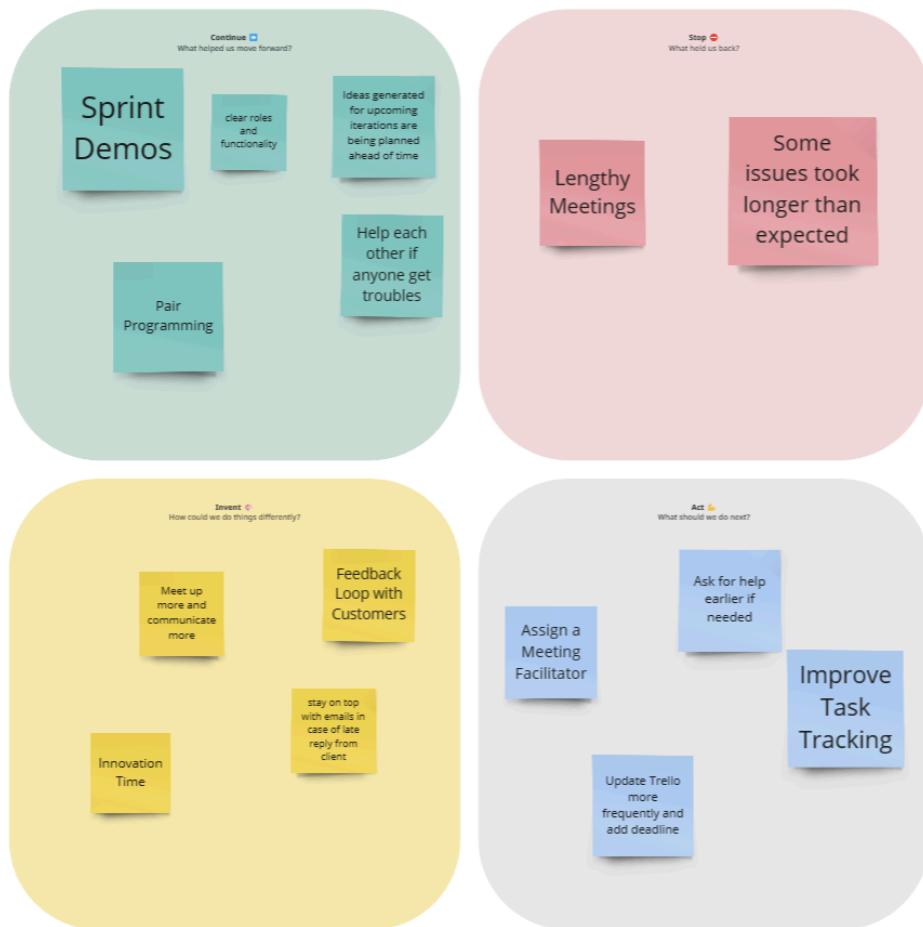
Participants

- [@Yuanbo Xu](#)
- [@Pangfeng ZHENG](#)
- [@Yongli Qin](#)
- [@Daniel Su](#)
- [@Yun-Chi Hsiao](#)
- [@Guixian Li](#)

Goals

- Reflection and evaluation: Review all aspects of the last Sprint together, including problems encountered when performing tasks, team collaboration, and achieving goals.
- Identify successes and challenges: Identify which practices are successful and should be maintained and which challenges or problems require improvement.

Sprint 2 Retrospective



Sprint Planning - 3

Date & Time

13:00 April. 10th, 2024

Participants

- @Yuanbo Xu
- @Pangfeng ZHENG
- @Yongli Qin
- @Daniel Su
- @Yun-Chi Hsiao
- @Guixian Li

Goals

- client review on sprint 2
- complete unfinished user stories in [sprint 2](#)
- update requirements based on client review
- continue development on allocated user stories
- update risk management strategy if detect new risk
- test and document completed features
- complete user stories allocated
- update development speed

Discussion Topics

Time	Item	Presenter	Notes
10 mins	Reflect Sprint 2 Task	@Guixian Li	<p>Discuss unfinished tasks from Sprint 2.</p> <p>Reflect on Sprint 2 code, design and feedback.</p> <p>Compile the tasks for Sprint 2 using the checklist provided on the LMS.</p>
30 mins	Sprint 3 Tasks	@Guixian Li	Assign tasks in Sprint 3 according to each member's role and responsibility.

✓ Action Items

- ✓ Determine the specific tasks for Sprint 3.
- ✓ @Yuanbo Xu @Guixian Li Backend API, Unit testing
- ✓ @Pangfeng ZHENG Interaction between Front and Back end
- ✓ @Yongli Qin @Daniel Su @Yun-Chi Hsiao Frontend Development
- ✓ Allocate specific tasks to each team member.

Decisions

-  We split features between members based on Trello - see [Trello](#)
-  Update Prototypes to improve user experiences - see [Prototype](#)
-  Confirm Product from Sprint 2 with the client

Retrospective - 3

Date & Time

11:00 May. 20th, 2024

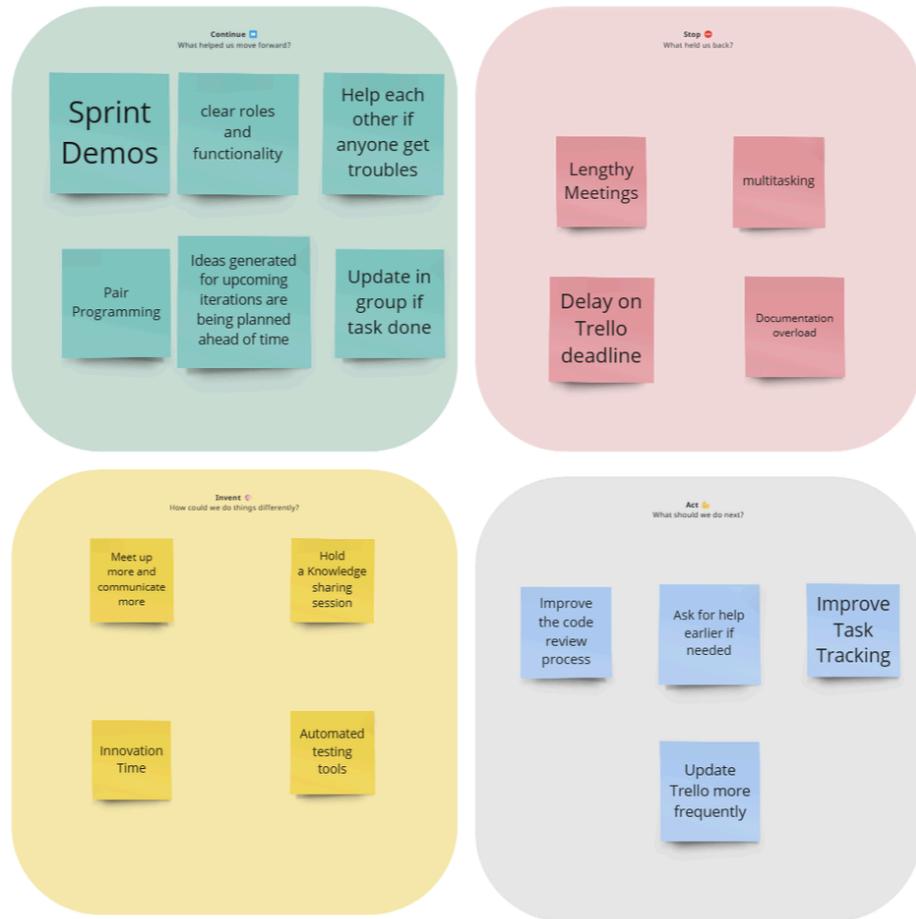
Participants

- [@Yuanbo Xu](#)
- [@Pangfeng ZHENG](#)
- [@Yongli Qin](#)
- [@Daniel Su](#)
- [@Yun-Chi Hsiao](#)
- [@Guixian Li](#)

Goals

- Reflection and evaluation: Review all aspects of the last Sprint together, including problems encountered when performing tasks, team collaboration, and achieving goals.
- Identify successes and challenges: Identify which practices are successful and should be maintained and which challenges or problems require improvement.

Sprint 3 Retrospective



Review Meeting - 3

Date & Time

14:00 May. 22th, 2024

Participants

- [@Pangfeng ZHENG](#)
- [@Daniel Su](#)
- [@Yun-Chi Hsiao](#)
- [@Guixian Li](#)
- [@Yongli Qin](#)
- [@Yuanbo Xu](#)
- **The client**

Goals

- Review final design.

Discussion Topics

Time	Item	Presenter	Notes
30 mins	Product Review	@Guixian Li	<ol style="list-style-type: none">1. Sprint 3 Product Demo2. Reveal all functions3. Follow-up database and deployment cost

Action Items

- Resolve bugs in creating a new booking
- Test all other functions
- Enable delete for booking input selections

Sprint Planning - 4

Date & Time

13:00 May. 20th, 2024

Participants

- @Yuanbo Xu
- @Pangfeng ZHENG
- @Yongli Qin
- @Daniel Su
- @Yun-Chi Hsiao
- @Guixian Li

Goals

- client review on sprint 3
- complete unfinished user stories in **sprint 3**
- update requirements based on client review
- refine UI design
- finish test and document completed features
- deployment-ready and list deployment costs for client
- hand in the final product with database access and deployment URL

Discussion Topics

Time	Item	Presenter	Notes
10 mins	Reflect Sprint 3 Task	@Guixian Li	<p>Discuss unfinished tasks from Sprint 3.</p> <p>Reflect on Sprint 3 code, design and feedback.</p> <p>Compile the tasks for Sprint 3 using the checklist provided on the LMS.</p>
30 mins	Sprint 4 Tasks	@Guixian Li	<p>List unfinished requirements.</p> <p>List Deployment plans and costs.</p> <p>List the items need to be handed in (database access, email host)</p> <p>Need to Merge Database Login and authentication</p>

✓ Action Items

- Determine the specific tasks for Sprint 4.
- @Yuanbo Xu @Daniel Su Merge Database Login and Authentication to the system.
- @Yuanbo Xu @Guixian Li Prepare database access for client, Plans for deployment
- @Pangfeng ZHENG Complete Frontend testing

@Yongli Qin @Daniel Su @Yun-Chi Hsiao Refine UI Design

Prepare suggestions for plans and cost of deployment

⌚ Decisions

👉 Confirm Product from Sprint 3 with the client

👉 Deployment Plan

👉 When to hand in the final product

👉 How the client should access the database

Retrospective - 4

Date & Time

15:00 June, 4th, 2024

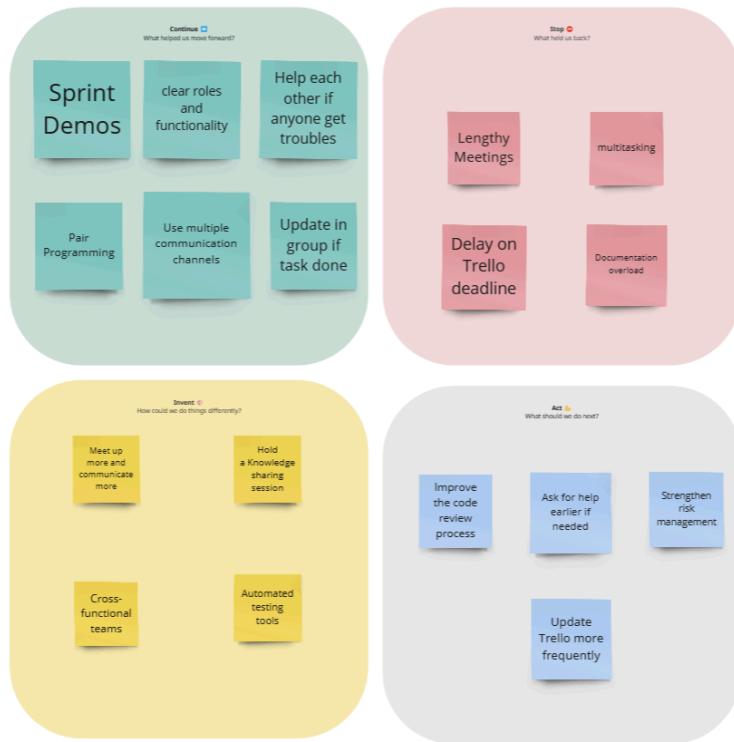
Participants

- [@Yuanbo Xu](#)
- [@Pangfeng ZHENG](#)
- [@Yongli Qin](#)
- [@Daniel Su](#)
- [@Yun-Chi Hsiao](#)
- [@Guixian Li](#)

Goals

- Reflection and evaluation: Review all aspects of the last Sprint together, including problems encountered when performing tasks, team collaboration, and achieving goals.
- Identify successes and challenges: Identify which practices are successful and should be maintained and which challenges or problems require improvement.

Sprint 4 Retrospective



Review Meeting - 4

Date & Time

11:15 June, 4th, 2024

Participants

- @Pangfeng ZHENG
- @Daniel Su
- @Yun-Chi Hsiao
- @Guixian Li
- The client

Goals

- Review final design.

Discussion Topics

Time	Item	Presenter	Notes
30 mins	Product Review	@Guixian Li	1. Final Product Demo

Action Items

Decisions

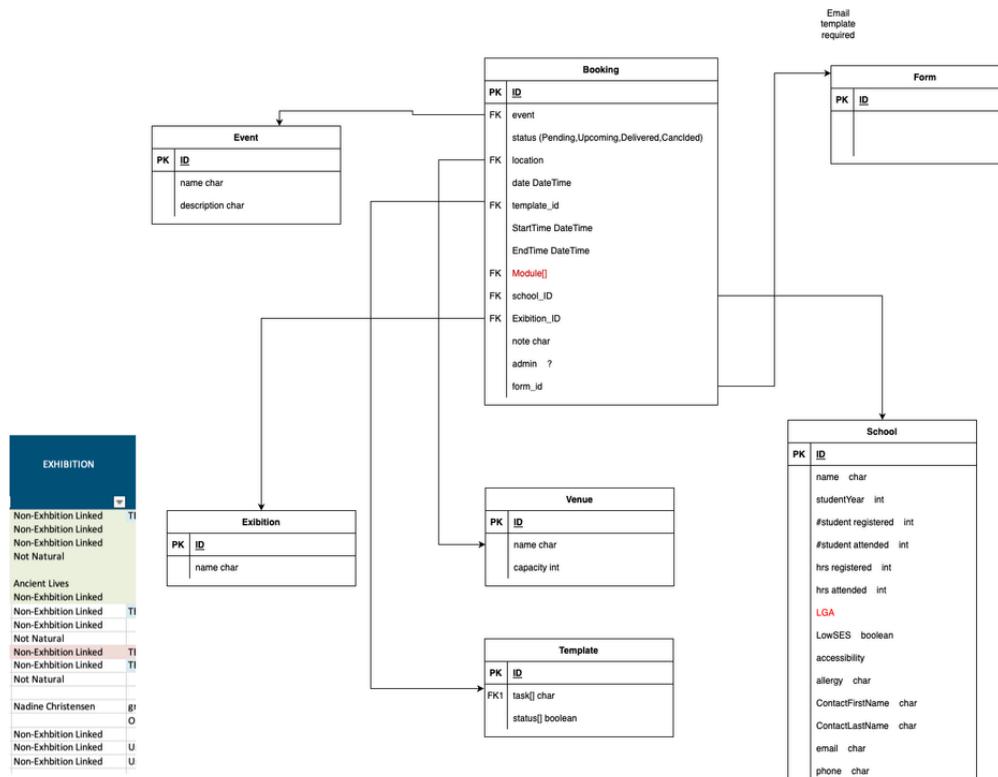
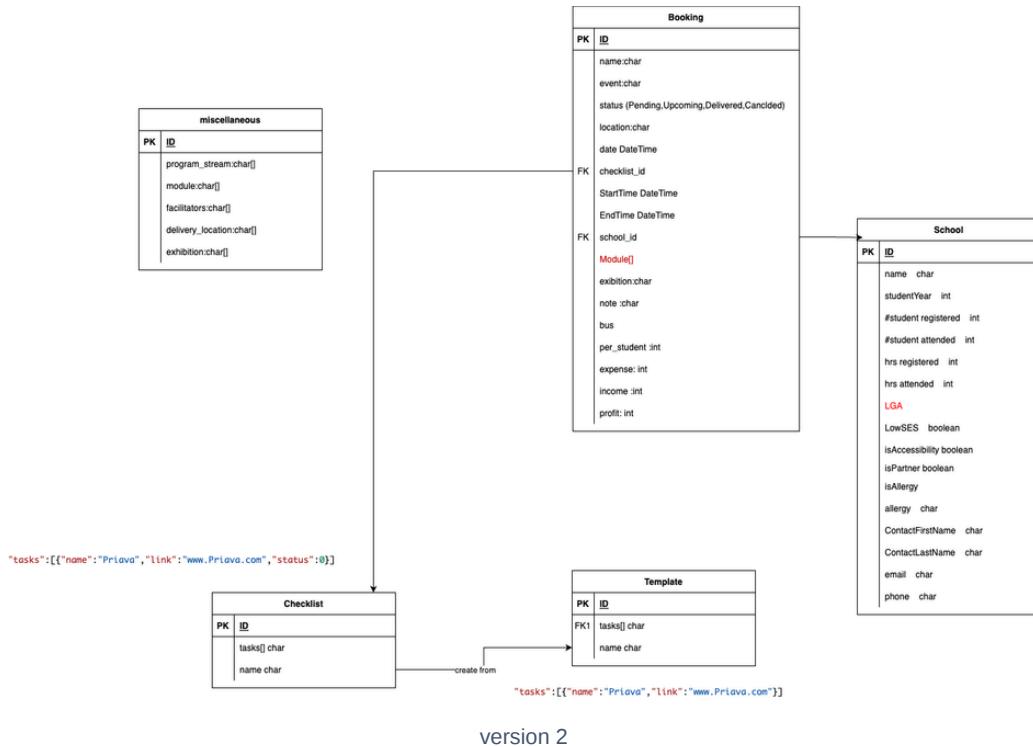
 ARCHITECTURE1. **Database Model:**

- Represents the **structure of the database** including the relationships and constraints between data entities.

2. **API (Application Programming Interface):**

- Defines the **interactions** between multiple software intermediaries, specifying how they should **communicate** and **share data**.

Database Model



API

✓ API Naming Convention Example

- GET api/ticket/ - Retrieves all tickets
- GET api/ticket/11321312adfdas/ - Retrieves a specific ticket
- POST api/ticket/ - Creates a new ticket
- PUT api/ticket/12312321fdasfsda/ - Updates ticket #12
- DELETE api/ticket/12321321fdsafdsf/ - Deletes ticket #12
- All requests end with /

📘 Postman API

<https://www.postman.com/kaibai/workspace/sc-api>



📘 Template

Method	URL Example	Request Body	Response	Description
GET	api/template/		<pre>1 [2 { 3 "id": "6614cea3647dc4b8a12f2625", 4 "name": "template1", 5 "task": [6 { 7 "name": "step1", 8 "link": "http://www.google." 9 }, 10 { 11 "name": "step2", 12 "link": "http://www.example" 13 }, 14 { 15 "name": "step3", 16 "link": "" 17 }, 18 { 19 "name": "step4", 20 "link": "" 21 } 22] 23 }, 24 { 25 "id": "6614cea3647dc4b8a12f2625", 26 "name": "template1", 27 "task": [</pre>	

```

28     {
29         "name": "step1",
30         "link": "http://www.google.com"
31     },
32     {
33         "name": "step2",
34         "link": "http://www.example.com"
35     },
36     {
37         "name": "step3",
38         "link": ""
39     },
40     {
41         "name": "step4",
42         "link": ""
43     }
44 ]
45 }
46 ]

```

GET	api/template/6614cea3647dc4b8a12f2625/	<pre> 1 { 2 "id": "6614cea3647dc4b8a12f2625", 3 "name": "template1", 4 "task": [5 { 6 "name": "step1", 7 "link": "http://www.google.com" 8 }, 9 { 10 "name": "step2", 11 "link": "http://www.example.com" 12 }, 13 { 14 "name": "step3", 15 "link": "" 16 }, 17 { 18 "name": "step4", 19 "link": "" 20 } 21] 22 } </pre>
-----	--	--

POST	api/template/	<pre> 1 { 2 "id": "6614cea3647dc4b8a12f2625", 3 "name": "template1", 4 "task": [5 { 6 "name": "step1", 7 "link": "http://www.google.com" 8 }, 9 { 10 "name": "step2", 11 "link": "http://www.example.com" 12 }, 13 { 14 "name": "step3", 15 "link": "" 16 }, 17 { 18 "name": "step4", 19 "link": "" 20 } 21] 22 } </pre>	<pre> 1 { 2 "name": "template2", 3 "task": [4 { 5 "name": "step1", 6 "link": "http://www.google.com" 7 }, 8 { 9 "name": "step2", 10 "link": "http://www.example.com" 11 }, 12 { 13 "name": "step3", 14 "link": "" 15 }, 16 { 17 "name": "step4", 18 "link": "" 19 } 20] 21 } </pre>
------	---------------	--	--

		<pre> 17 { 18 "name": "step4", 19 "link": "" 20 } 21] 22 }</pre>	<pre> 17 "name": "step4", 18 "link": "" 19 } 20], 21 "_id": "661e4be0e977d7711c301ae9" 22 }</pre>	
PUT	api/template/661e4be0e977d7711c301ae9/	<pre> 1 { 2 "name": "template2", 3 "task": [4 { 5 "name": "step1", 6 "link": "" 7 }, 8 { 9 "name": "step2", 10 "link": "" 11 }, 12 { 13 "name": "step3", 14 "link": "" 15 }, 16 { 17 "name": "step4", 18 "link": "" 19 } 20] 21 }</pre>	<pre> 1 { 2 "status": "success", 3 "id": "661e4be0e977d7711c301ae9", 4 "updated": 1 5 }</pre>	
DELETE	api/template/66177abaa93b4dbfb5e70fa/		<pre> 1 { 2 "status": "success", 3 "id": "66177abaa93b4dbfb5e70fa" 4 }</pre>	

Checklist

Method	URL Example	Request Body	Response	Description
GET	api/checklist/		<pre> 1 [2 { 3 "id": "661e4c8de977d7711c301aee", 4 "name": "template2", 5 "task": [6 { 7 "name": "step1", 8 "link": "http://www.baidu.com", 9 "status": 0 10 }, 11 { 12 "name": "step2", 13 "link": "http://www.example.com", 14 "status": 0 15 }, 16 { 17 "name": "step3", 18 "link": "http://www.test.com", 19 "status": 0 20 } 21] 22 }</pre>	

```

18         "link": "",
19         "status": 0
20     },
21     {
22         "name": "step4",
23         "link": "",
24         "status": 0
25     }
26   ]
27 }
28 ]

```

GET	api/checklist/66 1e4c8de977d7 711c301aee/	<pre> 1 { 2 "id": "661e4c8de977d7711c301aee", 3 "name": "template2", 4 "task": [5 { 6 "name": "step1", 7 "link": "http://www.baidu.com", 8 "status": 0 9 }, 10 { 11 "name": "step2", 12 "link": "http://www.example.com", 13 "status": 0 14 }, 15 { 16 "name": "step3", 17 "link": "", 18 "status": 0 19 }, 20 { 21 "name": "step4", 22 "link": "", 23 "status": 0 24 } 25] 26 } </pre>
-----	---	--

POST	api/checklist/ 66 1e4be0e977d7 711c301ae9/	<pre> 1 { 2 "name": "template2", 3 "task": [4 { 5 "name": "step1", 6 "link": "http://www.baidu.com", 7 "status": 0 8 }, 9 { 10 "name": "step2", 11 "link": "http://www.example.com", 12 "status": 0 13 }, 14 { 15 "name": "step3", 16 "link": "", 17 "status": 0 18 }, 19 { 20 "name": "step4", 21 "link": "", 22 "status": 0 23 } 24] 25 } </pre>	To create a checklist, you need use a valid template ID
------	---	---	---

			<pre> 21 "link": "", 22 "status": 0 23 } 24], 25 "_id": "661e4c8de977d7711c301aee" 26 }</pre>	
PUT	api/checklist/661e4c8de977d7711c301aee/		<pre> 1 { 2 "name": "temp" 3 "task": [4 { 5 "name": "link" 6 "link": "http://example.com/task1" 7 "status": 0 8 }, 9 { 10 { 11 "name": "link" 12 "link": "http://example.com/task2" 13 "status": 0 14 }, 15 { 16 "name": "link" 17 "link": "http://example.com/task3" 18 "status": 0 19 }, 20 { 21 "name": "link" 22 "link": "http://example.com/task4" 23 "status": 0 24 } 25] 26 }</pre>	<pre> 1 { 2 "status": "success", 3 "id": "661e4c8de977d7711c301aee", 4 "updated": 1 5 }</pre>
DELETE	api/checklist/6603d23b9e45d1f805c5fdd1/		<pre> 1 { 2 "status": "success", 3 "id": "6603d23b9e45d1f805c5fdd1" 4 }</pre>	

Booking

Method	URL Example	Request Body	Response	Description
GET	api/booking/		<pre> 1 [2 { 3 "id": "661633364ee0ba663dcf23c3", 4 "name": "Annual Science Fair", 5 "event": "Science Fair 2024", 6 "status": "Processing", 7 "location": "Exhibition Center - Hall A", 8 "date": "2024-09-05T09:00:00Z", 9 "checklist_id": "661631c15ca3f59d6", 10 "startTime": "2024-09-05T10:00:00Z", 11 "endTime": "2024-09-05T16:00:00Z", 12 "duration": "06:00:00" 13 } 14] 15 }</pre>	

```

13     "school_id": "661630a9577bed19d130",
14     "exhibition": "Local Innovators Show",
15     "note": "Extra chairs needed in the hall",
16   },
17   {
18     "id": "661633751a32e0d0bd170449",
19     "name": "Annual Science Fair",
20     "event": "Science Fair 2024",
21     "status": "Processing",
22     "location": "Exhibition Center - Hall A",
23     "date": "2024-09-05T09:00:00Z",
24     "checklist_id": "661e4fdae977d7711c301af",
25     "checklist": [
26       {
27         "id": "661e4fdae977d7711c301af",
28         "name": "Paid Workshop",
29         "task": [
30           {
31             "name": "Planner (from LER & Roster)",
32             "link": "",
33             "status": 0
34           },
35           {
36             "name": "Make initial contact with exhibitors",
37             "link": "",
38             "status": 0
39           },
40           {
41             "name": "Note: Take Feedback from exhibitors",
42             "link": "",
43             "status": 0
44           },
45           {
46             "name": "If struggling to get responses, follow up",
47             "link": "",
48             "status": 0
49           },
50           {
51             "name": "Update holds on roster based on responses",
52             "link": "https://apac-schools.com/roster-updates",
53             "status": 0
54           },
55           {
56             "name": "Add to LER & Roster if not already present",
57             "link": "",
58             "status": 0
59           },
60           {
61             "name": "Respond with thanks and next steps",
62             "link": "",
63             "status": 0
64           },
65           {
66             "name": "If Future Focus, add to backlog",
67             "link": "",
68             "status": 0
69           },
70           {
71             "name": "KE to roster for follow-up",
72             "link": "",
73             "status": 0
74         ]
75       }
76     ]
77   }
78 }
```

```

73         },
74         {
75             "name": "Check in with",
76             "link": "",
77             "status": 0
78         }
79     ],
80     {
81         "startTime": "2024-09-05T10:00:00Z",
82         "endTime": "2024-09-05T16:00:00Z",
83         "school_id": "661630a9577bed19d130",
84         "school": {
85             "name": "Greenwood Elementary",
86             "studentYear": 2020,
87             "numStudentAttended": 150,
88             "numStudentRegistered": 200,
89             "hourRegistered": 1200,
90             "hourAttended": 1100,
91             "lowSES": true,
92             "allergy": "Peanuts",
93             "contactFirstName": "Jane",
94             "contactLastName": "Doe",
95             "email": "jane.doe@greener.com",
96             "phone": "555-1234",
97             "note": "",
98             "isAccessibility": false,
99             "isAllergy": false,
100            "isPartner": false
101        },
102        "exhibition": "Local Innovators Show",
103        "note": "Extra chairs needed in the hall"
104    }
105 ]

```

GET

api/booking/6603d23b9e45d1f805c5fdd1/

```

1  {
2      "id": "661633364ee0ba663dcf23c3",
3      "name": "Annual Science Fair",
4      "event": "Science Fair 2024",
5      "status": "Processing",
6      "location": "Exhibition Center - Hall A",
7      "date": "2024-09-05T09:00:00Z",
8      "checklist_id": "661631c15ca3f59d6",
9      "startTime": "2024-09-05T10:00:00Z",
10     "endTime": "2024-09-05T16:00:00Z",
11     "school_id": "661630a9577bed19d130",
12     "exhibition": "Local Innovators Show",
13     "note": "Extra chairs needed in the hall"
14 },
15 {
16     "id": "661633751a32e0d0bd170449",
17     "name": "Annual Science Fair",
18     "event": "Science Fair 2024",
19     "status": "Processing",
20     "location": "Exhibition Center - Hall A",
21     "date": "2024-09-05T09:00:00Z",
22     "checklist_id": "661e4fd9e977d7711c301a1",
23     "checklist": [
24         {
25             "id": "661e4fd9e977d7711c301a1",
26             "name": "Paid Workshop",
27             "task": [

```

```

27         },
28         "name": "Planner (from",
29         "link": "",
30         "status": 0
31     },
32     {
33         "name": "Make initial",
34         "link": "",
35         "status": 0
36     },
37     {
38         "name": "Note: Take F"]
39         "link": "",
40         "status": 0
41     },
42     {
43         "name": "If struggling",
44         "link": "",
45         "status": 0
46     },
47     {
48         "name": "Update holds",
49         "link": "https://apac-",
50         "status": 0
51     },
52     {
53         "name": "Add to LER &",
54         "link": "",
55         "status": 0
56     },
57     {
58         "name": "Respond with",
59         "link": "",
60         "status": 0
61     },
62     {
63         "name": "If Future Focu",
64         "link": "",
65         "status": 0
66     },
67     {
68         "name": "KE to roster",
69         "link": "",
70         "status": 0
71     },
72     {
73         "name": "Check in with",
74         "link": "",
75         "status": 0
76     }
77 ]
78 },
79 "startTime": "2024-09-05T10:00:00Z",
80 "endTime": "2024-09-05T16:00:00Z",
81 "school_id": "661630a9577bed19d130",
82 "school": {
83     "name": "Greenwood Elementary",
84     "studentYear": 2020,
85     "numStudentAttended": 150,
86     "numStudentRegistered": 200,

```

		<pre> 87 "hourRegistered": 1200, 88 "hourAttended": 1100, 89 "lowSES": true, 90 "allergy": "Peanuts", 91 "contactFirstName": "Jane", 92 "contactLastName": "Doe", 93 "email": "jane.doe@greenwood.org", 94 "phone": "555-1234", 95 "note": "", 96 "isAccessibility": false, 97 "isAllergy": false, 98 "isPartner": false 99 }, 100 "exhibition": "Local Innovators Showcase", 101 "note": "Extra chairs needed in the lot" 102 } </pre>	
POST	api/booking/	<pre> 1 { 2 "name": "Annual Science Fair", 3 "event": "Science Fair 2024", 4 "status": "Processing", 5 "location": "Exhibition Center - Hall B", 6 "date": "2024-09-05T09:00:00Z", 7 "checklist_id": "661e4fdae977d7711c301aff", 8 "startTime": "2024-09-05T10:00:00Z", 9 "endTime": "2024-09-05T16:00:00Z", 10 "school_id": "661630a9577bed19d130de77", 11 "exhibition": "Local Innovators Showcase", 12 "note": "Extra chairs needed in the lot", 13 "bus": { 14 "bus_req": false, 15 "isBooked": false, 16 "status": 0, 17 "price": 0.0, 18 "date_paid": null, 19 "invoice": null 20 }, 21 "per_student": 0, 22 "expense": 0, 23 "income": 0, 24 "profit": 0 25 } </pre>	checklist_id and school_id has to be valid
PUT	api/booking/6603d23b9e45d1f805c5fdd1/	<pre> 1 { 2 "name": "Annual Science Fair", 3 "event": "Science Fair 2024", 4 "status": "Success", 5 "location": "Exhibition Center - Hall B", 6 "date": "2024-09-05T09:00:00Z", 7 "checklist_id": "661e4fdae977d7711c301aff", 8 "startTime": "2024-09-05T10:00:00Z", 9 "endTime": "2024-09-05T16:00:00Z", 10 "school_id": "661630a9577bed19d130de77", 11 "exhibition": "Local Innovators Showcase", 12 "note": "Extra chairs needed in the lot", 13 "bus": { 14 "bus_req": false, 15 "isBooked": false, 16 "status": 0, 17 "price": 0.0, 18 "date_paid": null, 19 "invoice": null 20 }, 21 "per_student": 0, 22 "expense": 0, 23 "income": 0, 24 "profit": 0 25 } </pre>	

		<pre> 17 "price": 0.0 18 "date_paid": 19 "invoice": " 20 }, 21 "per_student": 0 22 "expense": 0, 23 "income": 0, 24 "profit": 0 25 }</pre>	
DELETE	api/booking/6603d23b9e45d1f805c5fdd1/		<pre> 1 { 2 "status": "success", 3 "id": "66049ef698b69f53ee92f823" 4 }</pre>

School

Method	URL Example	Request Body	Response	Description
GET	api/school/		<pre> 1 [2 { 3 "id": "661f69f37fedbcf1319be202", 4 "name": "Greenwood Elementary123213", 5 "studentYear": 2020, 6 "numStudentAttended": 150, 7 "numStudentRegistered": 200, 8 "hourRegistered": 1200, 9 "hourAttended": 1100, 10 "lowSES": true, 11 "allergy": "Peanuts", 12 "contactFirstName": "Jane", 13 "contactLastName": "Doe", 14 "email": "jane.doe@greenwood.edu", 15 "phone": "555-1234", 16 "note": "", 17 "isAccessibility": false, 18 "isAllergy": false, 19 "isPartner": false 20 } 21]</pre>	
GET	api/school/6603d23b9e45d1f805c5fdd1/		<pre> 1 { 2 "id": "661f69f37fedbcf1319be202", 3 "name": "Greenwood Elementary123213", 4 "studentYear": 2020, 5 "numStudentAttended": 150, 6 "numStudentRegistered": 200, 7 "hourRegistered": 1200, 8 "hourAttended": 1100, 9 "lowSES": true, 10 "allergy": "Peanuts", 11 "contactFirstName": "Jane", 12 "contactLastName": "Doe", 13 "email": "jane.doe@greenwood.edu", 14 "phone": "555-1234",</pre>	

			<pre> 15 "note": "", 16 "isAccessibility": false, 17 "isAllergy": false, 18 "isPartner": false 19 }</pre>	
POST	api/school/	<pre> 1 { 2 "name": "Greenwo 3 "studentYear": 2 4 "numStudentAttende 5 "numStudentRegiste 6 "hourRegistered": 1 7 "hourAttended": 1 8 "lowSES": true, 9 "allergy": "Peanut 10 "contactFirstName": "John", 11 "contactLastName": "Doe", 12 "email": "jane.doe@greenwood.edu", 13 "phone": "555-1234", 14 "note": "", 15 "isAccessibility": false, 16 "isAllergy": false, 17 "isPartner": false 18 }</pre>	<pre> 1 { 2 "name": "Greenwood Elementary123", 3 "studentYear": 2020, 4 "numStudentAttended": 150, 5 "numStudentRegistered": 200, 6 "hourRegistered": 1200, 7 "hourAttended": 1100, 8 "lowSES": true, 9 "allergy": "Peanuts", 10 "contactFirstName": "Jane", 11 "contactLastName": "Doe", 12 "email": "jane.doe@greenwood.edu", 13 "phone": "555-1234", 14 "_id": "661e5111e977d7711c301b01" 15 }</pre>	
PUT	api/school/661 630a9577bed1 9d130de77/	<pre> 1 { 2 "name": "Greenwo 3 "studentYear": 2 4 "numStudentAttende 5 "numStudentRegiste 6 "hourRegistered": 1 7 "hourAttended": 1 8 "lowSES": true, 9 "allergy": "Peanut 10 "contactFirstName": "John", 11 "contactLastName": "Doe", 12 "email": "jane.doe@greenwood.edu", 13 "phone": "555-1234", 14 "note": "", 15 "isAccessibility": false, 16 "isAllergy": false, 17 "isPartner": false 18 }</pre>	<pre> 1 { 2 "status": "success", 3 "id": "661e5111e977d7711c301b01", 4 "updated": 1 5 }</pre>	
DELETE	api/school/660 3d23b9e45d1f 805c5fdd1/		<pre> 1 { 2 "status": "success", 3 "id": "66049ef698b69f53ee92f823" 4 }</pre>	

Miscellaneous

Method	URL Example	Request Body	Response	Description
GET	api/miscellaneous/		<pre> 1 { 2 "id": "661f624bce65511a21305a41", 3 "module": [4 "Access Code", 5 "F: Bespoke", 6 "TBC", 7 "TOUR: Nadine Christensen - Facilitated", 8 "TOUR: Not Natural - Facilitated", 9 "VISIT: Ancient Lives - Self-directed", 10 "VISIT: Nadine Christensen - Self-Directed", 11 "W: App It!", 12 "W: Design It!", 13 "W: Future Food", 14 "W: Mission Control", 15 "W: Sustainable Communities", 16 "W: Take Flight" 17], 18 "program_stream": [19 "ART: Excursions", 20 "SCoE: Excursions", 21 "STEAM: Excursions", 22 "STEAM: Special Outreach Projects" 23], 24 "facilitators": [25 "EB", 26 "ER", 27 "JC", 28 "MC", 29 "MeLB", 30 "MK", 31 "Teacher Delivered", 32 "TS", 33 "XC" 34], 35 "delivery_location": [36 "Buxton", 37 "Embedded (25%)", 38 "Incursion", 39 "Old Quad", 40 "SGM: EG, WG", 41 "SGM: SGMT", 42 "SGM: W2", 43 "SGM: W3" 44], 45 "exhibition": [46 "Ancient Lives", 47 "Nadine Christensen", 48 "Non-Exhibition Linked", 49 "Not Natural" 50] 51 }</pre>	
POST	api/miscellaneous/	<pre> 1 { 2 "id": "661f624bce65511a21305a41", 3 "module": [4 "Access Code" </pre>	<pre> 1 { 2 "module": [3 "Access Code", 4 "F: Bespoke", </pre>	

		<pre> 5 "F: Bespoke" 6 "TBC", 7 "TOUR: Nadine Christensen - Facilitated" 8 "TOUR: Not Natural - Facilitated" 9 "VISIT: Ancient Lives - Self-directed" 10 "VISIT: Nadine Christensen - Self-Directed" 11 "W: App It!" 12 "W: Design It!" 13 "W: Future Food" 14 "W: Mission Control" 15 "W: Sustainable Communities" 16 "W: Take Flight" 17], 18 "program_stream": [19 "ART: Excursions", 20 "SCoE: Excursions", 21 "STEAM: Excursions", 22 "STEAM: Special Outreach Projects" 23], 24 "facilitators": [25 "EB", 26 "ER", 27 "JC", 28 "MC", 29 "MelB", 30 "MK", 31 "Teacher Delivered", 32 "TS", 33 "XC" 34], 35 "delivery_location": [36 "Buxton", 37 "Embedded (25%)", 38 "Incursion", 39 "Old Quad", 40 "SGM: EG, WG", 41 "SGM: SGMT", 42 "SGM: W2", 43 "SGM: W3" 44], 45 "exhibition": [46 "Ancient Lives", 47 "Nadine Christensen", 48 "Non-Exhibition Linked", 49 "Not Natural" 50] 51 } </pre>	<pre> 5 "TBC", 6 "TOUR: Nadine Christensen - Facilitated" 7 "TOUR: Not Natural - Facilitated" 8 "VISIT: Ancient Lives - Self-directed" 9 "VISIT: Nadine Christensen - Self-Directed" 10 "W: App It!", 11 "W: Design It!", 12 "W: Future Food", 13 "W: Mission Control", 14 "W: Sustainable Communities", 15 "W: Take Flight" 16], 17 "program_stream": [18 "ART: Excursions", 19 "SCoE: Excursions", 20 "STEAM: Excursions", 21 "STEAM: Special Outreach Projects" 22], 23 "facilitators": [24 "EB", 25 "ER", 26 "JC", 27 "MC", 28 "MelB", 29 "MK", 30 "Teacher Delivered", 31 "TS", 32 "XC" 33], 34 "delivery_location": [35 "Buxton", 36 "Embedded (25%)", 37 "Incursion", 38 "Old Quad", 39 "SGM: EG, WG", 40 "SGM: SGMT", 41 "SGM: W2", 42 "SGM: W3" 43], 44 "exhibition": [45 "Ancient Lives", 46 "Nadine Christensen", 47 "Non-Exhibition Linked", 48 "Not Natural" 49], 50 "_id": "661f624bce65511a21305a41" 51 } </pre>
PUT	api/miscellaneous/	<pre> 1 { 2 "id": "661f624bce65511a21305a41" 3 "module": [4 "Access Codes" 5 "F: Bespoke" 6 "TBC", 7 "TOUR: Nadine Christensen - Facilitated" 8 "TOUR: Not Natural - Facilitated" 9 "VISIT: Ancient Lives - Self-directed" 10 "VISIT: Nadine Christensen - Self-Directed" 11 "W: App It!" 12 "W: Design It!" 13 "W: Future Food" 14 "W: Mission Control" 15 "W: Sustainable Communities" 16 "W: Take Flight" 17], 18 "program_stream": [19 "ART: Excursions", 20 "SCoE: Excursions", 21 "STEAM: Excursions", 22 "STEAM: Special Outreach Projects" 23], 24 "facilitators": [25 "EB", 26 "ER", 27 "JC", 28 "MC", 29 "MelB", 30 "MK", 31 "Teacher Delivered", 32 "TS", 33 "XC" 34], 35 "delivery_location": [36 "Buxton", 37 "Embedded (25%)", 38 "Incursion", 39 "Old Quad", 40 "SGM: EG, WG", 41 "SGM: SGMT", 42 "SGM: W2", 43 "SGM: W3" 44], 45 "exhibition": [46 "Ancient Lives", 47 "Nadine Christensen", 48 "Non-Exhibition Linked", 49 "Not Natural" 50] 51 } </pre>	<pre> 1 { 2 "status": "success", 3 "updated": 1 4 } </pre>

```
13         "W: Future F
14         "W: Mission
15         "W: Sustainab
16         "W: Take Fl
17     ],
18     "program_stream"
19     "ART: Excursu
20     "SCoE: Excurs
21     "STEAM: Excu
22     "STEAM: Specia
23 ],
24     "facilitators":
25     "EB",
26     "ER",
27     "JC",
28     "MC",
29     "MelB",
30     "MK",
31     "Teacher Devel
32     "TS",
33     "XC"
34 ],
35     "delivery_locati
36     "Buxton",
37     "Embedded (201
38     "Incursion",
39     "Old Quad",
40     "SGM: EG, W1"
41     "SGM: SGMT",
42     "SGM: W2",
43     "SGM: W3"
44 ],
45     "exhibition": [
46     "Ancient Lives"
47     "Nadine Christi
48     "Non-Exhibitio
49     "Not Naturalis
50   ]
51 }
```

Analytics

Chart 1: Number of Participants and Registrants Each Month

- **Type:** Bar Chart
- **X-Axis:** Month (e.g., January, February, etc.)
- **Y-Axis:** Number of People
- **Parameters:**
 - Participants (actual attendees)
 - Registrants (those who registered)

API

```
1 /api/chart/1/
```

Result

```
1 {
2   "participants": [120, 110, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10],
3   "registrants": [240, 220, 200, 180, 160, 140, 120, 100, 80, 60, 40, 20]
4 }
```

Note

It sums the all numbers of participants and registrants in all times.

Chart 2: Number of Participants in Each Term by Stream

- **Type:** Line Graph
- **X-Axis:** Term (e.g., Term 1, Term 2, etc.)
- **Y-Axis:** Number of Participants
- **Parameters:**
 - Multiple streams (e.g., Stream A, Stream B, etc.), each represented by a different line

API

```
/api/chart/2/
```

Result

```
1 {
2   "streams": {
3     "SCOE": [
4       0,
5       578,
6       0,
7       0
8     ],
9     "STEAM": [
10       0,
```

```

11      578,
12      0,
13      0
14  ],
15 "NN Tour+ (UN)EXPECTED WORKSHOP + CHICKENOSAURUS WORKSHOP": [
16      0,
17      0,
18      0,
19      0
20  ],
21 "ART": [
22      0,
23      867,
24      0,
25      0
26  ]
27 }
28 }
```

Note

Every location will contain an array for the number of attendants [`#term_1, #term_2, #term_3, #term_4`].

Chart 3: Number of Users by Time Period for Different Locations

- **Type:** Line Graph
- **X-Axis:** Time Period (Term)
- **Y-Axis:** Number of Users
- **Parameters:**
 - Multiple locations (e.g., Location A, Location B, etc.), each represented by a different line

API

```
/api/chart/3/
```

Result

```

1 {
2   "locations": {
3     "D R003": [
4       0,
5       289,
6       0,
7       0
8     ],
9     "D ROOM2": [
10      0,
11      289,
12      0,
13      0
14    ],
15     "Buxton": [
16       0,
17       0,
18       0,
```

```

19         0
20     ],
21     "D R0043": [
22         0,
23         289,
24         0,
25         0
26     ],
27     "D ROOM1": [
28         0,
29         578,
30         0,
31         0
32     ],
33     "D ROOM3": [
34         0,
35         578,
36         0,
37         0
38 ]
39 }
40 }
```

Chart 4: Grade Distribution for Each Stream

- **Type:** Pie Chart
- **X-Axis:** Not applicable (as it is a pie chart)
- **Y-Axis:** Not applicable (as it is a pie chart)
- **Parameters:**
 - Grades (7 to 12)
- **Filters:**
 - Stream (selectable from different available streams to view specific data)

API

/api/chart/4/

Result

```

1   "grades_by_stream": {
2     "ART": {
3       "7": 867
4     },
5     "NN Tour+ (UN)EXPECTED WORKSHOP + CHICKENOSAURUS WORKSHOP": {
6       "7": 0
7     },
8     "STEAM": {
9       "7": 578
10    },
11    "SCOE": {
12      "7": 578
13    }
14 }
```

TESTS

Test Directory Outline

I. Backend Testing

- **Location:** /src/back-end/api/tests
- **Purpose:** Validates the server-side APIs, ensuring they handle requests and return expected results for all CRUD operations.

1. Unit Test - GET

- **Description:** Tests all GET requests to fetch data from the server.
- **Test Cases Include:** - Fetching all records. - Fetching a single record by ID. - Fetching with queries and filters.

2. Unit Test - PUT

- **Description:** Tests PUT requests for updating existing data.
- **Test Cases Include:** - Updating records with valid data. - Handling updates with invalid or incomplete data.

3. Unit Test - DELETE

- **Description:** Tests DELETE requests to remove data.
- **Test Cases Include:** - Deleting a record by ID. - Ensuring records are not accessible post-deletion.

4. Unit Test - CREATE

- **Description:** Tests POST requests for creating new records.
- **Test Cases Include:** - Creating records with valid data. - Handling attempts to create records with invalid data.

II. Frontend Testing

- **Location:** /src/front-end/src/__tests__
- **Purpose:** Ensures that the user interface behaves as expected under various scenarios and interactions.

1. Unit Test - Dashboard

- **Description:** Tests the Dashboard component's responsiveness and functionality.
- **Test Cases Include:**
 - Rendering essential elements and UI components.
 - Interactions with filter options (e.g., type, date, location).
 - Sorting and displaying data based on user selections.

2. Unit Test - NewBooking

- **Description:** Tests the NewBooking component's ability to handle form inputs, validate data, and submit new bookings.
- **Test Cases Include:**
 - Switching between different booking types (Delivery, School, Bus, Others).
 - Correctly capturing and validating user inputs.
 - Submitting forms and handling the responses.

3. Unit Test - Templates

- **Description:** Verifies the functionality of the Templates component, focusing on its ability to list, add, and manage template items effectively.
- **Test Cases Include:**
 - Loading and displaying templates from a data fetch.

- Interactions with the UI to add new templates.
- Handling user actions such as selecting and viewing template details.

4. Unit Test - TemplateDetail

- **Description:** Focuses on testing the TemplateDetail component, ensuring it handles detailed interactions for creating and editing templates.
- **Test Cases Include:**
 - Loading and displaying individual template details.
 - Adding, updating, and deleting tasks within a template.
 - Validating user input and managing state transitions.
 - Reacting to user actions like save, delete, and discard changes.

Backend - Testing

Test Document: MongoDB Integration and API Response Test

1. Introduction

- This document outlines the test strategy for the `DBTest` class, which is responsible for validating the integration of MongoDB with a Django application and the correct responses of the application's REST API endpoints.

2. Scope

- This test plan covers the following:
 - Connection to the MongoDB database using the Django settings configuration.
 - Retrieval of data from specific API endpoints to ensure the expected responses are received for both successful and erroneous scenarios.

3. Objectives

- The main objectives of these tests are:
 - To confirm that the application can connect to and interact with MongoDB correctly.
 - To verify that the API responds with the correct status codes and data for various endpoints when accessed with correct and incorrect parameters.

4. Roles and Responsibilities

- **Test Engineer** (@Guixian Li) : Responsible for writing, executing, and maintaining the test cases.
- **Developer** (@Yuanbo Xu) : To review test failures and address any bugs or issues in the code.
- **Quality Assurance (one of the team members)**: Oversees the testing process and ensures quality standards.

5. Testing Strategy

5.1 Test Environment

- The tests will be executed in a controlled development environment with access to a MongoDB instance configured through Django settings.

5.2 Test Data

- Test data will include predefined identifiers for querying the API endpoints.
- Examples of valid data can be found in the team's GitHub repository  [LINKS & RESOURCES](#)

5.3 Types of Tests

- **Unit Tests**: To validate individual functions and API.
- Test Execution Schedule

The tests will be performed during the completion of every feature of the development cycle, before deployment. The expected duration of the testing phase is one week.

1. Risks and Contingencies

- **Risk**: Incorrect configuration of the database connection.
- **Mitigation**: Review and verify configuration settings before executing tests.
- **Risk**: Incomplete implementation of API endpoints.
- **Mitigation**: Coordinate with developers to ensure all endpoints are implemented before testing.

2. Defect Management

Defects found during testing will be logged into the project's issue tracker with severity, steps to reproduce, and screenshots if applicable. The development team will prioritize defects based on severity.

3. Metrics and Reporting

Test results will be compiled into a report showing:

- Number of tests executed.
- Number of tests passed/failed.
- Detailed logs for any failures.

4. Approval

Upon completion, the test report will be reviewed and approved by the rest of the team.

Coverage Score = 4/6

Unit Tests - GET

Test Cases:

Test Case 1: Retrieve All Documents

Objective: Validate that the system can retrieve all documents from each collection and return a HTTP 200 OK status.

Method: `get_all`

Test Steps:

1. Ensure each collection (`school`, `checklist`, `template`, `booking`) has pre-populated data.
2. Call `get_all` for each URI corresponding to the collections.
3. Assert that the HTTP response status is 200 OK for each collection.

Expected Result: The API should successfully retrieve all documents from each specified collection, and the response should be HTTP 200 OK.

Test Case 2: Retrieve Document by ID - Success

Objective: Ensure the system can correctly retrieve a document by its ID and return a HTTP 200 OK status.

Method: `get_by_id`

Test Steps:

1. Insert a document with a known ID into each collection.
2. Call `get_by_id` with the known ID for each document.
3. Assert the HTTP response status is 200 OK.
4. Verify the content of the response matches the requested document.

Expected Result: The system should successfully retrieve the document by ID from each collection, and the response should be HTTP 200 OK.

Test Case 3: Retrieve Document by ID - Not Found

Objective: Test the system's response when attempting to retrieve a document using a non-existent ID.

Method: `get_by_id_error`

Test Steps:

1. Use an ID that does not exist in the database.
2. Call `get_by_id_error` for each collection URI with the non-existent ID.
3. Assert the HTTP response status is 404 Not Found.

Expected Result: The system should return HTTP 404 Not Found for each invalid ID request.

Unit Tests - PUT

Test Cases:

Test Case 1: Successful Update by ID

Objective: Validate that documents can be successfully updated in the database and return a HTTP 200 OK status.

Method: `update_by_id`

Test Steps:

1. Insert initial documents into the respective collection.
2. Construct a valid update request.
3. Call `update_by_id` with the valid ID and request data.
4. Assert the HTTP response status is 200 OK.
5. Verify the document's updated fields in the database.

Expected Result: The document should be updated successfully, and the response should be HTTP 200 OK.

Test Case 2: Update by ID with Bad Request

Objective: Ensure the system correctly handles an update attempt with invalid data by returning a HTTP 400 Bad Request status.

Method: `update_by_id_bad_request`

Test Steps:

1. Construct an update request with invalid data fields that violate schema or constraints.
2. Call `update_by_id_bad_request` with the valid ID but invalid request data.
3. Assert the HTTP response status is 400 Bad Request.

Expected Result: The system should not update the document and return HTTP 400 Bad Request.

Test Case 3: Update by ID Not Found

Objective: Verify that attempting to update a non-existent document ID results in a HTTP 404 Not Found status.

Method: `update_by_id_not_found`

Test Steps:

1. Construct a valid update request.
2. Call `update_by_id_not_found` with a non-existent document ID.
3. Assert the HTTP response status is 404 Not Found.

Expected Result: The system should return HTTP 404 Not Found, indicating the document does not exist.

Unit Tests - DELETE

Test Cases:

Test Case 1: Delete Document by ID - Success

Objective: Validate that the system can successfully delete a document by its ID and return a HTTP 200 OK status.

Method: `delete_by_id`

Test Steps:

1. Pre-insert a document into MongoDB with a known ID for each collection (school, checklist, template, booking).
2. Call `delete_by_id` using the known IDs for each collection.
3. Assert that the HTTP response status is 200 OK.
4. Optionally, verify that the document no longer exists in the database.

Expected Result: Each document is successfully deleted, and the API returns HTTP 200 OK.

Test Case 2: Delete Document by ID - Not Found

Objective: Test the system's response when attempting to delete a document using a non-existent ID.

Method: `delete_by_id_error`

Test Steps:

1. Use an ID that does not exist in the database for each collection.
2. Call `delete_by_id_error` for each collection URI with the non-existent ID.
3. Assert the HTTP response status is 404 Not Found.

Expected Result: The system should return HTTP 404 Not Found for each invalid ID request.

Unit Tests - CREATE

Test Cases

Test Case 1. Test Insert Operations

Objective:

To verify the successful creation of documents in the MongoDB database through the API for various collections (school, template, checklist, booking).

Method: `test_insertion`

Steps:

1. Insert a "school" document and verify if the response status is HTTP 201_CREATED.
2. Check if the document count in the school collection has increased by 1.
3. Repeat similar steps for "template", "checklist" (linked with template), and "booking" (linked with checklist and school) documents.
4. Verify the correctness of response data like `_id`.

Expected Results: Each insertion should result in an HTTP 201_CREATED status, and the count of documents in respective collections should increase by one.

Test Case 2. Test Insert Error Handling

Objective:

To ensure that the API properly handles errors during the insertion process due to incomplete or incorrect data.

Methods: `test_school_insert_error`, `test_template_insert_error`, `test_checklist_insert_error`,
`test_booking_insert_error`

Steps for Each Method:

1. Attempt to insert an incomplete or incorrectly formatted document into the respective collection.
2. Verify that the response status is appropriate for the error (e.g., HTTP_400_BAD_REQUEST for incomplete data, HTTP_404_NOT_FOUND for invalid references).

Expected Results: The API should return an error status code corresponding to the nature of the error (400 for bad requests, 404 for not found).

Unit Tests - Search

Test Cases

Test Case 1: Search Query - Success

Objective:

Ensure the system can handle a valid search query and return an HTTP 200 OK status.

Method:

```
test_search_query
```

Test Steps:

1. Define a sample search query with the key 'query' and value 'geelong'.
2. Send a GET request to the search endpoint with the query as a parameter.
3. Assert that the HTTP response status is 200 OK.

Expected Result:

The system should successfully process the search query and return an HTTP 200 OK status.

Test Case 2: Search Query - Missing Parameter

Objective:

Test the system's response when a search query parameter is not provided.

Method:

```
test_search_non_exist
```

Test Steps:

1. Send a GET request to the search endpoint without any query parameters.
2. Assert that the HTTP response status is 400 Bad Request.
3. Assert that the response contains an error message indicating that no query parameter was provided.

Expected Result:

The system should return an HTTP 400 Bad Request status and an error message stating "No query parameter provided."

Unit Tests - Miscellaneous DB Functions

Test Case 1: Create Document

Objective:

Ensure the system can handle the creation of a document in the `miscellaneous` collection and return an HTTP 201 Created status.

Method:

```
test_create
```

Test Steps:

1. Define the data to be inserted into the `miscellaneous` collection.
2. Send a POST request to the `miscellaneous` endpoint with the defined data.
3. Assert that the HTTP response status is 201 Created.
4. Count the number of documents in the `miscellaneous` collection.
5. Assert that the document count is 1.

Expected Result:

The system should successfully create a document in the `miscellaneous` collection, return an HTTP 201 Created status, and the collection should contain exactly one document.

Test Case 2: Retrieve All Documents

Objective:

Ensure the system can retrieve all documents from the `miscellaneous` collection and return an HTTP 200 OK status.

Method:

```
test_get_all
```

Test Steps:

1. Send a GET request to the `miscellaneous` endpoint.
2. Assert that the HTTP response status is 200 OK.

Expected Result:

The system should successfully retrieve all documents from the `miscellaneous` collection, and the response should be HTTP 200 OK.

Test Case 3: Update Document - Success

Objective:

Ensure the system can handle the update of a document in the `miscellaneous` collection and return an HTTP 200 OK status.

Method:

```
test_put
```

Test Steps:

1. Define the data to update the document in the `miscellaneous` collection.
2. Send a PUT request to the `miscellaneous` endpoint with the defined data.
3. Assert that the HTTP response status is 200 OK.

Expected Result:

The system should successfully update the document in the `miscellaneous` collection, and the response should be HTTP 200 OK.

Test Case 3: Update Document - Invalid Data

Objective:

Test the system's response when attempting to update a document with incomplete or invalid data.

Method:

```
test_put_not_valid
```

Test Steps:

1. Define the invalid data to update the document in the `miscellaneous` collection.
2. Send a PUT request to the `miscellaneous` endpoint with the invalid data.
3. Assert that the HTTP response status is 400 Bad Request.

Expected Result:

The system should return an HTTP 400 Bad Request status due to the incomplete or invalid data provided for the update.

Unit Tests - Charts

Test Case 1: Retrieve Chart 1 Data

Objective:

Ensure the system can retrieve data for Chart 1 and return an HTTP 200 OK status.

Method:

```
test_chart_1
```

Test Steps:

1. Send a GET request to the `chart_1` endpoint.
2. Assert that the HTTP response status is 200 OK.

Expected Result:

The system should successfully retrieve the data for Chart 1, and the response should be HTTP 200 OK.

Test Case 2: Retrieve Chart 2 Data

Objective:

Ensure the system can retrieve data for Chart 2 and return an HTTP 200 OK status.

Method:

```
test_chart_2
```

Test Steps:

1. Send a GET request to the `chart_2` endpoint.
2. Assert that the HTTP response status is 200 OK.

Expected Result:

The system should successfully retrieve the data for Chart 2, and the response should be HTTP 200 OK.

Test Case 3: Retrieve Chart 3 Data

Objective:

Ensure the system can retrieve data for Chart 3 and return an HTTP 200 OK status.

Method:

```
test_chart_3
```

Test Steps:

1. Send a GET request to the `chart_3` endpoint.
2. Assert that the HTTP response status is 200 OK.

Expected Result:

The system should successfully retrieve the data for Chart 3, and the response should be HTTP 200 OK.

Test Case 4: Retrieve Chart 4 Data

Objective:

Ensure the system can retrieve data for Chart 4 and return an HTTP 200 OK status.

Method:

```
test_chart_4
```

Test Steps:

1. Send a GET request to the `chart_4` endpoint.
2. Assert that the HTTP response status is 200 OK.

Expected Result:

The system should successfully retrieve the data for Chart 4, and the response should be HTTP 200 OK.

Frontend - Testing

Test Document for Dashboard Page (Frontend)

Overview

This document describes the frontend tests implemented to validate the functionality and UI of the Dashboard page within the application. These tests ensure the page correctly interacts with user inputs, properly filters and displays data, and responds with the expected UI changes.

Test Environment

- **Framework:** React
- **Testing Libraries:** `@testing-library/react`, `@testing-library/user-event`, `@testing-library/jest-dom`
- **Additional Tools:** React Router (BrowserRouter for navigation context)

Coverage Score = 4/6

Unit Test - Dashboard

Test Cases

Test Case 1: Render Dashboard Components Correctly

- **Objective:** Validate that the Dashboard component correctly renders all its initial UI elements.
- **Method:** `render`
- **Test Steps:**
 - a. Render the Dashboard component within a `BrowserRouter`.
 - b. Check for the presence of essential elements such as booking type filters and location filters.
- **Expected Result:** The "Booking", "All Types", and "All Locations" text elements are visible upon initial render.

Test Case 2: Toggle Active Class on Filter Buttons

- **Objective:** Ensure that clicking on filter buttons updates their visual state to reflect the current selection.
- **Method:** `userEvent.click`
- **Test Steps:**
 - a. Render the Dashboard component.
 - b. Simulate user clicks on the "Completed" filter button.
 - c. Verify that the "Completed" button receives an "active" class.
 - d. Verify that previously active buttons lose the "active" class.
- **Expected Result:** Only the clicked (Completed) button should have the "active" class, ensuring correct visual feedback for user interactions.

Test Case 3: Filter Bookings by Type

- **Objective:** Verify that the booking type filter correctly alters the displayed bookings.
- **Method:** `fireEvent.change`
- **Test Steps:**
 - a. Render the Dashboard component.
 - b. Select a specific booking type from the dropdown.
 - c. Verify that the internal state of the component reflects this selection.
- **Expected Result:** The component should only display bookings of the selected type, and the dropdown should show the selected value.

Test Case 4: Filter Bookings by Date Range

- **Objective:** Ensure the date range filters adjust which bookings are displayed based on the selected range.
- **Method:** `fireEvent.change`
- **Test Steps:**
 - a. Render the Dashboard component.
 - b. Set a start and end date via the date picker inputs.
 - c. Verify the values in the date pickers match the entered dates.
 - d. Optionally, check if displayed bookings fall within the selected date range.
- **Expected Result:** The date pickers should reflect the set dates, and the displayed bookings should be limited to those within the specified range.

Test Case 5: Sort Bookings by Date

- **Objective:** Test the functionality of the sorting feature for displaying bookings in ascending or descending order by date.
- **Method:** `userEvent.click`

- **Test Steps:**
 - a. Render the Dashboard component.
 - b. Click the sort button to toggle between ascending and descending order.
 - c. Verify that the button text updates to reflect the sort order.
 - d. Optionally, verify that bookings are sorted correctly in the UI.
- **Expected Result:** The sort button should toggle between "Sort Date Ascending" and "Sort Date Descending," and bookings should reorder accordingly.

Test Case 6: Filter Bookings by Location

- **Objective:** Confirm that the location filter effectively narrows down the displayed bookings based on selected locations.
- **Method:** `userEvent.selectOptions`
- **Test Steps:**
 - a. Render the Dashboard component.
 - b. Select a location from the dropdown menu.
 - c. Verify that only bookings from the selected location are displayed.
- **Expected Result:** The dashboard should only show bookings located in the chosen venue, with all displayed bookings meeting this criterion.

Unit Test - NewBooking

Test Cases

Test Case 1: Render and Category Switching

- **Objective:** Validate that the NewBooking component renders correctly and can switch between different category forms without errors.
- **Method:** `render`
- **Test Steps:**
 - a. Render the NewBooking component.
 - b. Verify that the initial category "Delivery" is displayed correctly by checking for the presence of "Program Stream:".
- **Expected Result:** The component should display elements specific to the "Delivery" category initially, indicating correct rendering and category state initialization.

Test Case 2: Handle School Form Input Correctly

- **Objective:** Ensure that the form for the "School" category accepts and correctly handles user input, including text inputs and toggling checkboxes.
- **Method:** `userEvent.click`, `userEvent.type`
- **Test Steps:**
 - a. Click on the "School" category button to switch forms.
 - b. Type into the "School Name" input field and verify the input retains the typed value.
 - c. Toggle a year checkbox and verify its checked state.
- **Expected Result:** The "School Name" input should reflect the typed text, and the year checkbox should maintain its toggled state, confirming correct state management and user input handling.

Test Case 3: Submit the Bus Form Correctly

- **Objective:** Test the submission of the "Bus" form, ensuring inputs accept values and the form can submit without issues.
- **Method:** `userEvent.type`, `userEvent.click`
- **Test Steps:**
 - a. Activate the "Bus" category and fill out required fields including "BUS REQ" and status.
 - b. Select an option from the status dropdown and submit the form.
 - c. Optionally, verify if any callbacks or effects occur upon submission.
- **Expected Result:** All form fields should accept and display the entered values, and clicking the submit button should trigger any defined actions or state changes related to form submission.

Test Case 4: Manage State Changes in the 'Others' Form

- **Objective:** Verify that the "Others" form correctly manages state changes when user input is provided, particularly for numerical inputs.
- **Method:** `userEvent.type`
- **Test Steps:**
 - a. Switch to the "Others" category.
 - b. Input a value into the "Profit" field and verify the field reflects this input.
- **Expected Result:** The "Profit" field should correctly update and retain the value input by the user, demonstrating effective state handling and responsiveness of the form.

Unit Test - Templates

Test Cases

Test Case 1: Display Loading Indicator

- **Objective:** Validate that the Templates component correctly displays a loading indicator while fetching data.
- **Method:** `render`
- **Test Steps:**
 - a. Mock the API to delay the response.
 - b. Render the Templates component within a BrowserRouter.
 - c. Check for the presence of the "Loading..." text immediately after rendering.
- **Expected Result:** The "Loading..." text is visible upon initial render, indicating that data fetching is in progress.

Test Case 2: Display Templates After Fetching

- **Objective:** Ensure that the Templates component correctly displays templates after data is fetched.
- **Method:** `waitFor`
- **Test Steps:**
 - a. Mock the API to resolve with a list of templates.
 - b. Render the Templates component.
 - c. Wait for the templates to be displayed.
- **Expected Result:** The component should display each template's name, confirming that data is successfully fetched and rendered.

Test Case 3: Handle Fetching Errors

- **Objective:** Verify that the Templates component handles errors during the data fetching process.
- **Method:** `waitFor`
- **Test Steps:**
 - a. Mock the API to reject with an error.
 - b. Render the Templates component.
 - c. Wait for an error message to be displayed.
- **Expected Result:** An error message "[ERROR] Failed to fetch templates." is shown, indicating proper error handling.

Test Case 4: Navigate on Template Selection

- **Objective:** Test the navigation functionality when a template is selected.
- **Method:** `userEvent.click`
- **Test Steps:**
 - a. Render the Templates component with a list of templates.
 - b. Simulate a user click on one of the template buttons.
 - c. Check the navigation function is called with the correct path.
- **Expected Result:** Upon clicking a template button, the application navigates to the detailed view of the selected template.

Unit Test - TemplateDetail

Test Case 1: Handles Task Addition

- **Objective:** Verify that the system correctly handles the addition of new tasks.
- **Method:** `fireEvent`
- **Test Steps:**
 - a. Mock the API call to return an initial state without tasks.
 - b. Render the `TemplateDetail` component within a `Router`.
 - c. Find and click the "Add Task" button.
 - d. Fill in the task name and link inputs.
 - e. Check if the inputs reflect the entered values.
- **Expected Result:**
 - The input for the task name should display "Test Task".
 - The input for the link should display "<http://newlink.com>".

Test Case 2: Updates Tasks on Blur

- **Objective:** Ensure that task updates are handled correctly when an input field is blurred.
- **Method:** `fireEvent`
- **Test Steps:**
 - a. Render the `Item` sub-component with initial task details.
 - b. Modify the task name input.
 - c. Trigger a blur event on the task name input.
 - d. Verify that the update function is called with the new task details.
- **Expected Result:**
 - The `onTaskUpdate` function should be called with the updated task name "Updated Task" and the original link.

Test Case 3: Deletes a Task

- **Objective:** Confirm that tasks can be deleted successfully.
- **Method:** `fireEvent`
- **Test Steps:**
 - a. Render the `Item` sub-component with a deletable task.
 - b. Click the delete button associated with the task.
 - c. Verify that the removal function is called for the correct task index.
- **Expected Result:**
 - The `onTaskRemove` function should be called with the index `0`, indicating the first task is targeted for deletion.

Unit Test - Calendar

Test Case 1: Verify Initial Loading Indicator

- **Objective:** Confirm that the Calendar component displays a loading indicator or a similar temporary message while it is fetching the event data.
- **Method:** `render`, `findByText`
- **Test Steps:**
 - a. Mock the `getAllBooking` API to simulate a delay in fetching data.
 - b. Render the `Calendar` component.
 - c. Use `findByText` to check for a temporary loading message immediately after rendering.
- **Expected Result:**
 - o A temporary loading message or indicator should be visible immediately after rendering, confirming that the component is in the process of fetching data.

Test Case 2: Display Calendar with Events After Data Fetch

- **Objective:** Ensure that the Calendar component correctly displays events after the API has fetched data.
- **Method:** `render`, `findByRole`
- **Test Steps:**
 - a. Mock the `getAllBooking` API to resolve with a predefined list of events.
 - b. Render the `Calendar` component.
 - c. Use `findByRole` to locate the specific date cell for "10 May 2024".
 - d. Verify that this cell contains the correct events data as expected based on the mock.
- **Expected Result:**
 - o The date cell for "10 May 2024" is present and contains the events as specified by the mock data, demonstrating that the component successfully fetches and renders the data.



CODE REVIEW

1. Frontend Code Review:

- How did the team perform **Frontend Section code review** - who participated in that, when did that happen, number of issues identified

2. Backend Code Review:

- How did the team perform **Backend Section code review** - who participated in that, when did that happen, number of issues identified

3. When to do Code Review:

- Whenever we want to do merge request.

4. Criteria for reviewing code:

- Functional Completeness
 - Ensure new features are completed as required.
 - Verify that the feature meets the user story or acceptance criteria.
- Code quality:
 - Follow coding standards and style guides.
 - Is the code easy to understand and maintain?
 - Use appropriate naming conventions.
 - Make sure there is no redundant or useless code
- Conflict Resolution
 - Before merging, resolve any code conflicts.
 - Ensure that the merged code executes without errors.
- Compatibility with existing functionality
 - Make sure new features don't break existing functionality.
- Performance
 - Check whether new features have a negative impact on application performance.
 - Assess performance bottlenecks and recommend optimisations.

Frontend Code Review Sprint #2

Total code review times: 15 (sprint 2)

Code Review Front-end #1 (Merge request #2)

Date: 28/03/2024

Merge Request From **Frontend/features/typography** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check if the font family matches the design
- Check if the font size matches the design (Difference handing might use different font size)
- Check if the colour and contrast matches the design

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #2 (Merge request #2)

Date: 28/03/2024

Merge Request From **Frontend/features/navigationBar** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Details: Review the code of navigation bar written by [@Yongli Qin](#) and check if the layout and structure of the code is correct.

Check list:

- Make sure navigation items are clear and easy to understand.
- Verify that the navigation structure is logical and users can easily find the information they want.
- Validate style changes on mouseover, focus, and activation states to provide user feedback.
- The logo for each option is clear and intuitive

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #3 (Merge request #3)

Date: 09/04/2024

Merge Request From **Frontend/features/dashboard** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- If the whole page is well structured.
- If the font size and font family is unified.
- Different buttons are available and the filter button works correctly.
- Hard code data is displayed correctly in the correct location

Number of Issues : 4

- There is a problem with the CSS of the dashboard page.
- The length of the divider is set too long.
- The page design needs to be improved.
- Code naming is not uniform

Action after code review:

- Redesign the dashboard page to make it more user friendly and modify the css.
- Re-naming the variables and match with the standard.
- Fix the problem of divider

Code Review Front-end #4 (Merge request #4)

Date: 09/04/2024

Merge Request From **Frontend/features/templates** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Whether it conflicts with previous code.
- If the whole page is well structured.
- If the font size and font family is unified.
- Different buttons are available

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #5 (Merge request #5)

Date: 09/04/2024

Merge Request From **Frontend/features/login** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Whether it conflicts with previous code.
- If the whole page is well structured.
- If the font size, font family and colour is unified.
- Different buttons are available

Number of Issues : 1

- inconsistent naming conventions

Action after code review:

- Fix the conventions of naming

Code Review Front-end #6 (Merge request #6)

Date: 10/04/2024

Merge Request From **Frontend/features/dashboard** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Whether it conflicts with previous code.
- If the whole page is well structured.
- If the font size, font family and colour is unified.
- Different buttons are available
- Check if the previous bugs is fixed

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #7 (Merge request #7)

Date: 10/04/2024

Merge Request From **Frontend/features/login** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Whether it conflicts with previous code.
- If the whole page is well structured.
- If the font size, font family and colour is unified.
- Different buttons are available
- Check if the previous bugs is fixed

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #8 (Merge request #9)

Date: 10/04/2024

Merge Request From **Frontend/features/fix** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check if the previous bugs is fixed

Number of Issues : 1

- Navigation Bar still contains minor bugs, when some of the icon is clicked, the colour should be changed

Action after code review:

- Address the existing bugs and verify whether there are any additional potential issues in the code.

Code Review Front-end #9 (Merge request #10)

Date: 10/04/2024

Merge Request From **Frontend/features/fix** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check if the previous bugs is fixed

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #10 (Merge request #12)

Date: 13/04/2024

Merge Request From **Frontend/features/templates-details** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Whether it conflicts with previous code.
- If the whole page is well structured.
- If the font size, font family and colour is unified.
- Different buttons are available

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #11 (Merge request #13)

Date: 13/04/2024

Merge Request From **Frontend/features/newbooking** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Whether it conflicts with previous code.
- If the whole page is well structured.
- If the font size, font family and colour is unified.
- Different buttons are available

Number of Issues : 1

- Inconsistent naming conventions

Action after code review:

- Fix the conventions of naming

Code Review Front-end #12 (Merge request #15)

Date: 13/04/2024

Merge Request From **Frontend/features/fix** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check if the previous bugs is fixed

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #13 (Merge request #17)

Date: 13/04/2024

Merge Request From **Frontend/features/template-api** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Whether it conflicts with previous code.
- If the whole page is well structured.
- If the font size, font family and colour is unified.
- Different buttons are available

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #14 (Merge request #29)

Date: 01/05/2024

Merge Request From **Frontend/features/dashboardAPI** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check whether the function is completed
- Check if the api can work properly
- Check if the booking will appear on dashboard after creating
- Check if all the information is recored and uploaded

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #15 (Merge request #28)

Date: 01/05/2024

Merge Request From **backend/features/database** to **Main**

Participants: [@Daniel Su](#) [@Yongli Qin](#) [@Yun-Chi Hsiao](#)

Check list:

- Update the database schema
- Make sure the previous database model can work properly

Number of Issues : 0 (Standards compliant)

Action after code review: None

Backend Code Review Sprint #2

Total code review times: 6 (sprint 2)

Code Review Back-end #1 (Merge request #8)

Date: 10/04/2024

Merge Request From **backend/features/rest-api** to **Main**

Participants: [@Guixian Li](#) [@Yuanbo Xu](#) [@Pangfeng ZHENG](#)

Check list:

- Review the code of rest-api written by [@Yuanbo Xu](#)
- Use postman to check if the api is working properly and return the expected result.

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Back-end #2 (Merge request #11)

Date: 10/04/2024

Merge Request From **backend/features/rest-api** to **Main**

Participants: [@Guixian Li](#) [@Yuanbo Xu](#) [@Pangfeng ZHENG](#)

Check list:

- Review the code of db-connection written by [@Pangfeng ZHENG](#)
- Use postman to check if the api is working properly and return the expected result.

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Back-end #3 (Merge request #14)

Date: 17/04/2024

Merge Request From **backend/features/db-unit-test** to **Main**

Participants: [@Guixian Li](#) [@Yuanbo Xu](#) [@Pangfeng ZHENG](#)

Check list:

- Review the code of db-unit-test written by [@Guixian Li](#)
- Check if the unit test is working and produce useful information.

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Back-end #4 (Merge request #16)

Date: 17/04/2024

Merge Request From **backend/features/views** to **Main**

Participants: @Guixian Li @Yuanbo Xu @Pangfeng ZHENG

Check list:

- Review the code of views written by @Yuanbo Xu
- check if all the serializer is generated correct

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Back-end #5 (Merge request #21)

Date: 23/04/2024

Merge Request From **backend/features/views** to **Main**

Participants: @Guixian Li @Yuanbo Xu @Pangfeng ZHENG

Check list:

- Check if the chatgpt code review can work properly.

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Back-end #6 (Merge request #27)

Date: 24/04/2024

Merge Request From **backend/features/deployment** to **Main**

Participants: @Guixian Li @Yuanbo Xu @Pangfeng ZHENG

Check list:

- Check if deployment can work
- Check if the link is active

Number of Issues : 0 (Standards compliant)

Action after code review: None

Front-end check list before doing the merge request

Functional Completeness

- Confirm that new features are fully implemented as required.
- Verify that the feature aligns with the user story or meets the acceptance criteria.

Code Quality

- Ensure adherence to front-end coding standards and style guides.
- Assess if the code is easy to understand and maintain.
- Check for appropriate use of naming conventions.
- Ensure CSS styles are consistent and adhere to design specifications.
- Redundant or unnecessary code has been removed.

Compatibility

- New features perform consistently across different browsers and devices.
- Ensure new features do not disrupt existing front-end functionality.

Performance

- Analyze if new features affect the loading and execution speed of the front-end application.
- Identify and optimize front-end performance bottlenecks, such as image optimization, asynchronous loading, etc.

Back-end check list before doing the merge request

Functional Completeness

- Confirm that new features are fully implemented as required.
- Verify that the functionality aligns with business logic and interface documentation.

Code Quality

- Adhere to back-end coding standards and style guides.
- Assess if the code is easy to understand and maintain.
- Check for appropriate use of naming conventions.
- Redundant or unnecessary code has been removed.

Data Handling

- Ensure data processing logic is correct and interactions with the database are efficient.
- Ensure data security and privacy measures are in place.

Performance

- Analyze if new features affect the processing and response times of the back-end application.
- Identify and optimize back-end performance bottlenecks, such as database query optimizations, caching mechanisms, etc.

AI Code Review

Important: Due to the time, we can't finish all the automated code review (we got the instructions on the morning of 2nd May so we decided to do the automated code Review with the most recent merge request)

Merge Request Feedback 1

Date: 2/05/2024

Participants: @Yongli Qin @Yun-Chi Hsiao @Daniel Su

Code Review Front-end #14 (Merge Request #29)

Feedback from AI (ChatGPT 4)

1. Documentation Defects:

- **Naming:** The variable and function names are generally clear and descriptive, which is good for maintainability and understanding the code's purpose.
- **Comment:** The code lacks comments, which could make it difficult for others (or the original developer at a later time) to understand the reasoning behind certain coding decisions or how parts of the system interact.

2. Visual Representation Defects:

- **Bracket Usage:** Brackets are used correctly throughout the code.
- **Indentation:** The code is properly indented, which helps in maintaining readability.
- **Long Line:** Some lines of code, especially method calls and conditionals, are quite long and could benefit from breaking into multiple lines for better readability.

3. Structure Defects:

- **Dead Code:** There appears to be no dead code in the snippet provided.
- **Duplication:** The code could be refactored to reduce duplication, especially in form fields and select options creation, which is repeated across different forms.

4. New Functionality:

- **Use Standard Method:** The handling of form states and changes can be streamlined using form management libraries like Formik or React Hook Form, which can also handle validation more cleanly.

5. Resource Defects:

- **Variable Initialization:** All variables are initialized appropriately.
- **Memory Management:** React generally handles memory management well, but ensuring cleanup in `useEffect` hooks (like unsubscribing from subscriptions) would be prudent if applicable.

6. Check Defects:

- **Check User Input:** The form does not include validation before submitting data, which can lead to invalid data being processed or even security vulnerabilities.

7. Interface Defects:

- **Parameter:** There are no obvious issues with parameter passing or function calls in terms of incorrect or missing parameters.

8. Logic Defects:

- **Compute:** The logic seems functional, but the actual data manipulations and conditions should be reviewed in detail for potential edge cases or errors in business logic.
- **Performance:** The use of complex states and frequent updates might impact performance, especially on slower devices. Consider optimizing state management or splitting components.

Recommendations

- **Comments:** Add detailed comments explaining the purpose and functionality of complex code segments.
- **Refactoring:** Consider using a form management library to reduce boilerplate and improve handling of form state and validation.
- **Long Lines:** Break long lines into multiple lines for improved readability.
- **Validation:** Implement robust input validation before processing form submissions to prevent processing invalid or malicious data.

- **Performance Optimization:** Review and optimize state management to prevent unnecessary re-renders and improve performance.

Reviewed Code

```

55         School: {
56             ...prev.School,
57             schools: schoolData
58         }
59     }));
60     .catch(error => console.error('Error fetching data:', error));
61 }, []]);
62
63
64 const handleDiscard = () => {
65     navigate('/dashboard');
66 };
67
68 const formatDateTime = (date, time) => {
69     return `${date}T${time}:00`;
70 };
71
72
73 const handleSubmit = async (event) => {
74     event.preventDefault();
75     const test = "1111"
76
77     const startDate = formatDateTime(data.Delivery.programDate, data.Delivery.startTime);
78     const endDate = formatDateTime(data.Delivery.programDate, data.Delivery.endTime);
79
80     const schoolData = {
81         name: data.School.schools.find(school => school.id === data.School.schoolSelect)?.name,
82         studentYear: data.School.studentYears[0],
83         numStudentRegistered: parseInt(data.School.registeredStudents),
84         lowSES: data.School.lowSES === 'Y',
85         // allergy: data.School.allergy === 'Y',
86         contactFirstName: data.School.contactInfo.firstName,
87         contactLastName: data.School.contactInfo.lastName,
88         email: data.School.contactInfo.email,
89         phone: data.School.contactInfo.phoneNumber,
90         note: data.School.additionalComments,
91         isAccessibility: data.School.accessibilityNeeds === 'Y',
92         isAllergy: data.School.allergenInfo === 'Y',
93         isPartner: data.School.isPartnerSchool === 'Y',
94         allergy:test
95     };
96
97     const schoolId = await createNewSchool(schoolData);
98
99     const checklistId = await createNewChecklist(data.Delivery.templateSelect)
100    const schoolIdValue = schoolId._id;
101
102    const checklistvalue = checklistId._id
103    console.log(checklistId)
104    console.log(checklistvalue)
105
106
107
108
109    const bus_test_status = 1
110    const busData = {
111        busReq: data.Bus.busReq === 'Y', //
112        isBooked: data.Bus.busBooked === 'Y', //

```

```

113     status: bus_test_status, //
114     price: parseFloat(data.Bus.price), //
115     date_paid: data.Bus.datePaid, //
116     invoice: data.Bus.invoiceNumber, //
117     // savedReceipt: data.Bus.savedReceipt === 'Y',
118     // expenseMaster: data.Bus.expenseMaster === 'Y',
119     // pinEmail: data.Bus.pinEmail === 'Y'
120   };
121   const bookingData = {
122     event:event_test, //
123     status:status, //
124     name: data.School.schools.find(school => school.id === data.School.schoolSelect)?.name, //
125     // school_id: data.School.schoolSelect, //
126     school_id: schoolIdValue,
127     programStream: data.Delivery.streamSelect, //
128     checklist_id: checklistvalue, //
129     facilitators: data.Delivery.facilitatorsSelect, //
130     location: data.Delivery.locationSelect, //
131     date: data.Delivery.programDate, //
132     term: parseInt(data.Delivery.term, 10),      // +
133     // startTime: data.Delivery.startTime,
134     // endTime: data.Delivery.endTime,
135     startTime : startDate,
136     endTime : endDate,
137     module_id: module_test, //
138     exhibition: data.Delivery.exhibitionSelect, //
139     note: data.Delivery.notes, //
140     bus: busData,//
141     per_student: parseInt(data.Others.perStudent), //
142     expense: parseFloat(data.Others.expenses), //
143     income: parseFloat(data.Others.income), //
144     profit: parseFloat(data.Others.profit) //
145   };
146   console.log(bookingData)
147
148   try {
149     const response = await createNewBooking(bookingData);
150     console.log('Booking created successfully!', response);
151     navigate('/dashboard');
152   } catch (error) {
153     console.error('Creating booking failed:', error);
154   }
155 };
156
157
158
159   const handleChange = (category, field, value) => {
160     setData(prev => ({
161       ...prev,
162       [category]: {
163         ...prev[category],
164         [field]: value
165       }
166     }));
167   };
168
169
170   const handleMultiSelectChange = (category, field, index, value) => {

```

```

171     setData(prev => ({
172       ...prev,
173       [category]: {
174         ...prev[category],
175         [field]: [
176           ...prev[category][field].slice(0, index),
177           value,
178           ...prev[category][field].slice(index + 1)
179         ]
180       }
181     }));
182   };
183
184   const handleChangeNested = (category, nestedField, subField, value) => {
185     setData(prev => ({
186       ...prev,
187       [category]: {
188         ...prev[category],
189         [nestedField]: {
190           ...prev[category][nestedField],
191           [subField]: value
192         }
193       }
194     }));
195   };
196
197
198
199
200   return (
201     <>
202       <Header>Create New Booking</Header>
203       <div className="newBookingFilterSection">
204         <button
205           className={`newBookingFilterBtn ${{
206             activeCategory === "Delivery" ? "active" : ""
207           }}`}
208           onClick={() => handleCategoryClick("Delivery")}
209         >
210           Delivery
211         </button>
212         <button
213           className={`newBookingFilterBtn ${{
214             activeCategory === "School" ? "active" : ""
215           }}`}
216           onClick={() => handleCategoryClick("School")}
217         >
218           School
219         </button>
220         <button
221           className={`newBookingFilterBtn ${{
222             activeCategory === "Bus" ? "active" : ""
223           }}`}
224           onClick={() => handleCategoryClick("Bus")}
225         >
226           Bus
227         </button>
228         <button

```

```

229         className={`newBookingFilterBtn ${
230             activeCategory === "Others" ? "active" : ""
231         }`}
232         onClick={() => handleCategoryClick("Others")}
233     >
234     Others
235   </button>
236   <span className="autoFillText" onClick={openPopup}>
237     Do you want to autofill the information? Click here!
238   </span>
239 </div>
240
241 {isPopupOpen && (
242   <div className="popup">
243     <div className="popupContent">
244       <textarea
245         placeholder="Enter autofill data"
246         value={autoFillData}
247         onChange={(e) => setAutoFillData(e.target.value)}
248       ></textarea>
249       <div className="popupButtons">
250         <button onClick={closePopup}>Back</button>
251         <button onClick={handleGenerate}>Generate</button>
252       </div>
253     </div>
254   </div>
255 )} 
256 <div>
257 {activeCategory === "Delivery" && (
258   <form className="newBookingForm">
259     <label>Program Stream:</label>
260     <select
261       value={data.Delivery.streamSelect}
262       onChange={(e) => handleChange('Delivery', 'streamSelect', e.target.value)}
263     >
264       <option value="">Please select a stream</option>
265       {data.Delivery.programStreams && data.Delivery.programStreams.map((stream, index) => (
266         <option key={index} value={stream}>{stream}</option>
267       )))
268     </select>
269
270     <label>Template:</label>
271     <select
272       value={data.Delivery.templateSelect}
273       onChange={(e) => handleChange('Delivery', 'templateSelect', e.target.value)}
274     >
275       <option value="" disabled>Please select a template</option>
276       {data.Delivery.templates && data.Delivery.templates.map((template, index) => (
277         <option key={index} value={template.id}>{template.name}</option>
278       )))
279     </select>
280
281
282     <label>Facilitators:</label>
283     <select
284       value={data.Delivery.facilitatorsSelect}
285       onChange={(e) => handleChange('Delivery', 'facilitatorsSelect', e.target.value)}
286   
```

```

287      >
288          <option value="">Please select facilitators</option>
289          {data.Delivery.facilitators && data.Delivery.facilitators.map((facilitator, index) => (
290              <option key={index} value={facilitator}>{facilitator}</option>
291          )))
292      </select>
293
294      <label>Delivery Location:</label>
295      <select
296          value={data.Delivery.locationSelect}
297          onChange={(e) => handleChange('Delivery', 'locationSelect', e.target.value)}
298      >
299          <option value="">Please select a location</option>
300          {data.Delivery.location && data.Delivery.location.map((location, index) => (
301              <option key={index} value={location}>{location}</option>
302          )))
303      </select>
304
305      <label>Program Date:</label>
306      <input
307          type="date"
308          value={data.Delivery.programDate}
309          onChange={(e) => handleChange('Delivery', 'programDate', e.target.value)}
310      />
311
312      <label>Term:</label>
313      <select
314          value={data.Delivery.term}
315          onChange={(e) => handleChange('Delivery', 'term', e.target.value)}
316      >
317          <option value="1">1</option>
318          <option value="2">2</option>
319      </select>
320
321      <label>Time (Start):</label>
322      <input
323          type="time"
324          value={data.Delivery.startTime}
325          onChange={(e) => handleChange('Delivery', 'startTime', e.target.value)}
326      />
327
328      <label>Time (End):</label>
329      <input
330          type="time"
331          value={data.Delivery.endTime}
332          onChange={(e) => handleChange('Delivery', 'endTime', e.target.value)}
333      />
334
335      {[ "Module 1", "Module 2", "Module 3" ].map((label, index) => (
336          <React.Fragment key={label}>
337              <label>{label}:</label>
338              <select
339                  value={data.Delivery.moduleSelects[index]}
340                  onChange={(e) => handleMultiSelectChange('Delivery', 'moduleSelects', index, e.target.value)}
341              >
342                  <option value="">Please select a module</option>
343                  {data.Delivery.module && data.Delivery.module.map((module, modIndex) => (
344                      <option key={modIndex} value={module}>{module}</option>

```

```

345         ))}
346     </select>
347   </React.Fragment>
348 })
349
350   <label>Exhibition:</label>
351   <select
352     value={data.Delivery.exhibitionSelect}
353     onChange={(e) => handleChange('Delivery', 'exhibitionSelect', e.target.value)}
354   >
355     <option value="">Please select an exhibition</option>
356     {data.Delivery.exhibition && data.Delivery.exhibition.map((exhibition, index) =>
357       <option key={index} value={exhibition}>{exhibition}</option>
358     ))}
359   </select>
360
361   <label>Amendments/Notes:</label>
362   <textarea
363     value={data.Delivery.notes}
364     onChange={(e) => handleChange('Delivery', 'notes', e.target.value)}
365   />
366 </form>
367 )
368
369 {activeCategory === "School" && (
370   <form className="newBookingForm">
371     <label htmlFor="school-input">School Name:</label>
372     <select
373       value={data.School.schoolSelect}
374       onChange={(e) => {
375         const selectedSchoolId = e.target.value;
376         const selectedSchool = data.School.schools.find(school => school.id === selectedSchoolId);
377         handleChange('School', 'schoolSelect', selectedSchoolId);
378         handleChangeNested('School', 'school', 'id', selectedSchoolId);
379         handleChangeNested('School', 'school', 'name', selectedSchool?.name);
380       }}
381     >
382       <option value="" disabled>Please select a school</option>
383       {data.School.schools.map((school, index) => (
384         <option key={index} value={school.id}>{school.name}</option>
385       ))}
386     </select>
387
388
389   <div className="form-group">
390     <span id="newBookingStudentYear">Student Years:</span>
391     {years.map((year) => (
392       <div key={year} className="checkbox-container">
393         <input
394           type="checkbox"
395           id={year}
396           checked={data.School.studentYears.includes(year)}
397           onChange={() => handleToggleYear(year)}
398         />
399         <label htmlFor={year} className="checkbox-label">{year}</label>
400       </div>
401     )));
402

```

```

403     </div>
404
405     <label>Student # (registered):</label>
406     <input
407         type="number"
408         value={data.School.registeredStudents}
409         onChange={(e) => handleChange('School', 'registeredStudents', e.target.value)}
410     />
411
412     <label>Low SES:</label>
413     <select
414         value={data.School.lowSES}
415         onChange={(e) => handleChange('School', 'lowSES', e.target.value)}
416     >
417         <option value="" disabled>Yes or No</option>
418         <option value="Y">Yes</option>
419         <option value="N">No</option>
420     </select>
421
422     <fieldset>
423         <legend>School Contact:</legend>
424         <label>First Name:</label>
425         <input
426             type="text"
427             value={data.School.contactInfo.firstName}
428             onChange={(e) => handleChangeNested('School', 'contactInfo', 'firstName', e.target.value)}
429         />
430
431         <label>Last Name:</label>
432         <input
433             type="text"
434             value={data.School.contactInfo.lastName}
435             onChange={(e) => handleChangeNested('School', 'contactInfo', 'lastName', e.target.value)}
436         />
437         <label>Email Address:</label>
438         <input
439             type="email"
440             value={data.School.contactInfo.email}
441             onChange={(e) => handleChangeNested('School', 'contactInfo', 'email', e.target.value)}
442         />
443         <label>Phone Number:</label>
444         <input
445             type="tel"
446             value={data.School.contactInfo.phoneNumber}
447             onChange={(e) => handleChangeNested('School', 'contactInfo', 'phoneNumber', e.target.value)}
448         />
449         <label>Teaching Area/Job Title:</label>
450         <input
451             type="text"
452             value={data.School.contactInfo.jobTitle}
453             onChange={(e) => handleChangeNested('School', 'contactInfo', 'jobTitle', e.target.value)}
454         />
455     </fieldset>
456
457     <label>Additional Comments:</label>
458     <textarea
459         value={data.School.additionalComments}
460         onChange={(e) => handleChange('School', 'additionalComments', e.target.value)}>
```

```

461      />
462
463      <label>Accessibility Needs Communicated:</label>
464      <select
465          value={data.School.accessibilityNeeds}
466          onChange={(e) => handleChange('School', 'accessibilityNeeds', e.target.value)}
467      >
468          <option value="" disabled>Yes or No</option>
469          <option value="Y">Yes</option>
470          <option value="N">No</option>
471      </select>
472
473      <label>Allergen and Anaphylaxis Communicated:</label>
474      <select
475          value={data.School.allergenInfo}
476          onChange={(e) => handleChange('School', 'allergenInfo', e.target.value)}
477      >
478          <option value="" disabled>Yes or No</option>
479          <option value="Y">Yes</option>
480          <option value="N">No</option>
481      </select>
482
483      <label>Is Partner School:</label>
484      <select
485          value={data.School.isPartnerSchool}
486          onChange={(e) => handleChange('School', 'isPartnerSchool', e.target.value)}
487      >
488          <option value="" disabled>Yes or No</option>
489          <option value="Y">Yes</option>
490          <option value="N">No</option>
491      </select>
492  </form>
493 }
494
495
496 {activeCategory === "Bus" && (
497     <form className="newBookingForm">
498         <label htmlFor="bus-req">BUS REQ:</label>
499         <select
500             id="bus-req"
501             value={data.Bus.busReq}
502             onChange={(e) => handleChange('Bus', 'busReq', e.target.value)}
503         >
504             <option value="">Yes or No</option>
505             <option value="Y">Yes</option>
506             <option value="N">No</option>
507         </select>
508
509         <label>BUS BOOKED:</label>
510         <select
511             id="bus-booked"
512             value={data.Bus.busBooked}
513             onChange={(e) => handleChange('Bus', 'busBooked', e.target.value)}
514         >
515             <option value="">Yes or No</option>
516             <option value="Y">Yes</option>
517             <option value="N">No</option>
518         </select>

```

```

519      <label htmlFor="status">Status:</label>
520      <select
521          id="status"
522          value={data.Bus.status}
523          onChange={(e) => handleChange('Bus', 'status', e.target.value)}
524      >
525          <option value="">Select Status</option>
526          <option value="paid">Paid</option>
527          <option value="processing">Processing</option>
528      </select>
529
530
531      <label htmlFor="price">Price:</label>
532      <input
533          id="price"
534          type="text"
535          value={data.Bus.price}
536          onChange={(e) => handleChange('Bus', 'price', e.target.value)}
537      />
538
539      <label htmlFor="date-paid">Date Paid:</label>
540      <input
541          id="date-paid"
542          type="date"
543          value={data.Bus.datePaid}
544          onChange={(e) => handleChange('Bus', 'datePaid', e.target.value)}
545      />
546
547      <label htmlFor="invoice-number">Invoice #:</label>
548      <input
549          id="invoice-number"
550          type="text"
551          value={data.Bus.invoiceNumber}
552          onChange={(e) => handleChange('Bus', 'invoiceNumber', e.target.value)}
553      />
554
555      <div className="checkbox-container">
556          <input
557              type="checkbox"
558              id="saved-receipt"
559              checked={data.Bus.savedReceipt === "Y"}
560              onChange={(e) => handleChange('Bus', 'savedReceipt', e.target.checked ? "Y" : "N")}
561          />
562          <label htmlFor="saved-receipt">Saved Receipt:</label>
563      </div>
564
565      <div className="checkbox-container">
566          <input
567              type="checkbox"
568              id="enter-into-expense-master"
569              checked={data.Bus.enterIntoExpenseMaster === "Y"}
570              onChange={(e) => handleChange('Bus', 'enterIntoExpenseMaster', e.target.checked ? "Y" : "N")}
571          />
572          <label htmlFor="enter-into-expense-master">Enter into Expense Master:</label>
573      </div>
574
575      <div className="checkbox-container">
576          <input

```

```

577         type="checkbox"
578         id="pin-categorise-email"
579         checked={data.Bus.pinCategoriseEmail === "Y"}
580         onChange={(e) => handleChange('Bus', 'pinCategoriseEmail', e.target.checked ? "Y" : "N")}
581     />
582     <label htmlFor="pin-categorise-email">Pin/Categorise Email:</label>
583   </div>
584 </form>
585 }
586
587
588 {activeCategory === "Others" && (
589   <form className="newBookingForm">
590     <label htmlFor="per-student">Per Student:</label>
591     <input
592       id="per-student"
593       type="number"
594       value={data.Others.perStudent}
595       onChange={(e) => handleChange('Others', 'perStudent', e.target.value)}
596     />
597
598     <label htmlFor="expenses">Cost:</label>
599     <input
600       id="expenses"
601       type="number"
602       value={data.Others.expenses}
603       onChange={(e) => handleChange('Others', 'expenses', e.target.value)}
604     />
605
606     <label htmlFor="income">Income:</label>
607     <input
608       id="income"
609       type="number"
610       value={data.Others.income}
611       onChange={(e) => handleChange('Others', 'income', e.target.value)}
612     />
613
614     <label htmlFor="profit">Profit:</label>
615     <input
616       id="profit"
617       type="number"
618       value={data.Others.profit}
619       onChange={(e) => handleChange('Others', 'profit', e.target.value)}
620     />
621   </form>
622 }
623
624 </div>
625 <div className="newBookingButtons">
626   <Button type="discard" onClick={handleDiscard}>
627     DISCARD
628   </Button>
629   <Button type="submit" onClick={handleSubmit}>
630     Create
631   </Button>
632 </div>
633 </>
634 );

```

```

635 ];
636
637 export default NewBooking;

```

Based on the feedback,

we did some improvement in order to produce more professional code

- Adding Comment
- Breaking the long line into multiple lines for better readability.
- Avoid duplicated code especially in form fields and select options creation
- The handling of form states and changes can be streamlined using form management libraries like Formik or React Hook Form, which can also handle validation more cleanly.
- Improve the security vulnerabilities that the form does not include validation before submitting data, which can lead to invalid data being processed.
- Review and optimize state management to prevent unnecessary re-renders and improve performance.

Consider

This feedback is very useful even it will tell out there are some library that we can use. We basically adopted all the suggestions given by gpt, because the suggestions he gave are very standard and help to make our code more standardized and eliminate potential errors. We are still students, and the code we write Sometimes it's hard to meet the standard all the time. With the feedback we can make sure that we are on the right track or not.

Modified Code

```

1 import React, { useEffect, useState } from 'react';
2 import { navigate } from 'react-router-dom'; // Assuming this is from 'react-router-dom'
3 import { useForm } from 'react-hook-form'; // Example of a form management library
4
5 /**
6  * NewBooking component handles creating new bookings with detailed forms for delivery, school, bus, and other
7  * Utilizes useForm from React Hook Form for state management and validation, reducing boilerplate and improvin
8 */
9 const NewBooking = () => {
10   const { register, handleSubmit, setValue, watch, formState: { errors } } = useForm();
11   const [data, setData] = useState({}); 
12   const [isPopupOpen, setIsPopupOpen] = useState(false);
13   const [activeCategory, setActiveCategory] = useState('');
14
15   useEffect(() => {
16     fetchInitialData();
17   }, []);
18
19 // Fetch initial data and set up form defaults
20 const fetchInitialData = async () => {
21   try {
22     const [miscellaneousData, schoolData, templateData] = await Promise.all([
23       getAllMiscellaneous(),
24       getAllSchool(),
25       getAllTemplates()
26     ]);
27     setData({
28       Delivery: {

```

```

29     programStreams: miscellaneousData.program_stream,
30     facilitators: miscellaneousData.facilitators,
31     location: miscellaneousData.delivery_location,
32     module: miscellaneousData.module,
33     exhibition: miscellaneousData.exhibition,
34     templates: templateData
35   },
36   School: {
37     schools: schoolData
38   }
39 });
40 } catch (error) {
41   console.error('Error fetching data:', error);
42 }
43 };
44
45 // Handle form submission
46 const onSubmit = async (formData) => {
47   try {
48     const bookingData = prepareBookingData(formData);
49     const response = await createNewBooking(bookingData);
50     console.log('Booking created successfully!', response);
51     navigate('/dashboard');
52   } catch (error) {
53     console.error('Creating booking failed:', error);
54   }
55 };
56
57 // Prepare booking data from form data
58 const prepareBookingData = (formData) => {
59   const { startDate, endDate, schoolSelect, ...rest } = formData;
60   const schoolDetails = data.School.schools.find(school => school.id === schoolSelect);
61
62   return {
63     ...rest,
64     startDate: formatDateTime(startDate, formData.startTime),
65     endDate: formatDateTime(endDate, formData.endTime),
66     schoolName: schoolDetails.name,
67     schoolId: schoolDetails.id,
68   };
69 };
70
71 // Utility to format date and time into a single string
72 const formatDateTime = (date, time) => `${date}T${time}:00`;
73
74 // Render form fields dynamically based on active category
75 const renderFormFields = () => {
76   switch (activeCategory) {
77     case "Delivery":
78       return renderDeliveryFields();
79     case "School":
80       return renderSchoolFields();
81     case "Bus":
82       return renderBusFields();
83     case "Others":
84       return renderOtherFields();
85     default:
86       return null;

```

```

87     }
88   };
89
90   // Example field rendering function for Delivery
91   const renderDeliveryFields = () => (
92     <>
93       <select {...register('streamSelect', { required: true })}>
94         <option value="">Please select a stream</option>
95         {data.Delivery.programStreams.map((stream, index) => (
96           <option key={index} value={stream}>{stream}</option>
97         )));
98       </select>
99       {errors.streamSelect && <p>This field is required.</p>}
100      {/* More fields can be added here */}
101    </>
102  );
103
104  return (
105    <div>
106      <h1>Create New Booking</h1>
107      <form onSubmit={handleSubmit(onSubmit)}>
108        {renderFormFields()}
109        <button type="submit">Create</button>
110        <button type="button" onClick={() => navigate('/dashboard')}>Discard</button>
111      </form>
112    </div>
113  );
114};
115
116 export default NewBooking;

```

Merge Request Feedback 2

Date: 02/05/2024

Participants: @Guixian Li @Pangfeng ZHENG @Yuanbo Xu

Code Review Merge Pull Request #29

Feedback from AI (ChatGPT 4)

1. Documentation Defects

- **Quality of Names:** The naming of components and variables is generally clear and descriptive (`Header`, `GlobalFilter`, `ColumnFilter`, `filteredColumns`). However, some names like `Database` for the component might be too generic given it represents a specific view or functionality related to a table with pagination and sorting.
- **Comments:** There are no comments in the code. Adding comments explaining the purpose of main functions, especially the complex logic for filtering columns based on the selected group, would be helpful for maintenance and understanding.

2. Visual Representation Defects

- **Bracket Usage:** All JSX and JavaScript blocks appear to be correctly wrapped in brackets.
- **Indentation:** The code mostly follows a consistent indentation pattern which is good for readability. However, the closing bracket of the `Popover` component and its contents could benefit from better indentation to enhance readability.
- **Long Code Statements:** Several lines, especially within JSX, are quite long and would benefit from breaking into multiple lines (e.g., lines involving `<FontAwesomeIcon ... />`).

3. Structure Defects

- **Dead Code:** There are no apparent instances of dead code in the snippet.
- **Duplication:** There seems to be minimal code duplication. The structure is fairly DRY (Don't Repeat Yourself), which is commendable.

4. New Functionality

- **Use Standard Method:** The code uses standardized methods and React hooks effectively (`useMemo`, `useState`). The use of `react-table` hooks for sorting, filtering, and pagination is appropriate and follows best practices for such features.

5. Resource Defects

- **Variable Initialization:** All variables are correctly initialized. However, note that data entries with missing fields (e.g., `{ id: 5 } to { id: 11 }`) might lead to rendering issues or errors if the code attempts to access missing properties.
- **Memory Management:** React and `react-table` handle most memory management internally. Usage of `useMemo` for data and columns is appropriate for avoiding unnecessary recalculations.

6. Check Defects

- **Check User Input:** Basic input handling is present via `select` for column group selection. However, there's no explicit validation of user input which could be a concern if more complex inputs were expected.

7. Interface Defects

- **Parameter:** Parameters used in function calls and component props appear correct and appropriately passed.

8. Logic Defects

- **Compute:** The logic for filtering based on the selected group is appropriate and should function as intended based on the current setup.
- **Performance:** The use of memoization and controlled component updates should ensure good performance. However, as data grows larger, consider evaluating performance again, especially regarding the rendering of potentially large datasets.

Recommendations

- Improvements could include adding comments
- break long lines for better readability
- add user input validations if the scope expands to handle more complex data or interactions.

Reviewed Code

```

1 import React, { useMemo, useState } from 'react';
2 import Header from '../components/Header/Header';
3 import './styles/Database.css';
4 import { useTable, useSortBy, useFilters, useGlobalFilter, usePagination } from 'react-table';
5 import { Grouped_Columns } from '../components/Table/Column';
6 import { Popover } from "@material-ui/core";
7 import ColumnFilter from '../components/Table/ColumnFilter';
8 import GlobalFilter from '../components/Table/GlobalFilter';
9 import { TypographyParagraph, } from "../components/Typography/Typography";
10 import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
11 import { faSort, faSortUp, faSortDown, faTableColumns } from "@fortawesome/free-solid-svg-icons";
12
13 function Database() {
14   const data = useMemo(() => ([
15     { id: 1, ProgramStream: 'SCoE: Excursions', Status: 'Delivered', School: 'Aireys Inlet Primary School' , Pr
16     { id: 2, ProgramStream: 'SCoE: Excursions', Status: 'Delivered', School: 'Antonine College' , ProgramDate:
17     { id: 3, ProgramStream: 'SCoE: Excursions', Status: 'Delivered', School: 'Beaufort Primary School' , Progra
18     { id: 4, ProgramStream: 'STEAM: Excursions', Status: 'Delivered', School: 'Dorset Primary School' , Program
19     { id: 5 },
20     { id: 6 },
21     { id: 7 },
22     { id: 8 },
23     { id: 9 },
24     { id: 10 },
25     { id: 11 },
26   ]), []);
27
28   const columns = useMemo(() => Grouped_Columns, [])
29
30   const [selectGroup, setSelectGroup] = useState('All');
31
32   const groupOptions = useMemo(() => {
33     const options = columns.filter(column => column.columns).map(column => ({
34       label: column.Header,
35       value: column.Header
36     }));
37     options.unshift({label: 'All', value: 'All'});
38     return options;
39   }, [columns]);
40
41   const handleSelectGroup = (event) => {
42     setSelectGroup(event.target.value);
43   };
44
45   const filteredColumns = useMemo(() => {
46     const baseColumns = [{{
47       Header: 'ID',
48       accessor: 'id',
49       canSort: false,
50     }};
51     if (selectGroup === 'All') {

```

```

52     return columns;
53   }
54   else {
55     const cols = columns.find(column => column.Header === selectGroup)?.columns;
56     return baseColumns.concat({Header: selectGroup, columns: cols});
57   }
58 }, [selectGroup, columns]);
59
60 const [anchor, setAnchor] = useState(null);
61
62 const openPopover = (event) => {
63   setAnchor(event.currentTarget);
64 }
65
66 const defaultColumn = useMemo(() => {
67   return {
68     Filter: ColumnFilter
69   }
70 }, [])
71
72 const {
73   getTableProps,
74   getTableBodyProps,
75   headerGroups,
76   page,
77   nextPage,
78   previousPage,
79   canNextPage,
80   canPreviousPage,
81   pageOptions,
82   prepareRow,
83   state,
84   setGlobalFilter,
85   allColumns,
86 } = useTable({
87   columns: filteredColumns,
88   data,
89   defaultColumn,
90 }, useFilters, useGlobalFilter, useSortBy, usePagination)
91
92 const { globalFilter, pageIndex } = state
93
94 const sortIcon = <FontAwesomeIcon icon={faSort} style={{ fontSize: "15px" }} />
95 const sortUp = <FontAwesomeIcon icon={faSortUp} style={{ fontSize: "15px" }} />
96 const sortDown = <FontAwesomeIcon icon={faSortDown} style={{ fontSize: "15px" }} />
97
98 return (
99 <div>
100   <Header> Database </Header>
101   <div className='bg'>
102     <GlobalFilter filter={globalFilter} setFilter={setGlobalFilter} /> {' '}
103     <select onChange={handleSelectGroup} value={selectGroup}>
104       {groupOptions.map(option => (
105         <option key={option.value} value={option.value}>{option.label}</option>
106       )))
107     </select> {' '}
108     <button onClick={openPopover} className='Button' >
109       <FontAwesomeIcon icon={faTableColumns} style={{ fontSize: "15px" }} />

```

```

110     </button>
111     <Popover
112       open={Boolean(anchor)}
113       anchorEl={anchor}
114       onClose={() => setAnchor(null)}
115       anchorOrigin={{
116         vertical: 'bottom',
117         horizontal: 'left'
118     }}>
119       <div>
120         {allColumns.map(column => (
121           <div key = {column.id}>
122             <label>
123               <input type='checkbox' {...column.getToggleHiddenProps()} />
124               {column.Header}
125             </label>
126           </div>
127         )))
128       </div>
129     </Popover>
130   <div className='scrollx'>
131     <table {...getTableProps()} className='table'>
132       <thead>
133         {headerGroups.map(headerGroup => (
134           <tr {...headerGroup.getHeaderGroupProps()}>
135             { headerGroup.headers.map((column) => (
136               <th {...column.getHeaderProps()} className='th'>
137                 <div>
138                   {column.render('Header')}
139                   {' '}
140                   {column.canSort && (
141                     <button {...column.getSortByToggleProps()} className='Button'>
142                       <span>
143                         {column.isSorted ? (column.isSortedDesc ? sortUp : sortDown) : sortIcon}
144                       </span>
145                     </button>
146                   )}
147                 </div>
148               <div>
149                 {column.canFilter ? column.render('Filter') : null}
150               </div>
151             </th>
152           )))
153         </tr>
154       )))
155     </thead>
156     <tbody {...getTableBodyProps()}>
157       {page.map((row) => {
158         prepareRow(row)
159         return(
160           <tr {...row.getRowProps()}>
161             {row.cells.map((cell) => {
162               return <td {...cell.getCellProps()} className='td'>
163                 {cell.render('Cell')}
164               </td>
165             )))
166           </tr>
167         )
168       })
169     </tbody>
170   </table>
171 </div>

```

```

168         })
169     </tbody>
170   </table>
171   <div>
172     <button onClick={() => previousPage()} disabled={!canPreviousPage}>Previous</button>{' '}
173     <span>
174       Page{' '}
175       <strong>
176         {pageIndex + 1} of {pageOptions.length}
177       </strong>{' '}
178     </span>
179     <button onClick={() => nextPage()} disabled={!canNextPage}>Next</button>
180   </div>
181 </div>
182 </div>
183 </div>
184 );
185 }
186 export default Database;
187

```

Based on the feedback,

we made some improvements in order to produce more professional code

- Refactor the JSX to enhance readability by breaking down long lines and improving indentation.
- Handling potential null or undefined values
- Add comments for clarity and documentation

Consider

The feedback provided is comprehensive and effectively targets key areas where the code could be improved in terms of readability, maintainability, and robustness. We basically adopted all the suggestions given by GPT, because the suggestions he gave are very standard and help to make our code more standardized and eliminate potential errors. We are still students, and the code we write. Sometimes it's hard to meet the standard all the time. With the feedback we can make sure that we are on the right track or not.

Modified Code

```

1 import React, { useMemo, useState } from 'react';
2 import { useTable, useSortBy, useFilters, useGlobalFilter, usePagination } from 'react-table';
3 import { Popover } from "@material-ui/core";
4 import Header from "../components/Header/Header";
5 import ColumnFilter from "../components/Table/ColumnFilter";
6 import GlobalFilter from "../components/Table/GlobalFilter";
7 import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
8 import { faSort, faSortUp, faSortDown, faTableColumns } from "@fortawesome/free-solid-svg-icons";
9
10 /**
11  * Database component provides a UI for viewing and managing a dataset with functionalities such as sorting, fi
12  * Utilizes react-table library for handling data operations efficiently.
13 */
14 function Database() {
15   // useMemo is used here to memoize data and columns for performance optimization
16   const data = useMemo(() => ([
17     { id: 1, ProgramStream: 'SCoE: Excursions', Status: 'Delivered', School: 'Aireys Inlet Primary School', Pro

```

```

18 { id: 2, ProgramStream: 'SCoE Excursions', Status: 'Delivered', School: 'Antonine College', ProgramDate: ''
19 // More data objects...
20 ]), []);
21
22 const columns = useMemo(() => Grouped_Columns, []);
23
24 // State for managing selected group for filtering columns
25 const [selectGroup, setSelectGroup] = useState('All');
26 const groupOptions = useMemo(() => {
27   const options = columns.filter(column => column.columns).map(column => ({
28     label: column.Header,
29     value: column.Header
30   }));
31   options.unshift({ label: 'All', value: 'All' });
32   return options;
33 }, [columns]);
34
35 // Handling selection change for group filter
36 const handleSelectGroup = (event) => {
37   setSelectGroup(event.target.value);
38 };
39
40 // Filter columns based on the selected group
41 const filteredColumns = useMemo(() => {
42   if (selectGroup === 'All') {
43     return columns;
44   } else {
45     const selectedGroupColumns = columns.find(column => column.Header === selectGroup)?.columns || [];
46     return [{ Header: 'ID', accessor: 'id', disableSortBy: true }, ...selectedGroupColumns];
47   }
48 }, [selectGroup, columns]);
49
50 // State for managing Popover component visibility
51 const [anchor, setAnchor] = useState(null);
52 const openPopover = (event) => {
53   setAnchor(event.currentTarget);
54 };
55
56 // Default column properties setup
57 const defaultColumn = useMemo(() => ({
58   Filter: ColumnFilter
59 }), []);
60
61 // Using useTable hook from react-table to initialize table instance
62 const {
63   getTableProps,
64   getTableBodyProps,
65   headerGroups,
66   page,
67   nextPage,
68   previousPage,
69   canNextPage,
70   canPreviousPage,
71   pageOptions,
72   prepareRow,
73   state,
74   setGlobalFilter,
75 } = useTable({

```

```

76     columns: filteredColumns,
77     data,
78     initialState: { pageIndex: 0 },
79     defaultColumn,
80   }, useFilters, useGlobalFilter, useSortBy, usePagination);
81
82   const { globalFilter, pageIndex } = state;
83
84   return (
85     <div>
86       <Header title="Database Management"/>
87       <GlobalFilter filter={globalFilter} setFilter={setGlobalFilter}/>
88       <select onChange={handleSelectGroup} value={selectGroup}>
89         {groupOptions.map(option => (
90           <option key={option.value} value={option.value}>{option.label}</option>
91         ))}
92       </select>
93       <button onClick={openPopover} className='button'>
94         <FontAwesomeIcon icon={faTableColumns} />
95       </button>
96     <Popover
97       open={Boolean(anchor)}
98       anchorEl={anchor}
99       onClose={() => setAnchor(null)}
100      anchorOrigin={{ vertical: 'bottom', horizontal: 'left' }}
101    >
102      <div>
103        {filteredColumns.map(column => (
104          <div key={column.id}>
105            <label>
106              <input type='checkbox' {...column.getToggleHiddenProps()} />
107              {column.Header}
108            </label>
109            </div>
110          )));
111        </div>
112      </Popover>
113      <div className='table-container'>
114        <table {...getTableProps()}>
115          <thead>
116            {headerGroups.map(headerGroup => (
117              <tr {...headerGroup.getHeaderGroupProps()}>
118                {headerGroup.headers.map(column => (
119                  <th {...column.getHeaderProps()}>
120                    {column.render('Header')}
121                    {column.canSort && (
122                      <button {...column.getSortByToggleProps()}>
123                        {column.isSorted
124                          ? column.isSortedDesc
125                            ? sortDown
126                            : sortUp
127                          : sortIcon}
128                        </button>
129                      )}
130                      {column.canFilter ? column.render('Filter') : null}
131                    </th>
132                  )));
133                </tr>

```

```
134         ))}
135     </thead>
136     <tbody {...getTableBodyProps()}>
137       {page.map(row => {
138         prepareRow(row);
139         return (
140           <tr {...row.getRowProps()}>
141             {row.cells.map(cell => (
142               <td {...cell.getCellProps()}>{cell.render('Cell')}
143             )));
144           </tr>
145         );
146       });
147     </tbody>
148   </table>
149   <div className='pagination'>
150     <button onClick={() => previousPage()} disabled={!canPreviousPage}>Previous</button>
151     <span>
152       Page {pageIndex + 1} of {pageOptions.length}
153     </span>
154     <button onClick={() => nextPage()} disabled={!canNextPage}>Next</button>
155   </div>
156 </div>
157 </div>
158 );
159 }
160
161 export default Database;
```

Merge Request Feedback 3

Date: 2/05/2024

Participants: @Guixian Li @Pangfeng ZHENG @Yuanbo Xu

Code Review Back-end #6 (Merge request #27)

Feedback from AI (ChatGPT 4)

1. Documentation Defects:

- **Naming:** The naming conventions used in the code are clear and descriptive, which is excellent for maintainability. Variable and function names like `BookingView`, `BookingViewID`, and `connect_mongodb` clearly indicate their functionality.
- **Comment:** The initial comments and docstrings at the beginning of the configuration files provide a helpful overview and links for further documentation, which is good. However, the `BookingView` class could benefit from additional comments explaining the purpose of each method more thoroughly.

2. Visual Representation Defects:

- **Bracket Usage:** Python doesn't use brackets for block delimitation; however, parentheses are correctly used where needed.
- **Indentation:** The code is well indented, adhering to Python's strict indentation requirements for defining blocks of code.
- **Long Line:** There are a few lines, particularly those involving MongoDB queries and URLs, that are quite long and could be wrapped to improve readability.

3. Structure Defects:

- **Dead Code:** There doesn't appear to be any dead code. Each part of the code has a clear purpose.
- **Duplication:** The code seems DRY (Don't Repeat Yourself) for the most part, although the pattern of fetching and embedding documents from MongoDB is repeated and could potentially be refactored into a utility function.

4. New Functionality:

- **Use Standard Method:** The code uses Django's and DRF's standard methods effectively. However, it might benefit from utilizing Django's model relations more effectively instead of manually handling foreign key relations through MongoDB queries.

5. Resource Defects:

- **Variable Initialization:** All variables are properly initialized.
- **Memory Management:** Python and Django manage memory efficiently; however, care should be taken with database connections to ensure they are closed or handled via context managers where appropriate.

6. Check Defects:

- **Check User Input:** The `APIView` methods use serializers for validating user input, which is appropriate. However, further validations could be added to ensure the integrity of incoming data, especially in complex updates or inserts.

7. Interface Defects:

- **Parameter:** The use of parameters in API calls, database queries, and function calls is correctly handled. Error handling for missing or incorrect parameters is implemented well.

8. Logic Defects:

- **Compute:** The logic within each API method is well-structured, although error handling could be enhanced by adding more specific checks or validations before performing database operations.
- **Performance:** While the direct database operations performed via MongoDB might be efficient, considering the use of Django's ORM for more complex queries or data manipulation could potentially improve performance and scalability.

Recommendations

- **Add more comments** in complex areas, particularly in the API views, to explain the logic and decision-making processes.
- **Refactor repeated logic** into helper functions or methods to reduce code complexity and improve maintainability.

- Consider using Django's ORM capabilities more extensively if possible, to leverage its features like relationship handling, which can simplify the code and potentially enhance performance.
- Review error handling strategies to cover potential edge cases or failure points in the API, enhancing the robustness of the system.

Reviewed Code

```

1  class BookingView(APIView):
2      def get(self, request, *args, **kwargs):
3          db = connect_mongodb()
4          booking_collection = db['booking']
5          checklist_collection = db['checklist']
6          school_collection = db['school']
7          #send_booking_ref_to_client("123", ['leonali0329@gmail.com'])
8
9          # Fetch all booking documents
10         booking_documents = list(booking_collection.find())
11
12         # Enrich each booking document with checklist and school details
13         enriched_bookings = []
14         for booking in booking_documents:
15             # If checklist_id exists, fetch the checklist document
16             if 'checklist_id' in booking and booking['checklist_id']:
17                 checklist_id = ObjectId(booking['checklist_id'])
18                 checklist_document = checklist_collection.find_one({'_id': checklist_id})
19                 if checklist_document:
20                     # Include the checklist details directly in the booking document
21                     booking['checklist'] = checklist_document
22
23             # If school_id exists, fetch the school document
24             if 'school_id' in booking and booking['school_id']:
25                 school_id = ObjectId(booking['school_id'])
26                 school_document = school_collection.find_one({'_id': school_id})
27                 if school_document:
28                     # Include the school details directly in the booking document
29                     booking['school'] = school_document
30
31         enriched_bookings.append(booking)
32
33         # Serialize the enriched booking documents
34         serializer = BookSerializer(enriched_bookings, many=True)
35         return Response(serializer.data, status=status.HTTP_200_OK)
36
37     def post(self, request, *args, **kwargs):
38         print("!!!!!!")
39         db = connect_mongodb()
40         booking_collection = db['booking']
41         checklist_collection = db['checklist']
42         school_collection = db['school']
43         serializer = BookSerializer(data=request.data)
44         print("aaaaaaaaa")
45         if serializer.is_valid():
46             # Insert data into MongoDB
47             print("11111111")
48             new_data = serializer.validated_data
49             if 'checklist_id' in request.data:
50                 print("22222222")
                 checklist_id = ObjectId(request.data['checklist_id'])

```

```

52     new_data['checklist_id'] = checklist_id
53     print("555555")
54     checklist_document = checklist_collection.find_one({'_id': checklist_id})
55     print("66666")
56     if not checklist_document:
57         print("77777")
58         return Response({"error": "checklist not found"}, status=status.HTTP_404_NOT_FOUND)
59     if 'school_id' in request.data:
60         print("3333333")
61         school_id = ObjectId(request.data['school_id'])
62         new_data['school_id'] = school_id
63         school_document = school_collection.find_one({'_id': school_id})
64         if not school_document:
65             return Response({"error": "school not found"}, status=status.HTTP_404_NOT_FOUND)
66
67         print("$$$$$")
68         print(new_data)
69         result = booking_collection.insert_one(new_data)
70         # print(result)
71         # Optionally add the MongoDB ID to the response
72         new_data['_id'] = str(result.inserted_id)
73         response_data = {**new_data, '_id': str(result.inserted_id)}
74         # Convert ObjectIds to strings for JSON serialization
75         if 'school_id' in response_data:
76             response_data['checklist_id'] = str(response_data['checklist_id'])
77         if 'school_id' in response_data:
78             response_data['school_id'] = str(response_data['school_id'])
79         if booking_collection.find_one({'_id': result.inserted_id}):
80             school_id = ObjectId(request.data['school_id'])
81             school_email = school_collection.find_one({'_id': school_id})['email']
82             # change to school email
83             send_booking_ref_to_client(str(result.inserted_id), ['leonali0329@gmail.com'])
84
85         # response_data = json.dumps(new_data, default=str)
86         return Response(response_data, status=status.HTTP_201_CREATED)
87         print("Validation Failed:", serializer.errors)
88         return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
89
90
91 class BookingViewID(APIView):
92     print("????")
93     def get(self, request, *args, **kwargs):
94         if 'id' not in kwargs:
95             return Response({'error': 'PUT method expects an id'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
96         db = connect_mongodb()
97         booking_collection = db['booking']
98         checklist_collection = db['checklist']
99         school_collection = db['school']
100
101        # Find the booking document by _id
102        booking_id = ObjectId(kwargs['id'])
103        booking_document = booking_collection.find_one({'_id': booking_id})
104
105        if booking_document:
106            # If checklist_id exists, fetch the checklist document
107            if 'checklist_id' in booking_document:
108                checklist_id = ObjectId(booking_document['checklist_id'])
109                checklist_document = checklist_collection.find_one({'_id': checklist_id})

```

```

110         booking_document['checklist'] = checklist_document # Embed the checklist document
111
112     # If school_id exists, fetch the school document
113     if 'school_id' in booking_document:
114         school_id = ObjectId(booking_document['school_id'])
115         school_document = school_collection.find_one({'_id': school_id})
116         booking_document['school'] = school_document # Embed the school document
117
118     # Serialize the enriched booking document
119     serializer = BookSerializer(booking_document)
120     return Response(serializer.data, status=status.HTTP_200_OK)
121
122     # If the booking document doesn't exist, return an error
123     return Response({'error': 'Booking not found'}, status=status.HTTP_404_NOT_FOUND)
124
125 def put(self, request, *args, **kwargs):
126     if 'id' not in kwargs:
127         return Response({'error': 'PUT method expects an id'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
128     db = connect_mongodb()
129     collection = db['booking']
130     serializer = BookSerializer(data=request.data)
131     if serializer.is_valid():
132         # Insert data into MongoDB
133         new_data = serializer.validated_data
134
135         update_result = collection.update_one({'_id': ObjectId(kwargs['id'])}, {'$set': new_data})
136         if update_result.matched_count == 0:
137             return Response({'error': 'No record found with the specified ID'}, status=status.HTTP_404_NOT_FOUND)
138         return Response({'status': 'success', 'id': kwargs['id'], 'updated': update_result.modified_count},
139             status=status.HTTP_200_OK)
140
141 def delete(self, request, *args, **kwargs):
142     if 'id' not in kwargs:
143         return Response({'error': 'DELETE method expects an id'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
144
145     db = connect_mongodb()
146     collection = db['booking']
147     delete_result = collection.delete_one({'_id': ObjectId(kwargs['id'])})
148
149     if delete_result.deleted_count == 0:
150         return Response({'error': 'No record found with the specified ID'}, status=status.HTTP_404_NOT_FOUND)
151     return Response({'status': 'success', 'id': kwargs['id']}, status=status.HTTP_200_OK)
152
153
154

```

Based on the feedback,

we did some improvement in order to produce more professional code

- Utilizing Django's model relations more effectively instead of manually handling foreign key relations through MongoDB queries.
- Further validations should be added to ensure the integrity of incoming data, especially in complex updates or inserts.
- Add more comments.
- Refactor repeated logic into helper functions or methods to reduce code complexity and improve maintainability.
- Review error handling

Consider

AI provided many suggestions to avoid potential errors in the back-end code review. We decided to follow his suggestions and improve our code.

Modified Code

```
1 from django.http import JsonResponse
2 from rest_framework.views import APIView
3 from rest_framework.response import Response
4 from rest_framework import status
5 from bson import ObjectId
6 from .models import Booking, Checklist, School
7 from .serializers import BookSerializer
8
9 def connect_mongodb():
10     # Placeholder for MongoDB connection logic
11     pass
12
13 def fetch_document(collection, object_id):
14     """Helper function to fetch a document using its ID from a specified collection."""
15     return collection.find_one({'_id': ObjectId(object_id)})
16
17 def enrich_booking_data(booking, checklist_collection, school_collection):
18     """Enrich booking data with related checklist and school details."""
19     if 'checklist_id' in booking and booking['checklist_id']:
20         booking['checklist'] = fetch_document(checklist_collection, booking['checklist_id'])
21
22     if 'school_id' in booking and booking['school_id']:
23         booking['school'] = fetch_document(school_collection, booking['school_id'])
24
25     return booking
26
27 class BookingView(APIView):
28     def get(self, request, *args, **kwargs):
29         db = connect_mongodb()
30         booking_collection = db['booking']
31         checklist_collection = db['checklist']
32         school_collection = db['school']
33
34         booking_documents = list(booking_collection.find())
35         enriched_bookings = [enrich_booking_data(booking, checklist_collection, school_collection) for booking
36
37             in booking_documents]
38         serializer = BookSerializer(enriched_bookings, many=True)
39         return Response(serializer.data, status=status.HTTP_200_OK)
40
41     def post(self, request, *args, **kwargs):
42         db = connect_mongodb()
43         booking_collection = db['booking']
44         checklist_collection = db['checklist']
45         school_collection = db['school']
46         serializer = BookSerializer(data=request.data)
47
48         if serializer.is_valid():
49             new_data = serializer.validated_data
50             result = booking_collection.insert_one(new_data)
51             new_data['_id'] = str(result.inserted_id)
```

```

52     # Send booking reference to the school's email
53     send_booking_ref_to_client(str(result.inserted_id), ['school_email@example.com'])
54
55     return Response(new_data, status=status.HTTP_201_CREATED)
56
57     return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
58
59 class BookingViewID(APIView):
60     def get(self, request, *args, **kwargs):
61         if 'id' not in kwargs:
62             return Response({'error': 'GET method expects an id'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
63
64         db = connect_mongodb()
65         booking_collection = db['booking']
66         checklist_collection = db['checklist']
67         school_collection = db['school']
68
69         booking_id = kwargs['id']
70         booking_document = booking_collection.find_one({'_id': ObjectId(booking_id)})
71
72         if booking_document:
73             enriched_booking = enrich_booking_data(booking_document, checklist_collection, school_collection)
74             serializer = BookSerializer(enriched_booking)
75             return Response(serializer.data, status=status.HTTP_200_OK)
76
77         return Response({'error': 'Booking not found'}, status=status.HTTP_404_NOT_FOUND)
78
79     def put(self, request, *args, **kwargs):
80         if 'id' not in kwargs:
81             return Response({'error': 'PUT method expects an id'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
82
83         db = connect_mongodb()
84         booking_collection = db['booking']
85         serializer = BookSerializer(data=request.data)
86
87         if serializer.is_valid():
88             update_result = booking_collection.update_one({'_id': ObjectId(kwargs['id'])}, {'$set': serializer.
89             if update_result.matched_count == 0:
90                 return Response({'error': 'No record found with the specified ID'}, status=status.HTTP_404_NOT_
91
92             return Response({'status': 'success', 'id': kwargs['id'], 'updated': update_result.modified_count},
93
94             return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
95
96     def delete(self, request, *args, **kwargs):
97         if 'id' not in kwargs:
98             return Response({'error': 'DELETE method expects an id'}, status=status.HTTP_405_METHOD_NOT_ALLOWED)
99
100        db = connect_mongodb()
101        booking_collection = db['booking']
102        delete_result = booking_collection.delete_one({'_id': ObjectId(kwargs['id'])})
103
104        if delete_result.deleted_count == 0:
105            return Response({'error': 'No record found with the specified ID'}, status=status.HTTP_404_NOT_FOUN
106
107        return Response({'status': 'success', 'id': kwargs['id']}, status=status.HTTP_200_OK)

```

Merge Request Feedback 4

Merge Request #38 - File: src/back-end/example/views/booking_view.py

Date: 19/05/2024

Participants: @Guixian Li @Yongli Qin

Code Review

1. Documentation Defects:

- a. Naming:

- The variable names and class names are generally clear (`BookingView`, `BookingViewID`). However, the use of generic names like `db`, `new_data`, and `serializer` could be more descriptive.

- b. Comment:

- Comments exist but are minimal. The code would benefit from more detailed comments explaining the logic, especially in complex sections like the `post` method in `BookingView`.

2. Visual Representation Defects:

- a. Bracket Usage:

- Bracket usage seems correct throughout the code.

- b. Indentation:

- The indentation is consistent and follows Python standards, which helps in maintaining readability.

- c. Long Line:

- Some lines, especially those with complex conditions or MongoDB queries, are quite long and could be broken down for better readability.

3. Structure Defects:

- a. Dead Code:

- There's a commented-out call to `send_booking_ref_to_client` in the `get` method of `BookingView` which should be removed if not needed.

- b. Duplication:

- The code to fetch `school_document` and `checklist_document` is repeated in both `get` and `put` methods of `BookingViewID`. This could be refactored into a separate method.

4. New Functionality:

- a. Use Standard Method:

- The handling of MongoDB ObjectIds and the conversion of these to strings for JSON serialization is repeated multiple times and could be standardized in a utility function.

5. Resource Defects:

- a. Variable Initialization:

- All variables are appropriately initialized.

- b. Memory Management:

- No explicit issues with memory management. Python's garbage collector handles most of the memory management.

6. Check Defects:

- **a. Check User Input:**

- The code checks if the `checklist_id` and `school_id` exist in the database before proceeding, which is good. However, more robust error handling and validation on user inputs could be implemented to enhance security and reliability.

7. Interface Defects:

- **a. Parameter:**

- Parameters in function calls are correctly used. However, the handling of parameters in API responses could be made more consistent, especially the error messages.

8. Logic Defects:

- **a. Compute:**

- The logic to handle different statuses and sending emails based on these conditions in the `put` method of `BookingViewID` can be made clearer and possibly refactored to reduce complexity.

- **b. Performance:**

- The performance seems acceptable, but as the database grows, the performance of fetching and updating documents should be monitored, especially since every booking fetch involves multiple database queries.

Summary of Recommendations:

- Enhance comments to improve maintainability and understanding.
- Refactor repeated code for fetching documents from the database into utility functions.
- Standardize and possibly centralize the handling of MongoDB ObjectId transformations.
- Implement more robust input validation and error handling.
- Consider performance implications of multiple database queries in single API calls as data scales.

These changes will help in making the code more maintainable, efficient, and easier to understand, aligning with the project's goal of creating a unified and efficient booking management system.

Based on the feedback,

we made some improvements to produce more professional code

- Refactor repeated code
- Add comments for clarity and documentation

Consider

The feedback provided is comprehensive and effectively targets key areas where the code could be improved in terms of readability, maintainability, and robustness. We can't really do much about the performance as it is very difficult to refactor some of the code as we already have many built on top of it.

Merge Request Feedback 5

Merge Request #38 - File: `src/back-end/example/views/chart_view.py`

Date: 20/05/2024

Participants: @Yongli Qin @Yun-Chi Hsiao @Daniel Su

1. Documentation Defects:

- a. Naming:
 - The naming of classes and methods is generally clear (`chartOneView`, `get`), indicating their purpose well. However, the specific chart names like `ChartOneView` could be more descriptive, indicating what the chart represents.
- b. Comment:
 - The code lacks in-line comments explaining complex logic, especially within the MongoDB aggregation pipelines. Adding comments would enhance maintainability and readability.

2. Visual Representation Defects:

- a. Bracket Usage:
 - The usage of brackets seems correct and consistent throughout the code.
- b. Indentation:
 - Indentation is consistent and follows Python's PEP 8 standards, which improves readability.
- c. Long Line:
 - Some lines, especially those containing MongoDB aggregation pipelines, are quite long and could be broken into multiple lines for better readability.

3. Structure Defects:

- a. Dead Code:
 - No apparent dead code is detected.
- b. Duplication:
 - The structure of the MongoDB queries and the response handling in each chart view are similar and could be refactored into a shared method to reduce duplication.

4. New Functionality:

- a. Use Standard Method:
 - Standard methods are used appropriately. However, the repeated pattern of database connection and query execution could be standardized into utility functions.

5. Resource Defects:

- a. Variable Initialization:
 - Variables are initialized appropriately.
- b. Memory Management:
 - The code does not explicitly handle memory management issues, but Python's garbage collector would manage unused objects.

6. Check Defects:

- a. Check User Input:

- There is no explicit user input validation in the provided code. It assumes that the database entries are correctly formatted and exist.

7. Interface Defects:

- a. Parameter:

- The function parameters are used correctly, and there are no apparent issues with missing or incorrect parameters.

8. Logic Defects:

- a. Compute:

- The logic for aggregating and processing data seems correct based on the MongoDB pipeline operations.

- b. Performance:

- While the MongoDB queries appear optimized with aggregation pipelines, performance could still be an issue with large datasets.
Proper indexing in MongoDB could help improve query performance.

Summary of Recommendations:

1. **Improve Naming:** Use more descriptive names for classes representing specific charts.
2. **Enhance Comments:** Add in-line comments to complex MongoDB aggregation pipelines to explain the logic.
3. **Reduce Code Duplication:** Refactor similar query and response handling code into shared methods.
4. **User Input Validation:** Add checks for user input where necessary, especially if user parameters are eventually introduced in queries.
5. **Performance Considerations:** Ensure MongoDB is properly indexed to handle potentially large datasets efficiently.

These changes would help improve the overall quality, maintainability, and performance of the code.

Based on the feedback,

we made some improvements to produce more professional code

- Improve Naming conventions
- reduce code duplication
- Add comments for clarity and documentation

Consider

The feedback provided is comprehensive and effectively targets key areas where the code could be improved in terms of readability, maintainability, and robustness. Some input validation is redundant, most input errors are caught in frontend validation or serializers. We can't really do much about the performance as it is very difficult to refactor some of the code as we already have many built on top of it.

Merge Request Feedback 6

Merge request #38 - File: `src/back-end/example/views/checklist_view.py`

Date: 20/05/2024

Participants: @Yuanbo Xu @Pangfeng ZHENG

1. Documentation Defects:

- **Naming:**

- The class and method names (`ChecklistView`, `ChecklistViewID`, etc.) are generally descriptive and appropriate. However, the name `ChecklistViewID` could be more specific to indicate its purpose more clearly, such as `ChecklistByIdView`.

- **Comment:**

- The code lacks comments explaining the purpose of each class and method. Adding comments would enhance maintainability and readability, especially for other team members or future contributors.

2. Visual Representation Defects:

- **Bracket Usage:**

- Brackets are used correctly throughout the code.

- **Indentation:**

- The indentation appears consistent and follows Python's PEP 8 guidelines, which enhances readability.

- **Long Line:**

- Some lines, especially those with MongoDB queries and responses, are quite long. Breaking these into multiple lines would improve readability.

3. Structure Defects:

- **Dead Code:**

- There doesn't appear to be any dead code in the provided snippet.

- **Duplication:**

- The code for connecting to MongoDB (`connect_mongodb()`) is repeated in multiple methods. This could be refactored into a separate method or setup to reduce duplication and facilitate changes.

4. New Functionality:

- **Use Standard Method:**

- The code could benefit from using Django's standard views and mixins for CRUD operations to reduce boilerplate code and enhance maintainability.

5. Resource Defects:

- **Variable Initialization:**

- Variables are appropriately initialized within their respective scopes.

- **Memory Management:**

- Python handles memory management internally, but ensuring that database connections are properly managed (opened and closed) could prevent potential memory leaks or connection saturation.

6. Check Defects:

- **Check User Input:**

- The code checks for the presence of 'id' in `kwargs` but does not validate the format or type of 'id' before using it to query the database. Adding input validation would enhance security and robustness.

7. Interface Defects:

- **Parameter:**

- Parameters are used correctly in function calls. However, better error handling for incorrect parameters could improve usability and error feedback.

8. Logic Defects:

- **Compute:**
 - The logic for CRUD operations is correctly implemented. However, ensuring atomicity and handling potential exceptions during database operations could enhance reliability.
- **Performance:**
 - The performance seems adequate for typical use cases, but indexing database fields used in queries could improve performance, especially as data grows.

Summary of Recommendations:

- Add detailed comments to improve code understanding and maintenance.
- Refactor MongoDB connection setup to reduce duplication.
- Implement input validation for all user inputs to enhance security.
- Consider using Django's class-based views and mixins for CRUD operations to reduce boilerplate code.
- Ensure proper error handling and feedback for all operations, particularly database interactions.
- Optimize MongoDB queries with appropriate indexing to improve performance.

Based on the feedback,

we made some improvements to produce more professional code

- Refactor Mongodbc Connection code
- Improve error handling feedback
- Add comments for clarity and documentation

Consider

The feedback provided is comprehensive and effectively targets key areas where the code could be improved in terms of readability, maintainability, and robustness. We can't really do much about the Mongodbc query optimisation as it is very difficult to refactor some of the code as we already have many built on top of it.

Merge Request Feedback 7

Merge request # 38 - file: src/back-end/example/tests/test_charts.py

Date: 20/05/2024

Participants: @Guixian Li @Yun-Chi Hsiao @Yuanbo Xu

1. Documentation Defects:

- **a. Naming:** The naming of the test methods (`test_chart_1`, `test_chart_2`, etc.) is somewhat generic. It would be more informative if the names could reflect the specific functionality or feature they are testing.
- **b. Comment:** The comment provided in the code mentions the database context (`test` vs. `unit_test`), which is helpful. However, more comments explaining the purpose of overriding `_fixture_teardown` or the significance of each test would enhance understanding.

2. Visual Representation Defects:

- **a. Bracket Usage:** Brackets usage in the provided code snippet is correct.
- **b. Indentation:** Indentation is consistent and follows Python's PEP 8 standards, which enhances readability.
- **c. Long Line:** No lines exceed a reasonable length, maintaining readability.

3. Structure Defects:

- **a. Dead Code:** There does not appear to be any dead code in the snippet provided.
- **b. Duplication:** The code for performing the HTTP GET request and checking the response status is repeated in each test method. This could be refactored into a separate method to reduce duplication and improve maintainability.

4. New Functionality:

- **a. Use Standard Method:** The tests use Django's `reverse` and `status` which are standard. However, considering the use of MongoDB, ensuring that the MongoDB interactions are using best practices would be beneficial (not explicitly shown in the snippet).

5. Resource Defects:

- **a. Variable Initialization:** All variables are appropriately initialized in the provided code.
- **b. Memory Management:** There is no explicit memory management issue visible in the snippet. However, ensuring that the MongoDB connections are properly managed (opened and closed) would be important.

6. Check Defects:

- **a. Check User Input:** The tests do not show any user input validation directly. It's assumed that input validation is handled elsewhere in the application.

7. Interface Defects:

- **a. Parameter:** The use of parameters in function calls like `reverse` is correct. Ensuring that the URLs mapped to `chart_1`, `chart_2`, etc., are correctly defined in Django's URL configuration would be necessary.

8. Logic Defects:

- **a. Compute:** The logical flow for fetching and asserting the status of HTTP responses is correct.
- **b. Performance:** There are no obvious performance issues in the snippet. However, the efficiency of the database queries (not shown here) would significantly impact performance.

Summary of Recommendations:

1. Improve test method names to reflect the specific functionality they are testing.
2. Refactor repeated code for making HTTP GET requests into a separate method to reduce duplication.
3. Ensure proper management of MongoDB connections.
4. Consider adding more detailed comments to explain the purpose of methods and any non-trivial logic.
5. Validate and handle user input appropriately if not already done.

6. Confirm that URLs referenced in the `reverse` function calls are correctly configured in Django's URL patterns.

Based on the feedback,

we made some improvements to produce more professional code

- Refactor repeated code
- Improve test method
- Add comments for clarity and documentation

Consider

The feedback provided is comprehensive and effectively targets key areas where the code could be improved in terms of readability, maintainability, and robustness. Validation are mostly done in the serializer itself, so we did not add more validation in this case. Test MongoDB is created when "TESTING" mode is on.

Merge Request Feedback 8

Merge request #38 - file: `src/back-end/example/tests/test_misc_create.py`,

`src/back-end/example/tests/test_miscellaneous.py`

Date: 20/05/2024

Participants: [@Yuanbo Xu](#) [@Guixian Li](#) [@Daniel Su](#)

1. Documentation Defects:

- **Naming:**

- The class and method names (`DBTest`, `test_create`) are appropriately descriptive of their functionality.
- The MongoDB collection name `miscellaneous` could be more specific based on its intended use.

- **Comment:**

- The comment explaining the override of `_fixture_teardown` is helpful.
- Additional comments explaining the purpose of each test and the structure of the data being tested would enhance understandability.

2. Visual Representation Defects:

- **Bracket Usage:**

- Brackets are used correctly throughout the code.

- **Indentation:**

- The code is properly indented, which improves readability.

- **Long Line:**

- The `data` dictionary in the `test_create` method contains lines that are quite long and could be broken down for better readability.

3. Structure Defects:

- **Dead Code:**

- There is no dead code present in the snippet provided.

- **Duplication:**

- No duplication is evident in the provided code.

4. New Functionality:

- **Use Standard Method:**

- The test uses Django's built-in `APITestCase` and MongoDB's client effectively, which are standard methods for such operations.

5. Resource Defects:

- **Variable Initialization:**

- All variables are initialized appropriately.

- **Memory Management:**

- The provided snippet does not show any explicit memory management issues, but it's crucial to ensure that database connections are managed (opened and closed) properly in the larger context of the application.

6. Check Defects:

- **Check User Input:**

- The test does not include input validation checks, assuming that the data provided is correct. It's generally a good practice to validate test inputs to ensure robustness.

7. Interface Defects:

- **Parameter:**

- The parameters used in function calls (`reverse`, `post`, etc.) are correctly applied.

8. Logic Defects:

◦ Compute:

- The logic for testing the creation of a document in MongoDB appears correct.

◦ Performance:

- For this specific test case, performance is not a primary concern, but for larger datasets or more complex queries, performance considerations (like indexing in MongoDB) should be addressed.

Summary of Recommendations:

- **Enhance Documentation:** Add more detailed comments explaining the purpose and functionality of each part of the test.
- **Improve Code Readability:** Consider breaking down long lines in the `data` dictionary for better readability.
- **Input Validation:** Include input validation in tests to ensure data integrity and robustness.
- **Memory Management:** Ensure proper management of database connections, especially in a larger application context.
- **Performance Considerations:** For more extensive testing or larger datasets, consider performance optimizations like proper indexing in MongoDB.

File: `src/back-end/example/tests/test_miscellaneous.py`

1. Documentation Defects:

• a. Naming:

- The naming of the class `DBTest` is sufficiently clear, indicating that the class is meant for database-related tests.
- The method names (`test_get_all`, `test_put`, `test_put_not_valid`) are descriptive and reflect their functionality well.

• b. Comment:

- Comments are minimal but adequate for understanding the purpose of overridden methods and test cases. Adding more detailed comments could enhance understanding, especially for complex logic or when interfacing with external systems like databases.

2. Visual Representation Defects:

• a. Bracket Usage:

- No issues observed with bracket usage.

• b. Indentation:

- The code is consistently indented, which helps in maintaining readability.

• c. Long Line:

- The data dictionary in the `test_put` and `test_put_not_valid` methods contains long lines that could be broken down for better readability.

3. Structure Defects:

• a. Dead Code:

- No dead code detected.

• b. Duplication:

- The data dictionaries in `test_put` and `test_put_not_valid` methods are very similar and could potentially be refactored into a shared method or attribute to reduce duplication.

4. New Functionality:

• a. Use Standard Method:

- The code uses standard testing methodologies provided by Django's `APITestCase`, which is appropriate.

5. Resource Defects:

- a. Variable Initialization:
 - All variables are initialized appropriately.
- b. Memory Management:
 - Python handles memory management internally, and there's no explicit misuse of resources in the provided code snippet.

6. Check Defects:

- a. Check User Input:
 - The `test_put_not_valid` method checks for invalid input by expecting a 400 response, which is a good practice. However, it could be enhanced by checking specific error messages or codes to ensure that the failure is for the expected reasons.

7. Interface Defects:

- a. Parameter:
 - The use of `reverse` for URL resolution in Django is correctly implemented. Parameters for API calls are appropriately used.

8. Logic Defects:

- a. Compute:
 - The logic for the tests seems correct based on the provided code.
- b. Performance:
 - The code does not include any complex computations or algorithms that could hinder performance. However, continuous integration testing should monitor the performance as more features are added.

Summary of Recommended Changes:

- Enhance Comments:** Add more detailed comments explaining the purpose of the test cases, especially when interfacing with the database.
- Refactor Data Dictionaries:** Consider refactoring the data dictionaries in `test_put` and `test_put_not_valid` to reduce duplication and improve maintainability.
- Improve Input Validation Checks:** Enhance the `test_put_not_valid` method by checking specific error messages or codes to ensure precise failure reasons are captured.
- Break Down Long Lines:** Improve readability by breaking down long lines, especially in data dictionaries

Based on these two feedback,

we made some improvements to produce more professional code

- Refactor code
- Improve test method
- Break long lines for readability
- Add comments for clarity and documentation

Consider

The feedback provided is comprehensive and effectively targets key areas where the code could be improved in terms of readability, maintainability, and robustness. Validation are mostly done in the serializer itself, so we did not add more validation in this case. Test MongoDB is created when “TESTING” mode is on.

Merge Requests Feedback 9

Merge request #38 - file: src/back-end/example/tests/test_search.py

Date: 20/05/2024

Participants: @Yongli Qin @Pangfeng ZHENG

1. Documentation Defects:

- **Naming:**

- The class name `DBTest` is somewhat generic. A more descriptive name like `SearchQueryDBTest` might better reflect the specific functionality being tested.
- Method names such as `test_search_query` and `test_search_non_exist` are clear and appropriately describe the tests being performed.

- **Comment:**

- The comment `# sample query` is minimal and could be more descriptive about what the query is testing.
- Comments explaining the purpose of overriding `_fixture_teardown` and the use of `settings.TESTING` in `setUpClass` would be beneficial for maintainability.

2. Visual Representation Defects:

- **Bracket Usage:**

- No issues with bracket usage are observed in the snippet.

- **Indentation:**

- The indentation is consistent and follows Python's PEP 8 style guide.

- **Long Line:**

- No excessively long lines are present that would hinder readability.

3. Structure Defects:

- **Dead Code:**

- There is no apparent dead code in the snippet.

- **Duplication:**

- No duplication is evident within the provided code.

4. New Functionality:

- **Use Standard Method:**

- The code uses Django's standard testing mechanisms and MongoDB's client appropriately. No changes are necessary here.

5. Resource Defects:

- **Variable Initialization:**

- All variables are initialized properly.

- **Memory Management:**

- Python and Django handle memory management, and there are no apparent issues with how resources are handled in the test cases.

6. Check Defects:

- **Check User Input:**

- The test `test_search_non_exist` checks for missing query parameters and handles it by expecting a 400 status code, which is appropriate.

7. Interface Defects:

- **Parameter:**

- The `reverse('search')` function assumes the existence of a URL named 'search'. It's crucial to ensure that this URL is correctly defined in the Django URL configuration.

8. Logic Defects:

- **Compute:**

- The logic for handling different scenarios (existent and non-existent queries) is correctly implemented.

- **Performance:**

- Given the limited context, it's difficult to evaluate performance. However, for unit tests, performance seems adequate.

Summary of Recommendations:

- **Improve Documentation:** Add more descriptive comments, especially explaining why certain overrides are used and detailing the setup process.
- **Enhance Naming Conventions:** Consider renaming `DBTest` to something more descriptive of its specific function.
- **Interface Assurance:** Ensure that the URL name 'search' is correctly configured in the Django URLs.

Based on the feedback,

we made some improvements to produce more professional code

- Improve naming conventions
- Add comments for clarity and documentation

Consider

The feedback provided is comprehensive and effectively targets key areas where the code could be improved in terms of readability, maintainability, and robustness. The URL name is listed in `url.py`. Otherwise, most recommendation are followed by the team.

Frontend Code Review Sprint #3

Total code review times: 6 (sprint 3)

Code Review Front-end #1 (Merge request #30)

Date: 6/05/2024

Merge Request From **Frontend/features/search** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check if the search can work properly
- Check the other features is working

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #2 (Merge request #31)

Date: 6/05/2024

Merge Request From **Frontend/features/checklist** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check if the check list is implemented into booking page
- Check if the check list can be updated and deleted
- Check if the check list is matched with the templates

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #3 (Merge request #33)

Date: 13/05/2024

Merge Request From **Frontend/features/analytics** to **Main**

Participants: [@Yun-Chi Hsiao](#) [@Yongli Qin](#) [@Daniel Su](#)

Check list:

- Check diagram is showed
- Check if all the data is loaded properly
- Check if the other features is working

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #4 (Merge request #35)

Date: 13/05/2024

Merge Request From **Frontend/features/analytics** to **Main**

Participants: @Yun-Chi Hsiao @Yongli Qin @Daniel Su

Check list:

- Check if the database is connected with the analytics page
- Check if all the data is loaded properly
- Check if the other features is working

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #5 (Merge request #36)

Date: 15/05/2024

Merge Request From **Frontend/features/calendar** to **Main**

Participants: @Yun-Chi Hsiao @Yongli Qin @Daniel Su @Pangfeng ZHENG

Check list:

- Check if the calendar can display the booking information
- Check if all the booking is clickable
- Check if the other features is working

Number of Issues : 0 (Standards compliant)

Action after code review: None

Code Review Front-end #6 (Merge request #37)

Date: 19/05/2024

Merge Request From **Frontend/features/createNewBooking** to **Main**

Participants: @Yun-Chi Hsiao @Yongli Qin @Daniel Su

Check list:

- Check if new booking can be created by using auto fill function
- Check if the information of exist booking can be display
- Check if the other features is working
- Check if the booking can be deleted and updated

Number of Issues : 0 (Standards compliant)

Action after code review: None

Backend Code Review Sprint #3

Total code review times: 1 (sprint 3)

Code Review Back-end #1 (Merge request #34)

Date: 15/05/2024

Merge Request From **backend/features/databaseAPI** to **Main**

Participants: @Guixian Li @Yuanbo Xu @Pangfeng ZHENG

Check list:

- Check if the new url can working properly
- Check if the information that frontend needed is transferred

Number of Issues : 0 (Standards compliant)

Action after code review: None

TEMPLATE

 [Template - Decision documentation](#)

 [Template - Product requirements](#)

 [Template - Meeting notes](#)

Template - Decision documentation

Status	NOT STARTED / IN PROGRESS / COMPLETE
Impact	HIGH / MEDIUM / LOW
Driver	
Approver	
Contributors	
Informed	
Due date	
Resources	

Relevant data

Background

Options considered

	Option 1	Option 2
Description		
Pros and cons	 	 
Estimated cost	LARGE	MEDIUM

Action items



Outcome

✍️ Template - Product requirements

Target release	Type // to add a target release date
Epic	Type /Jira to add Jira epics and issues
Document status	DRAFT
Document owner	@ mention owner
Designer	@ designer
Tech lead	@ lead
Technical writers	@ writers
QA	

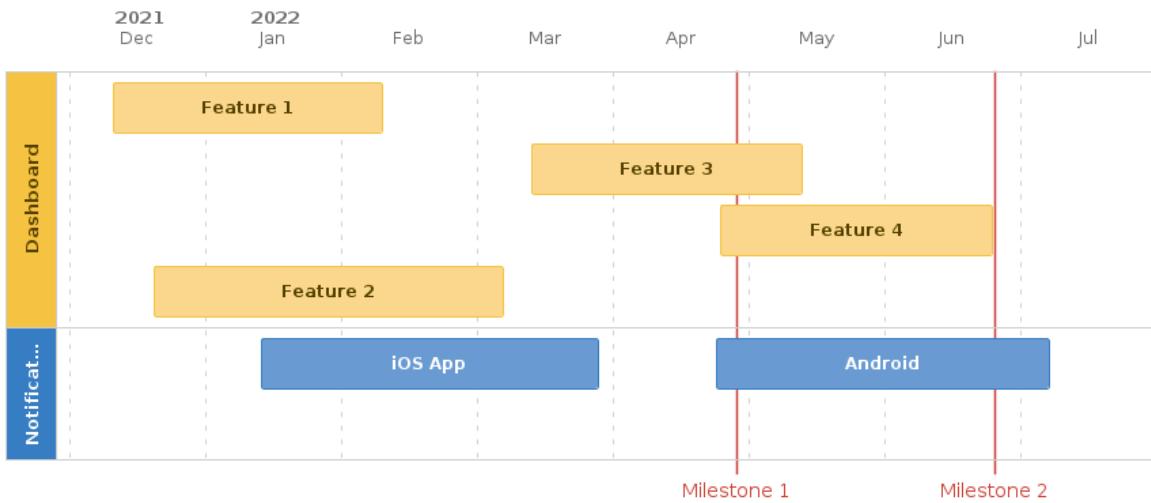
🎯 Objective

📊 Success metrics

Goal	Metric

🤔 Assumptions

🌟 Milestones



Requirements

Requirement	User Story	Importance	Jira Issue	Notes
		HIGH		

User interaction and design

Open Questions

Question	Answer	Date Answered

Out of Scope

-

Template - Meeting notes

 Date

Participants

-
-

Goals

-

Discussion topics

Time	Item	Presenter	Notes
			•

Action items

-

Decisions





LINKS & RESOURCES

- **Trello** [COMP90082_2024_SM1_SG_KOALA](#)
- **GitHub** [https://github.com/COMP90082-2024-SM1/SG-Koala](#) Connect your Github account
- **Figma** [Booking Management System](#)
- **Priava API**
 - <https://priavasupportcenter.ungerboeck.com/hc/en-us/categories/10843507387287-API-Integrations>



PRESENTATION



PRODUCT USER MANUAL



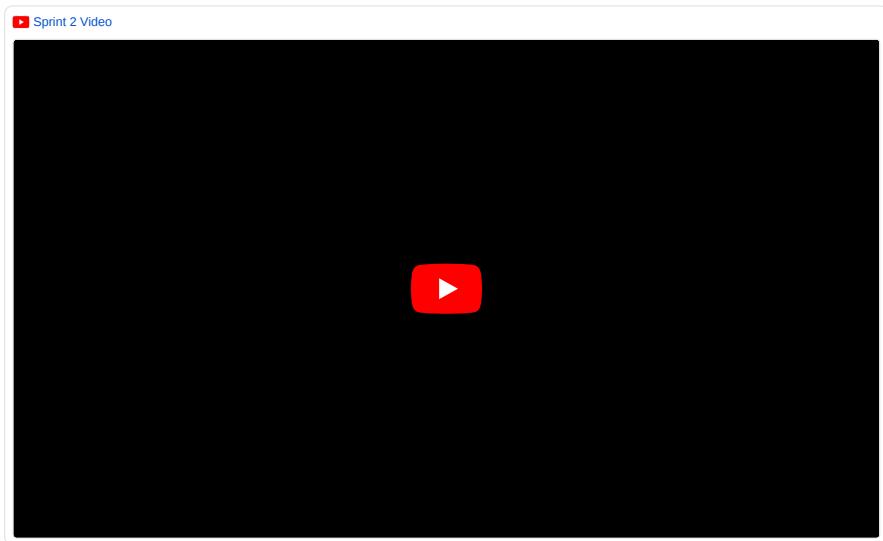


SPRINT2 DEMO

for Sprint 3 please use the URL link in our GitHub README

Sprint 2 demo: Link to demonstration video:

<https://www.youtube.com/watch?v=H5LxiF4b4GY&feature=youtu.be>



📺 FINAL DEMO

