TRAVEL PACKAGE WEBSITE

A Project Report submitted by

DON P D
(VAS22MCA-2020)

to APJ Abdul Kalam Technological University
in partial fulfillment of the requirements for the award of the degree of
Master of Computer Applications

Department of Computer Applications
Vidya Academy of Science & Technology
Thalakkottukara, Thrissur - 680 501
April 2024

Department of Computer Applications
Vidya Academy of Science & Technology
Thalakkottukara, Thrissur - 680 501
(http://www.vidyaacademy.ac.in)

CERTIFICATE

This is to certify that the report titled **Travel Package Website** is a bona-fide record of the work related to the paper 20MCA246 Project and Viva Voce done by **DON P D (Reg. No. VAS22MCA-2020)** of S4 MCA (2022 admissions) class of Vidya Academy of Science & Technology, Thrissur - 680501 in partial fulfillment of the requirement for the award of the Degree of Master of Computer Applications of APJ Abdul Kalam Technological University.

Guide/Supervisor
Name : Dr Sajay K R
Signature : .....
Date : April 29, 2024

Head of Department
Name : Dr Reji C Joy
Signature : .....
Date : April 29, 2024

(Seal of Department of Computer Applications)

External Supervisor
Name : .....
Signature : .....
Date : .....

Declaration

I, **DON P D**, studying in Fourth Semester MCA (2022 admissions) class of Vidya Academy of Science & Technology, Thrissur – 680501, hereby declare that the project report ("Travel Package Website") submitted by me for partial fulfillment of the requirements for the award of degree of Master of Computer Applications of APJ Abdul Kalam Technological University, Kerala, is a bona fide work done by me under supervision of Dr Sajay K R. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place	: Thrissur – 680501	Signature of student	: .....
Date	: April 29, 2024	Name of student	: DON P D

Acknowledgment

I wish to record my indebtedness and thankfulness to all who helped me prepare this report titled Travel Package Website and present it in a satisfactory way. This Report is part of my work related to the paper 20MCA246 Project and Viva Voce.

I am especially thankful to my guide and supervisor Dr Sajay K R in the Department of Computer Applications for giving me valuable suggestions and critical inputs in the preparation of this report. I am also thankful to Dr Reji C Joy, the Head of Computer Applications for encouragement.

My friends in my class have always been helpful and I am grateful to them for patiently listening to my presentations on my work related of the project.

DON P D
Reg. No. VAS22MCA-2020
S4 MCA (2022 Admissions)
Vidya Academy of Science & Technology
Thrissur -680 501.

Synopsis

The Travel Package Website revolutionizes the way travelers plan their adventures, presenting a sleek and intuitive platform that caters to the needs of modern explorers. Its user-friendly interface simplifies the process of discovering diverse destinations, crafting personalized itineraries, and seamlessly booking accommodations, transportation, and activities. By leveraging cutting-edge web technologies and responsive design principles, the website ensures a smooth and accessible experience across a range of devices, empowering users to plan their trips with ease from start to finish.

What sets the Travel Package Website apart is its strong emphasis on responsible travel and sustainability. Through curated destination guides and educational resources, the platform encourages users to make informed choices that support eco-conscious options and contribute positively to the places they visit. This commitment to promoting environmental awareness and ethical travel practices fosters a deeper connection between travelers and the destinations they explore, aligning with the growing trend of mindful and purposeful travel experiences.

Furthermore, the Travel Package Website thrives as a vibrant community hub for travelers worldwide. Its user-generated content feature enables individuals to share their experiences, recommendations, and insights, creating an enriched ecosystem of authentic reviews and ratings. This collaborative space not only helps users make well-informed decisions but also facilitates connections with like-minded individuals, enhancing the overall travel experience and fostering a sense of camaraderie among adventurers.

Contents

List of Tables

List of Figures

PROJECT REPORT

Chapter 1

INTRODUCTION

In an era where travel is more accessible and diverse than ever before, the need for a seamless and personalized trip planning experience has never been greater. Enter the Travel Package Website, a transformative online platform poised to redefine how modern travelers explore, plan, and embark on their adventures. With its innovative blend of cutting-edge web technologies, responsive design, and a strong focus on responsible travel practices, this website represents a paradigm shift in the travel industry.

The Travel Package Website isn't just another booking platform; it's a comprehensive solution designed to cater to the evolving needs and preferences of today's globetrotters. By offering a user-friendly interface that facilitates effortless exploration of diverse destinations, customizable itinerary creation, and hassle-free reservation processes for accommodations, transportation, and activities, this platform ensures that every aspect of travel planning is streamlined and enjoyable.

1.1 Project Overview

The Travel Package Website project is a multifaceted endeavor aimed at transforming the landscape of travel planning for modern adventurers. At its core, the project seeks to create an intuitive and user-friendly interface that simplifies the process of exploring destinations, crafting personalized itineraries, and making reservations seamlessly. Embracing sustainability and responsible travel, the project incorporates comprehensive guides and educational resources to empower users to make informed choices that align with eco-conscious practices. Furthermore, the project emphasizes community engagement through user-generated content features, fostering a dynamic space for travelers to share experiences, insights, and recommendations. By leveraging cutting-edge web technologies and collaboration opportunities with industry partners, the Travel Package Website project endeavors to set a new standard in personalized, efficient, and mindful travel planning experiences.

Chapter 2

SYSTEM ANALYSIS

2.1 The Existing System

The current iteration of the Travel Package Website serves as a reliable platform for travelers to explore destinations, customize itineraries, and make reservations. Its user-friendly interface allows users to easily navigate through various options for accommodations, transportation, and activities, streamlining the trip planning process. The website already incorporates some sustainability features, such as highlighting eco-conscious accommodations and providing basic travel tips for responsible tourism. Additionally, the platform includes a community section where users can share their travel experiences and recommendations, adding a social element to the overall user experience. While the existing website provides valuable services, there is room for enhancement in terms of expanding sustainability initiatives, improving user engagement features, and integrating advanced technologies for a more seamless experience.

2.2 Proposed System

The proposed system for the Travel Package Website envisions a user-friendly interface that empowers modern travelers to seamlessly explore diverse destinations, customize their itineraries, and make reservations with ease. This interface, designed to be intuitive and accessible across various devices, ensures a smooth and enjoyable user experience. Users will have the flexibility to tailor their travel plans according to their preferences and interests, selecting accommodations, transportation options, and activities that suit their needs. The integration of advanced web technologies will provide real-time information, interactive maps, and multimedia content for destinations, enhancing user engagement and delivering comprehensive travel insights. Moreover, the system will promote responsible travel by highlighting sustainable and eco-conscious options, encouraging users to make environmentally friendly choices. Additionally, a commu-

nity space within the website will allow travelers to share their experiences, recommendations, and insights, fostering connections among like-minded individuals and facilitating well-informed travel decisions. Overall, the Travel Package Website aims to combine technical proficiency with personalized and efficient travel planning, creating a deeper connection between travelers and the destinations they explore while advocating for sustainable travel practices.

Chapter 3

FEASIBILITY STUDY

A feasibility study is a systematic process that involves identifying, describing, and evaluating potential systems, ultimately selecting the most suitable one for a specific task. It involves assessing whether current software and hardware technologies can meet the identified user requirements. The study aims to determine the feasibility and cost-effectiveness of the proposed system from a business perspective, including assessing whether it can be developed within the allocated budget. Key aspects considered during the feasibility analysis include economic viability, technical feasibility, and operational feasibility.

3.1 Technical Feasibility

Cost/Benefit Analysis, often referred to as Economic Analysis, is a process aimed at evaluating the anticipated benefits and savings of a prospective system and contrasting them with associated costs. This assessment helps determine the economic feasibility of the system, benefiting both the developer and the client. Given the available information, it is increasingly apparent that the proposed system holds economic viability for both parties involved.

3.2 Economical Feasibility

Economic analysis, commonly known as cost/benefit analysis, is a crucial step in evaluating system effectiveness. This process involves assessing the expected benefits and savings of a system and comparing them against the associated costs. The decision to design and implement a system is based on this analysis. It provides top management with the economic justification for adopting a new system, which is valuable as they may prefer straightforward economic justifications over technical complexities. In the case of the proposed system, it is economically feasible as it requires only a single operator to manage the system. This operator is responsible for data entry

into the database through a user interface and can also present data in HTML tabular form, providing information about students either seeking admission or already admitted. This streamlined operation reduces operational costs significantly.

Additionally, implementing this system does not necessitate additional hardware resources and saves considerable time for the organization. These factors contribute to the organization's high satisfaction with the economic feasibility of the system, making it a viable and beneficial investment.

3.3 Operational Feasibility

It is Operational feasible, since the system is providing a attractive user interface to the operator/ end user, so they feel very easy to work onto it. Response to operator/end user is very fast and very good. Since, as we mentioned above that it requires much less amount of cost, it uses computer work so it is very fast to operate and it is very easy for user to work on it.

Chapter 4

SYSTEM DESIGN

The system design phase is crucial in developing the detailed architecture required for building a system or product. Just like any systematic approach, this software has undergone rigorous design phases to fine-tune efficiency, performance, and accuracy levels. The initial step in system design involves determining how the output will be generated and in what format. Examples of both input and output are also provided during this phase.

In the subsequent step, the design focuses on creating input data structures and master files that align with the requirements of the desired output. The processing phases are managed through program construction and testing, which includes creating a list of programs necessary to achieve the system's objectives and thorough documentation to ensure clarity and completeness. This systematic approach ensures that the system is designed to meet its intended purpose effectively and efficiently.

4.1 Input Design

Input design serves as the crucial link between an information system and its users. It involves developing specifications and procedures for preparing data, ensuring that transaction data is in a usable format for processing and data entry. Data can be input into the system by inspecting the computer to read data from written or printed documents, or it can involve individuals directly entering data into the system.

The primary focus of input design is to control the amount of input required, minimize errors, avoid delays and extra steps, and maintain a simple and efficient process. The system relies on various data related to asset items, depreciation rates, asset transfers, and physical verification for validation, checking, calculations, and report generation.

Additionally, the input design includes an error handling mechanism within the software. This mechanism detects wrong entries of input and raises error messages, ensuring data accuracy and

integrity throughout the processing stages.

4.2 Output Design

Quality output is pivotal, meeting user requirements and conveying information clearly. Outputs in a system serve as bridges, communicating processed results to users and other systems. In output design, clarity in information display for immediate use and in hard copy format is established, being a primary information source for users, augmenting decision-making processes. Intelligent output design enhances system-user relationships, streamlining decision-making. An organized approach to computer output design ensures user-friendly and effective output elements. During analysis and design, emphasis is on identifying specific outputs, selecting optimal presentation methods, and creating documents, reports, or formats that effectively convey system-generated information. This meticulous process ultimately enhances user satisfaction and system usability.

4.3 Database Design

Database design is a critical aspect of creating an efficient and scalable data management system. It involves strategically organizing data based on the chosen database model, which in the case of MongoDB, follows a document-oriented approach. The design process begins with identifying essential data elements and their relationships, resulting in a structured representation known as an ontology. MongoDB's schema flexibility allows for dynamic schemas, accommodating changes and evolving data requirements seamlessly.

In MongoDB, data is stored in flexible JSON-like documents called BSON, allowing for efficient storage and retrieval. Indexing plays a crucial role in optimizing query performance, with support for various types of indexes like single-field, compound, geospatial, and text indexes. These indexes enable fast data retrieval based on indexed fields, reducing search times and improving overall system performance.

Scalability is another key aspect of MongoDB's database design, with support for horizontal scaling through sharding. Sharding distributes data across multiple shards, allowing the database to handle large volumes of data and high traffic loads efficiently. This scalability ensures that MongoDB databases can grow seamlessly as data requirements increase.

Furthermore, MongoDB's document validation feature allows for defining rules and constraints within the database, ensuring data integrity and preventing inconsistencies. This contributes to efficient data updates and maintenance, minimizing the risk of database bloat and ensuring data accuracy.

Chapter 5

TESTING AND VALIDATION

Testing and validation are fundamental aspects of the Travel Package Website project, ensuring that the system meets requirements and functions as intended. Testing involves executing the website and verifying its performance against expectations, assessing components against specified criteria. Different types of testing, such as unit testing, integration testing, and acceptance testing, ensure the website's functionality is correct and efficient. Validation, on the other hand, confirms that the system meets user needs, involving activities like testing, review, and inspections.

In the context of the Travel Package Website project, testing and validation are vital for providing accurate information to travelers and stakeholders. These processes identify and address defects, errors, and bugs, allowing developers to improve the website's performance before launch. Ultimately, testing and validation enhance the website's reliability and efficiency, ensuring it functions as intended for a seamless user experience.

5.1 Testing Methods

System testing is a critical stage in the implementation process, focused on ensuring the system operates accurately and efficiently before live operations begin. In the development of newly created software, testing holds paramount importance as it represents the final opportunity for developers to address any issues before delivering the product to customers. Testing involves generating a comprehensive set of test data to uncover various types of errors that may arise in the software, contributing significantly to the system's success.

System testing encompasses executing the program with the aim of identifying errors, verifying completeness, and assessing whether user requirements are met. The primary objective is quality assurance, where tests are conducted and results compared against expected outcomes. In cases of discrepancies, debugging is conducted. A detailed testing strategy involves a test plan for each module, with system testing comprising a series of tests aimed at fully evaluating the

computer-based system against external specifications.

5.1.0.1 Unit Testing

This is the first level of testing. In this different modules are tested against the specification produces during the design of the modules. Unit testing is done during the coding phase and to test the internal logic of the modules. It refers to the verification of single program module in an isolated environment. Unit testing first focuses on the modules independently of one another to locate errors. After coding each dialogue is tested and run individually. All necessary coding where removed and it was ensured that all the modules are worked, as the programmer would expect. Logical errors found where corrected. So, by working all the modules independently and verifying the outputs of each module in the presence of staff we conducted that the programs was functioning as expected

5.1.0.2 Integration Testing

Data can be lost across an interface: one module can be adverse effort on another; sub functions when combined may not produce the desired major functions. Integration testing is a systematic testing for constructing the program structure. Conducting the tests is to uncover errors associated within the interface. The objective is to take unit tested to modules and build a program structure. All the modules are combined and tested as whole. Here correction is difficult because the vast expenses of the entire program complicate the isolation of causes. Thus in the integration testing step, all the errors uncovered are corrected for the next testing steps.

5.1.0.3 Output Testing

After performing the validation, the next step is output testing of the proposed system since no system could be useful if it does not produce the required output in the specific format. The output generated or displayed by the system under consideration is tested asking the users about the format required by then. Here, the output is considered into ways: one is on the screen and the other is printed format. The output format on the screen is found to be correct as the format designed according to the user needs. For the hard copy also, the output comes out as specified by the user

5.1.0.4 Security Testing

Security testing refers to ensuring that the system will protect its data from unauthorized access and modification and that it will continue to behave as expected. Different security concepts have been used to ensure that the system is secure. Authentication has been used to confirm the identity

of someone being either a user or an administrator. Authorisation has been used to ensure that users have access to pages that they are supposed to and that the administrator has an overall control of the system.

5.1.0.5 Black Box Testing

Black Box Testing also called behavioral testing, focuses on the functional requirement of software. It is complementary approach that is likely to uncover a different class of errors than white box method. Black Box testing is done to find out the following information:

- Incorrect or missing functions
- Interface errors
- Errors or database access
- Performance error
- Termination error

The mentioned testing is carried out successfully for this application according to the user's re- quirement specification.

5.1.0.6 White Box Testing

White Box Testing sometimes called glass box testing is test case design method that goes the control structure of the procedural design to derive test cases. Using white box testing methods, the software engineer can derive test cases that:

- Guarantee that all independent paths with in a module have been exercise at least once.
- Exercise all logical decision on their true and false rides.
- Executes all loops at their boundaries and within their operational boundaries
- Exercise internal data structure to ensure their validity.

5.2 Validation

Validation is the crucial process of assessing the final software product or system to determine if it aligns with specified requirements. Its primary aim is to confirm that the software system fulfills user needs and is usable by end-users. Validation encompasses a range of activities, including testing, review, walkthroughs, and inspections, all aimed at verifying that the software system meets predefined requirements. This step is integral in the software development life cycle, ensuring the software system achieves high quality and meets user expectations.

Chapter 6

SYSTEM SECURITY MEASURES

The protection of computer based resources that includes hardware, software, data, procedures and people against unauthorized use or natural Disaster is known as System Security. System Security can be divided into the following :

- physical: It refers to the technical innovations and procedures applied to the hardware and operation systems to protect against deliberate or accidental damage from a defined threat. Data security is the protection of data from loss, disclosure, modification and destruction.
- Operating System: No matter how to secure the system is, weakness in operating system security may serves as a means of unauthorized access to the network. Here Windows 10 as an operating system provides better level of security
- Network: Since almost all network system allows remote access through terminals and networks, softwarelevel security within the network software is important. Network security can be attained by setting firewall and security options available with windows.
- Database System: MongoDB, as a database system, offers robust security measures to protect data from unauthorized access and loss. MongoDB provides built-in user management features, allowing administrators to define user roles and permissions, thereby controlling access to databases and collections. Additionally, MongoDB supports encryption mechanisms for data at rest and in transit, ensuring that sensitive information remains secure even if accessed by unauthorized parties. Moreover, MongoDB offers a restore facility that enables users to recover data in case of accidental deletion or corruption, enhancing data reliability and availability. These security features make MongoDB a reliable choice for safeguarding data integrity and confidentiality in modern applications.
- Password: Password is the main security system for our site. This authorization mechanism limits the interaction with the resources to collection of user or system for enforcing in-

tegrity, confidentiality or availability constraints. There will be secret password for the user of the system. Hence, authorized users cannot access data of the system.

- System maintenance: Computer based business systems are dynamic. They must be able to accommodate changes in the business environment. These changes occur not only during the study, design, and development phases of the life cycle of the system, but also throughout its operational life. Maintenance involves the software industry captive, typing up the system resources. It means restoring something to its original condition. Maintenance involves a wide range of activities including correcting, coding and design errors, updating documentation and test data and upgrading user support. Maintenance is continued till the product is re-engineered or deployed to another platform. Maintenance is also done based on fixing the problems reported, changing the interface with other software or hardware enhancing the software.Security measures are provided to prevent unauthorized access of the database at various levels. An uninterrupted power supply should be so that the power failure or voltage fluctuations will not erase the data in the files. We can define maintenance by fair different maintenance activities.

  – Corrective maintenance.

  – Adaptive maintenance.

  – Perfective maintenance or enhancement.

  – Preventive maintenance or reengineering.

- Correction: Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct.

- Adaptation: Over time, the original environment (e.g. CPU, Operating System, Business Rules, and External Product Characteristics) for which was developed was likely to change. Adaptive maintenance results in the modification to the software to accommodate changes to its external environment.

- Enhancement: As the software is used, the user will recognize additional functions that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.

- Prevention: Computer software deteriorates due to change, and because of this, preventive maintenance, often called software reengineering, must be conducted to enable preventive maintenance makes changes to computer programs. So that they can be more easily corrected, adapted and enhanced.

Chapter 7

CONCLUSION

In conclusion, the Travel Package Website represents a significant step forward in the realm of travel planning and booking. With its user-friendly interface, diverse options for accommodations and activities, and a strong focus on community engagement, the current version of the website already provides valuable services to travelers. However, by implementing proposed enhancements that prioritize sustainability, expand destination guides, and enhance interactive features, the Travel Package Website is poised to become a leading platform for eco-conscious travelers seeking meaningful and immersive experiences. These improvements not only elevate the user experience but also contribute to a more responsible approach to travel, aligning with the growing global trend towards sustainable tourism. Overall, the Travel Package Website is on track to set a new standard in the industry, offering a holistic solution for travelers who value both convenience and environmental stewardship.

Chapter 8

SCOPE FOR FUTURE ENHANCEMENT

The scope for future enhancements in a travel package website can be extensive, offering opportunities to improve user experience, expand functionality, and stay competitive in the market. Here are some potential areas for enhancement:

- Personalized Recommendations: Implementing machine learning algorithms to analyze user behavior and preferences can enable the website to provide personalized travel recommendations. This can include suggesting destinations, accommodations, activities, and packages based on past interactions and user profiles.
- Advanced Search and Filtering: Enhance the search functionality by incorporating advanced filters such as budget range, travel dates, preferred activities, and location preferences. This can help users quickly find relevant travel packages tailored to their specific needs.
- Interactive Itinerary Planning: Introduce interactive itinerary planning tools that allow users to customize their travel plans. This can include drag-and-drop functionality to arrange activities, add or remove destinations, and visualize their itinerary in a user-friendly interface.
- Mobile App Integration: Develop a dedicated mobile app that complements the website, offering users a seamless experience across devices. The app can provide features like real-time notifications, offline access to travel information, and easy booking and payment options.
- Integration with Social Platforms: Enable social media integration to allow users to share their travel plans, experiences, and reviews directly from the website. This can also include social login options for a more streamlined user registration process.

- Multilingual Support: Expand the website's reach by offering multilingual support, allowing users from different regions to access and navigate the platform in their preferred language. This can enhance user engagement and attract a diverse audience.
- Virtual Reality (VR) Tours: Incorporate VR technology to offer virtual tours of destinations, accommodations, and attractions. This immersive experience can help users visualize their travel options better and make more informed decisions.
- Customer Support Enhancements: Implement AI-powered chatbots or virtual assistants to provide instant customer support, answer queries, and assist users throughout their journey, from initial booking to post-trip feedback.
- Integration with Travel APIs: Collaborate with third-party travel APIs to expand the range of services offered on the website, such as flight bookings, car rentals, travel insurance, and local experiences. This can provide users with a comprehensive travel planning platform.
- Data Analytics and Insights: Utilize data analytics tools to gather insights into user behavior, market trends, popular destinations, and booking patterns. Leveraging this data can help in refining marketing strategies, improving website performance, and enhancing the overall user experience.

APPENDICES

Appendix A

SCRUM PROCESS ARTIFACTS

A.1 Product Backlog

Priority	Product backlog items	User story	Estimate (Hours)
1	Database creation	As an operations engineer, I want to be able to store all customer information.	240
2	Login page	As a site member I want to login to the site.	160
3	Category page	As a site member, I want to be able to look for different categories of packages.	400
4	Booking process	As a site member, I want to be able to make bookings.	240
5	Package Management	As a Agent, I want to be able to able to add,view and delete packages.	80

A.1.1 Functional Requirements

The Fuctional Requirements include :

- User Registration and Authentication: Users can register with the website. Authentication during login using JWT tokens.
- Product Management: Authorized agents can add travel packages/products. Details include name, image, category, prices, and descriptions.
- Booking Functionality: Users can book travel packages. Booking details include user ID, user name, product name, quantity, and agent details.

- Approval System: Admin can approve/reject products and user registrations.
- User Profile Management: Users can view and manage their profile details. Access to uploaded products (for agents) and booked items.
- Image Upload and Storage: Agents can upload product images securely. System stores images and provides endpoints for retrieval.
- User Cart Management: Users can add/remove products from their carts. View cart contents.
- Product Listing: Endpoints to list available products, new collections, and popular products.
- Role-Based Access Control (RBAC): Different functionalities restricted based on user roles (admin, agent, user).

A.1.2 Non Functional Requirements

The Non Fuctional Requirements include :

- Performance: The system should be responsive and handle multiple concurrent user requests efficiently, ensuring low latency and optimal performance.
- Security: The system must implement secure authentication and authorization mechanisms, protect sensitive data (e.g., passwords, JWT tokens), and prevent common security vulnerabilities such as SQL injection and cross-site scripting (XSS).
- Scalability: The system should be designed to scale horizontally or vertically to accommodate increasing user traffic and data volume without compromising performance.
- Reliability: The system should be highly available and reliable, with minimal downtime and robust error handling to prevent data loss or corruption.
- Usability: The user interface should be intuitive, user-friendly, and accessible across different devices and screen sizes, enhancing overall user experience.
- Compatibility: The system should be compatible with modern web browsers and devices, ensuring consistent functionality and appearance.
- Data Integrity: The system must maintain data integrity by using appropriate data validation techniques, ensuring accurate and consistent data storage and retrieval.

A.1.3 System Configuration

System configuration (SC) in systems engineering encompasses the computers, processes, and devices that form the system and its boundaries. It specifically outlines and describes the components that define or specify what constitutes a system. Alternatively, the term system configuration can refer to a model. A well-configured system helps prevent resource conflicts and simplifies future upgrades with new equipment. Conversely, an inadequately configured system can result in unusual errors and challenges during upgrades.

A.1.3.1 Hardware Configuration

- Processor : Intel Core i5
- CPU Speed : 2.60GHz
- Hard Disk : 40GB to 80GB
- Memory : 8GB RAM
- Display : Intel HD Graphics 620
- Monitor : LED Monitor

A.1.3.2 Software Configuration

Software configuration is the setup and customization of software applications and operating systems within a computer system. It involves installation, customization, and maintenance to ensure optimal performance and compatibility.Here's a breakdown of the software components :

- Operating System: Windows 7 and above
- Web Server: Express.js
- Language: JavaScript
- Front End: React.js
- Back End: Node.js
- Database: MongoDB
- IDE: Visual Studio Code
- Framework: MERN (MongoDB, Express.js, React.js, Node.js)

A.1.4 Conceptual Models

Conceptual models serve as simplified representations of complex systems, phenomena, or ideas, aiming to enhance understanding and communication. By abstracting away intricate details, these models highlight fundamental relationships, structures, or processes, making them more accessible for analysis and interpretation. They find applications across various disciplines, including science, business, information science, psychology, and economics. Whether depicting scientific theories, business strategies, database structures, cognitive processes, or economic principles, conceptual models provide valuable frameworks for organizing knowledge, making predictions, and facilitating communication among researchers, practitioners, and educators.

A.1.4.1 ER Diagram/Class Diagram

An Entity-Relationship (ER) diagram is a graphical representation of the entities, relationships, attributes, and constraints within a database system. Entities are objects or concepts that are distinguishable, such as customers, products, or orders. Relationships depict how entities are related to each other, such as a customer placing an order. Attributes are properties or characteristics of entities, like a customer's name or an order's date. Constraints define rules or conditions that must be satisfied within the database, such as a customer being required to have a unique identifier. ER diagrams use symbols like rectangles for entities, diamonds for relationships, and ovals for attributes, with lines indicating connections between them. They provide a clear visualization of the database structure, aiding in database design, communication, and understanding of data dependencies.

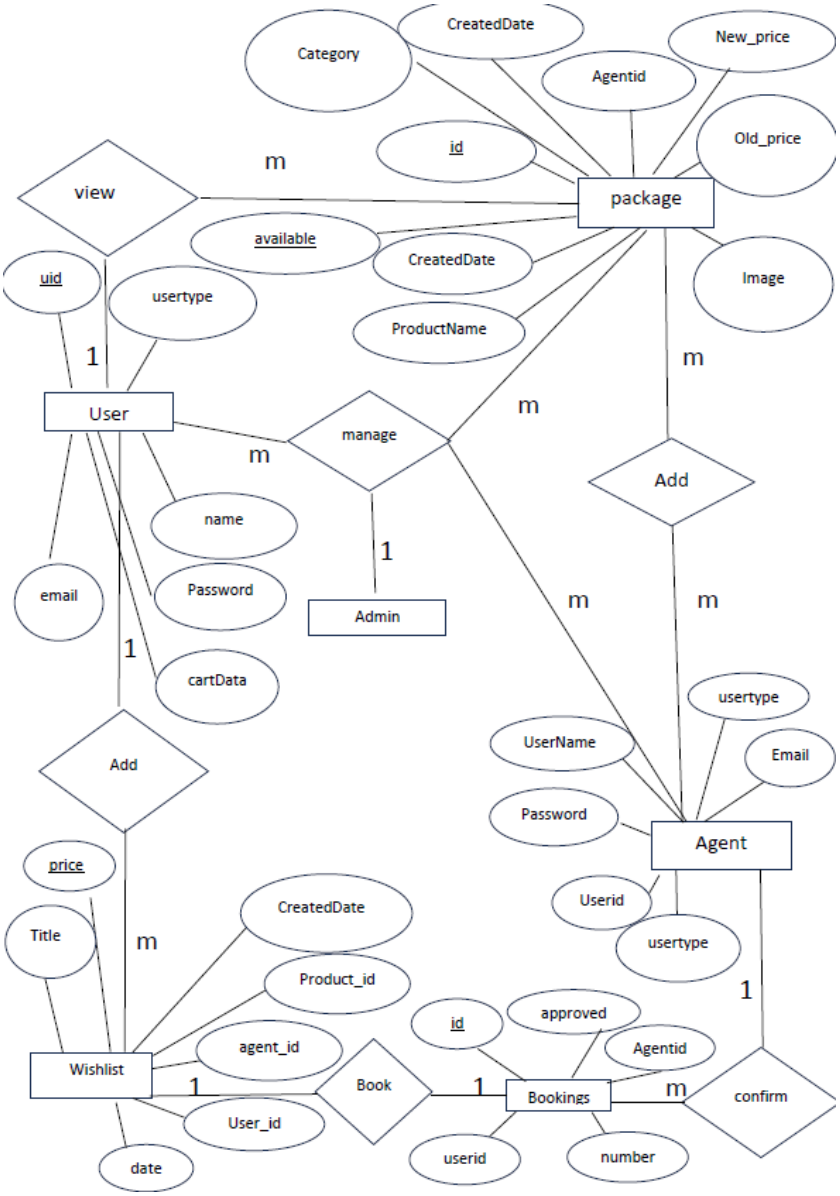


Figure A.1: ER diagram

A.1.4.2 Use Case/UML Diagrams

Use case diagrams play a crucial role in conducting high-level analysis of system requirements. They capture system functionalities in an organized manner, helping to understand the interactions between actors and the system itself. Actors can encompass human users, internal applications, or external systems that interact with the system being analyzed. When creating a use case diagram, it's essential to identify the following components:

- Functionalities represented as use cases

- Actors involved in the system
- Relationships among use cases and actors

These diagrams are pivotal in comprehending system dynamics, focusing on event flows and system behavior without delving into implementation details. They function as black boxes, revealing inputs, outputs, and the overall function without revealing internal workings.

While use case diagrams are primarily used for high-level design, they also include additional elements such as pre-conditions, post-conditions, and exceptions. These elements are instrumental in creating test cases for system testing purposes.

Although use case diagrams are not typically used for forward or reverse engineering, they can still be adapted for these processes. In forward engineering, use case diagrams aid in developing test cases, while in reverse engineering, they assist in extracting requirement details from existing applications. This versatility makes use case diagrams a valuable tool throughout the software development lifecycle.

A.2 Sprint Details

Task Name	Description	Priority	Status	Durations (Days)
Sprint 1	Public Pages	High	Completed	6
Task 1	Login page	High	Completed	3
Task 2	category page	Medium	Completed	1
Task 3	cart page	Medium	Completed	1
Task 4	profile page	Medium	Completed	1
Sprint 2	Admin page	High	Completed	3
Task 1	user List	low	Complete	1
Task 2	Product List	low	Completed	1
Task 3	Approve users	low	Completed	1
Sprint 3	Agent page	High	Completed	7
Task 1	Add packages	High	Completed	4
Task 2	View package	low	Completed	1
Task 3	Booking requests	Medium	Completed	2

A.3 Details of daily scrum meeting

Daily scrum meetings were held on each day of a sprint. The meetings were held in the office premises of the team at 9:30 am. These scrum meetings were strictly time-boxed to 15 minutes. This was to keep the discussion brisk but relevant.

All team members were required to attend scrum meetings. Since both the Scrum Master and product owner are committed team members, they are expected to attend and participate. Anyone else was allowed to attend, but was there only to listen. This made scrum meetings an excellent way for a Scrum team to disseminate information.

The daily scrum meeting was not used as a problem-solving or issue resolution meeting. Issues that were raised were taken off-line and usually dealt with by the relevant subgroup immediately after the meeting. During the daily scrum, each team member answered the following three questions:

- What did you do yesterday?
- What will you do today?
- Are there any impediments in your way?

By focusing on what each person accomplished on a day and would accomplish the next day, the team would gain an excellent understanding of what work had been done and what work remained. The daily scrum meeting was not a status update meeting in which a boss was collecting information about who was behind schedule. Rather, it was a meeting in which team members made commitments to each other.

A.4 Details of bi-weekly meeting

Bi-weekly Meeting No.	**Date**	**Details**
Bi-weekly Meeting 1	30/01/2024	Topic Submission
Bi-weekly Meeting 2	06/02/2024	Topic Approved
Bi-weekly Meeting 3	15/02/2024	First Review of the project
Bi-weekly Meeting 4	15/03/2024	Mid-term review of the project
Bi-weekly Meeting 5	09/04/2024	Submission of Draft Report
Bi-weekly Meeting 6	17/04/2024	Final Presentation with working model
Bi-weekly Meeting 7	22/04/2024	Final Report Submission

Appendix B

VERSIONING

Sprint Number	Current Version Number
Sprint 1	V1.0
Sprint 2	V1.1
Sprint 3	V2.0
Sprint 4	V2.1
Sprint 5	V2.2

Appendix C

DATA FLOW DIAGRAMS

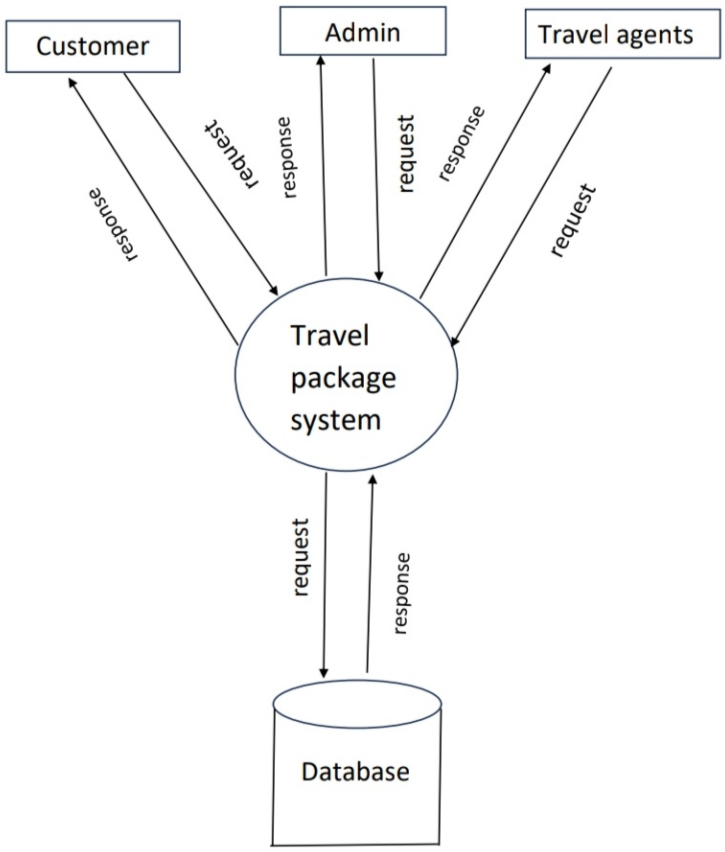


Figure C.1: level 0 dfd

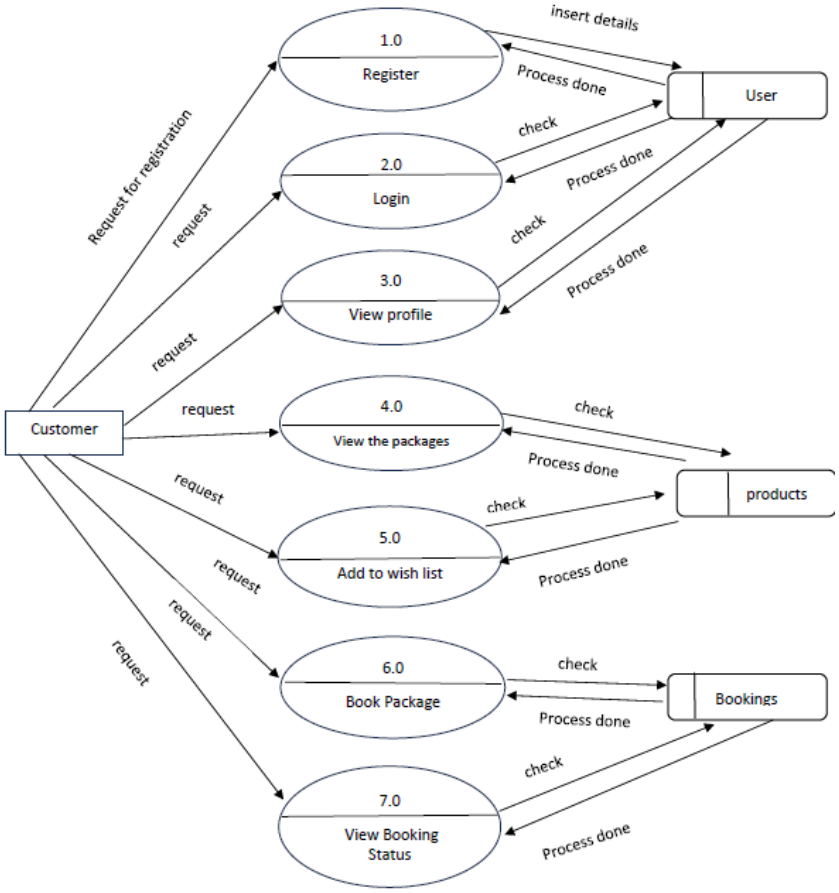


Figure C.2: level 1 user

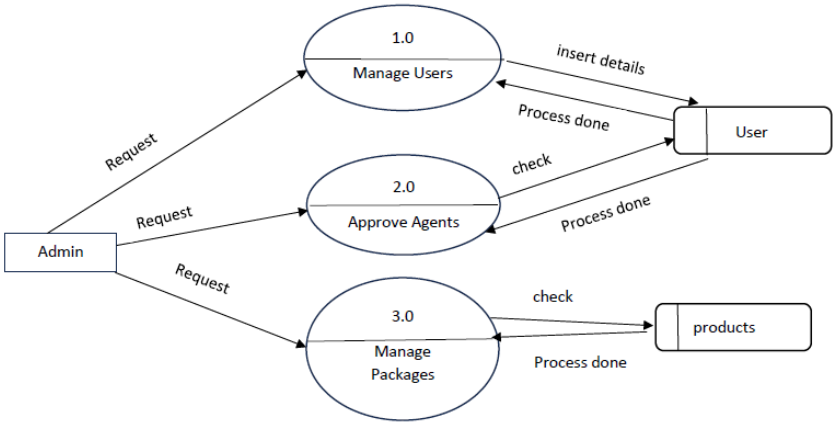


Figure C.3: level 1 admin

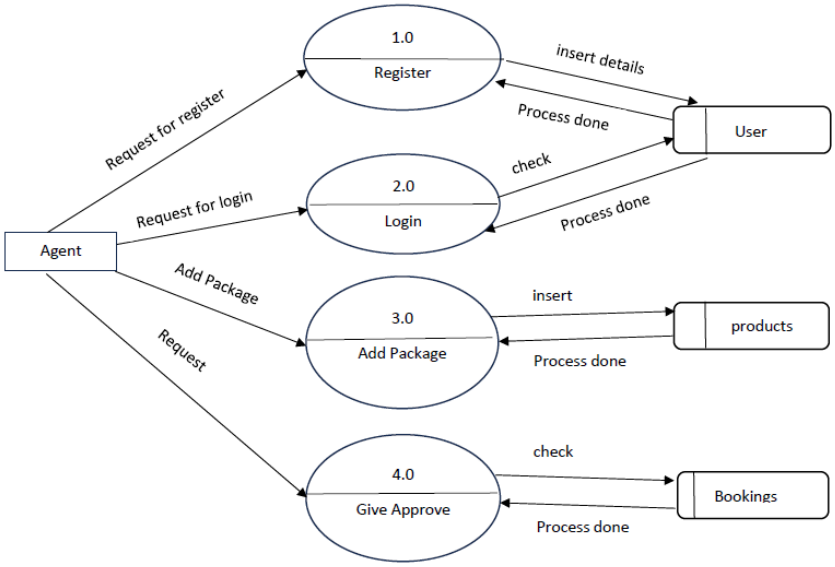


Figure C.4: level 1 agent

Appendix D

TABLE STRUCTURE

Column Name	Data Type	Constraints	Description
_id	ObjectId	Primary Key	Unique identifier for the product
id	Integer	Not Null	The ID of the product
name	String	Not Null	The name of the product
image	String	Not Null	URL of the image related to the product
category	String	Not Null	The category of the product
new_price	Integer	Not Null	The new price of the product
old_price	Integer	Not Null	The old price of the product
available	Boolean	Not Null	Indicates whether the product is available or not
agentId	String	Not Null	The ID of the agent responsible for the product
agentName	String	Not Null	The name of the agent responsible for the product
date	Date	Not Null	The date of the product

Table D.1: products

Column Name	Data Type	Constraints	Description
_id	ObjectId	Primary Key	Unique identifier for the product
uid	Integer	Not Null	The ID of the user
name	String	Not Null	The name of the user
email	String	Not Null	Email of the user
password	String	Not Null	Password of the user
cartData	Object	Not Null	Data related to the user's cart
uploadData	Object	Not Null	Data related to user uploads
usertype	String	Not Null	Type of user (e.g., agent)
approved	Boolean	Not Null	Indicates whether user is approved or not
date	Date	Not Null	Date associated with the user

Table D.2: Users

Column Name	Data Type	Description
_id	ObjectId	Unique identifier for the entry
userId	String	ID of the user associated with the entry
userName	String	Name of the user
productName	String	Name of the product
Number	Integer	Quantity of the product
agentId	String	ID of the agent associated with the entry
approved	Boolean	Indicates approval status
created_at	Date	Date and time of creation

Table D.3: Booking

Appendix E

SAMPLE SCREENSHOTS

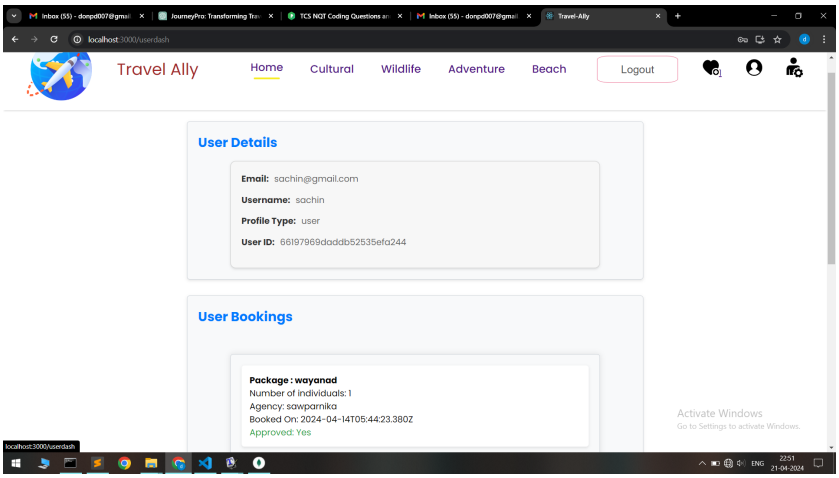


Figure E.1: Screenshot of user profile

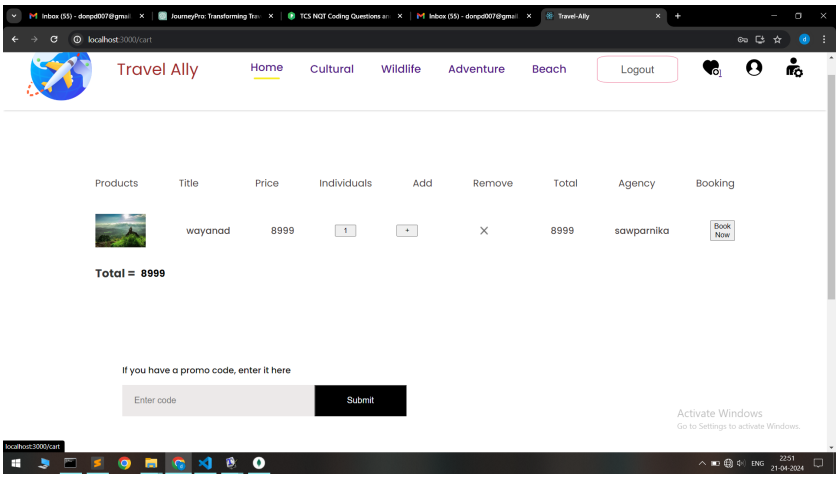


Figure E.2: Screenshot of whishlist

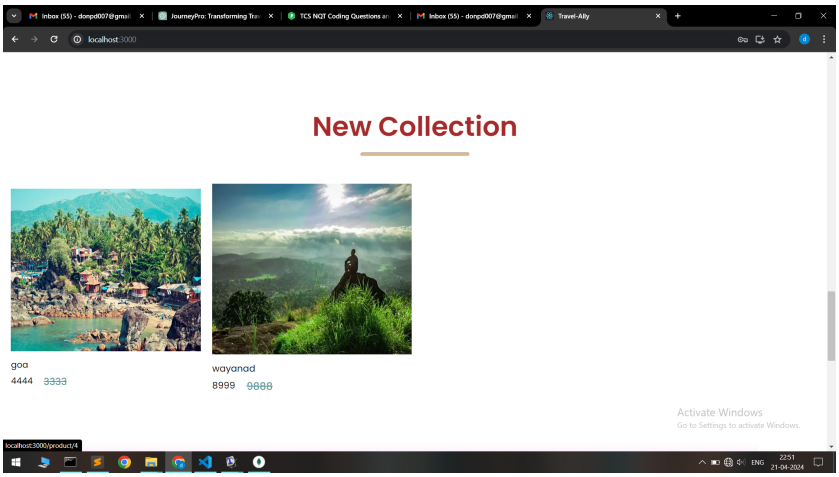


Figure E.3: Screenshot of new collection category

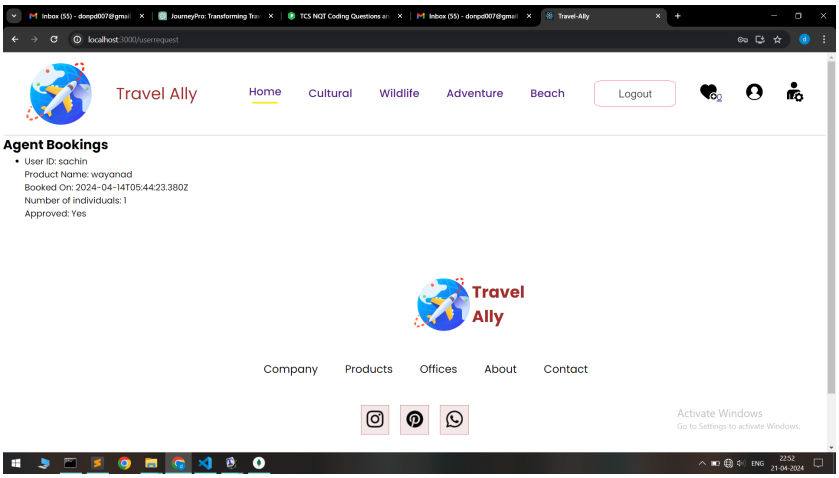


Figure E.4: Screenshot of agent view on request of booking

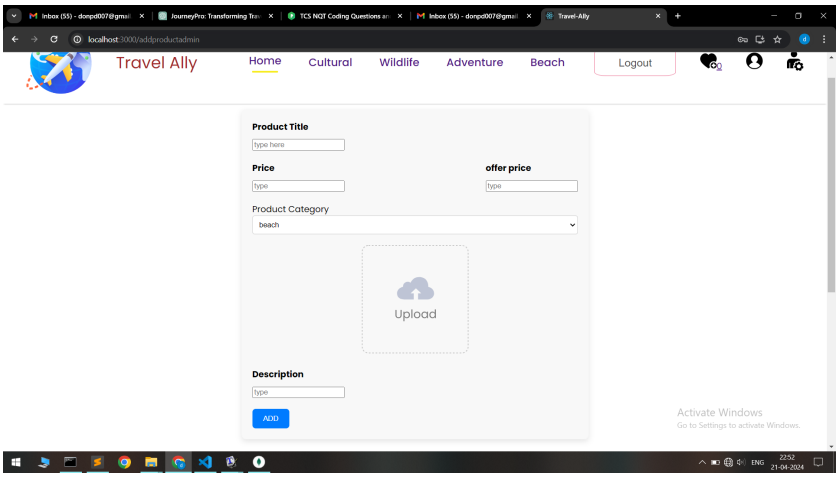


Figure E.5: Screenshot of Add product functions

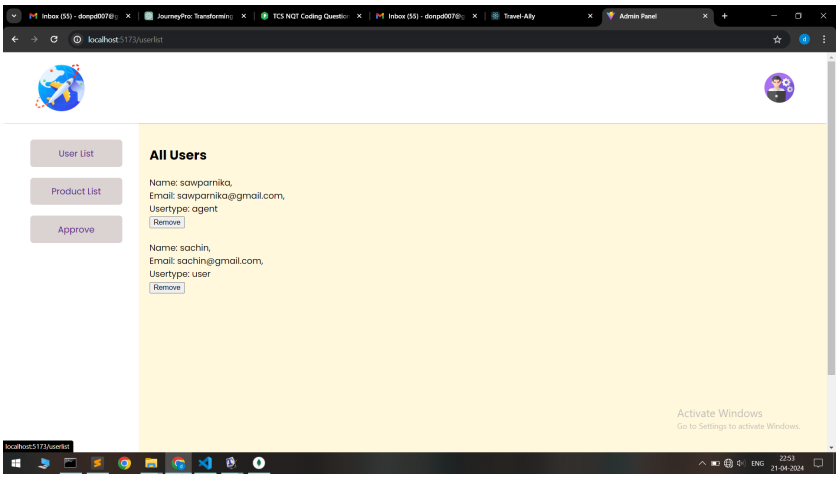


Figure E.6: Screenshot of all users list

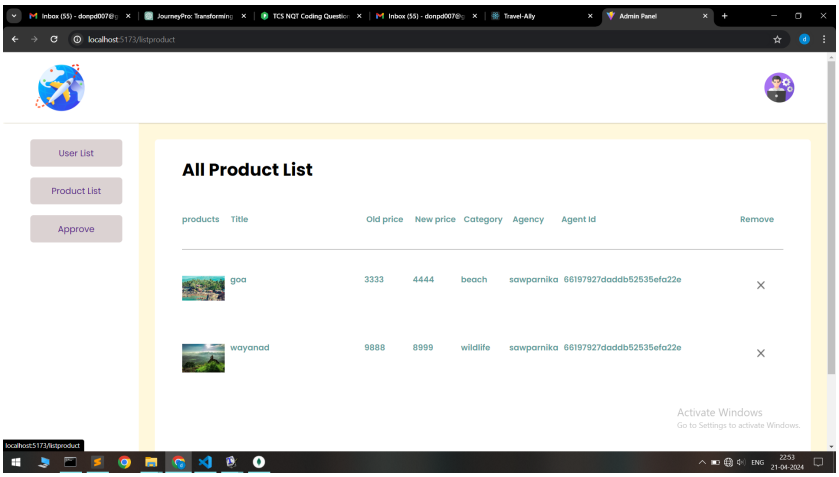


Figure E.7: Screenshot of all product

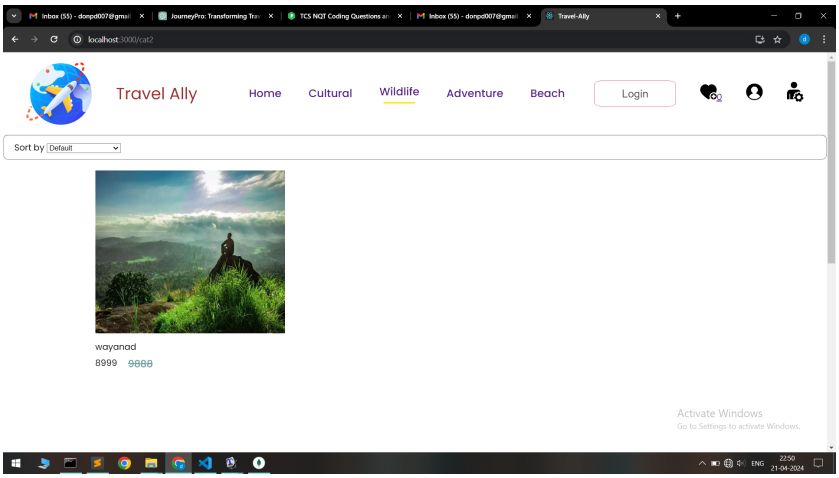


Figure E.8: Screenshot of category

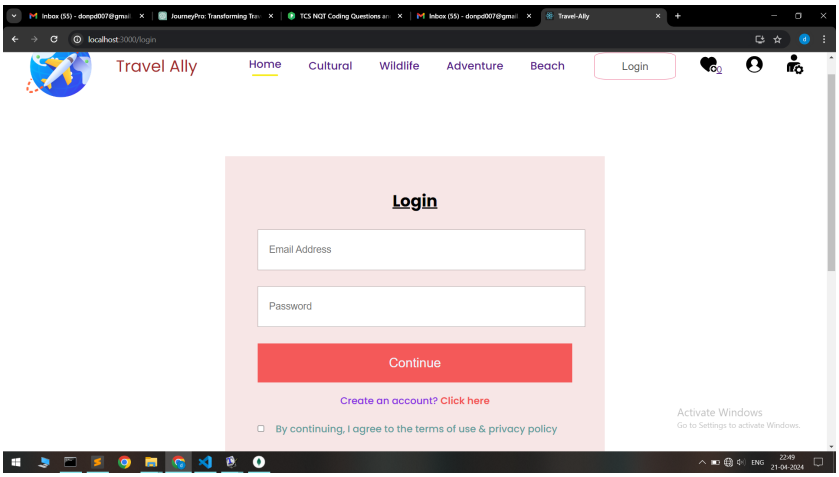


Figure E.9: Screenshot of login signup

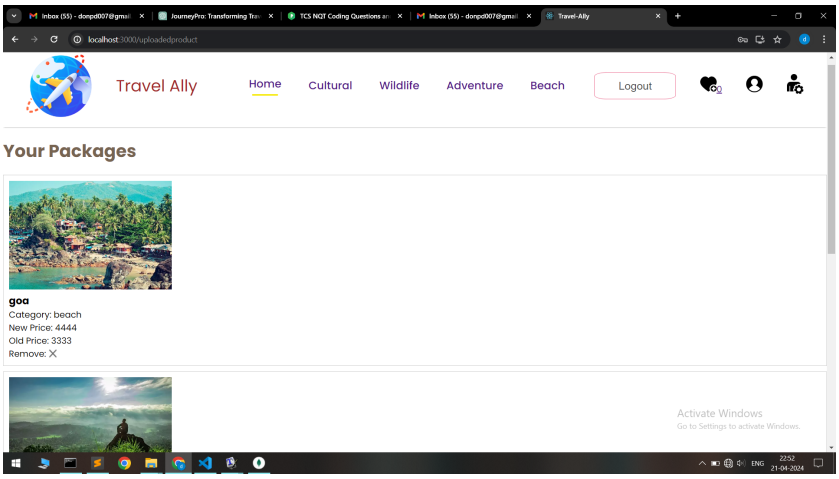
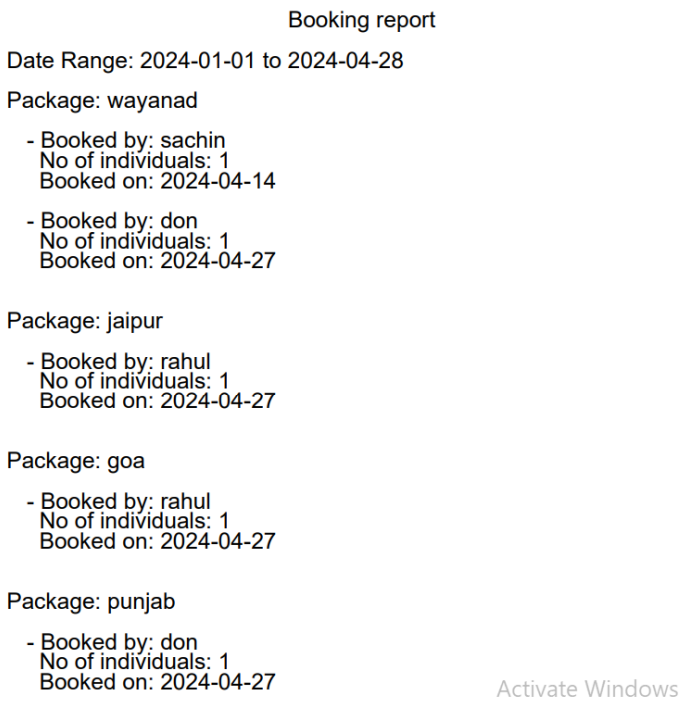


Figure E.10: Screenshot of package list of agent

Appendix F

REPORTS


Booking report

Date Range: 2024-01-01 to 2024-04-28

Package: wayanad

- Booked by: sachin
  No of individuals: 1
  Booked on: 2024-04-14

- Booked by: don
  No of individuals: 1
  Booked on: 2024-04-27

Package: jaipur

- Booked by: rahul
  No of individuals: 1
  Booked on: 2024-04-27

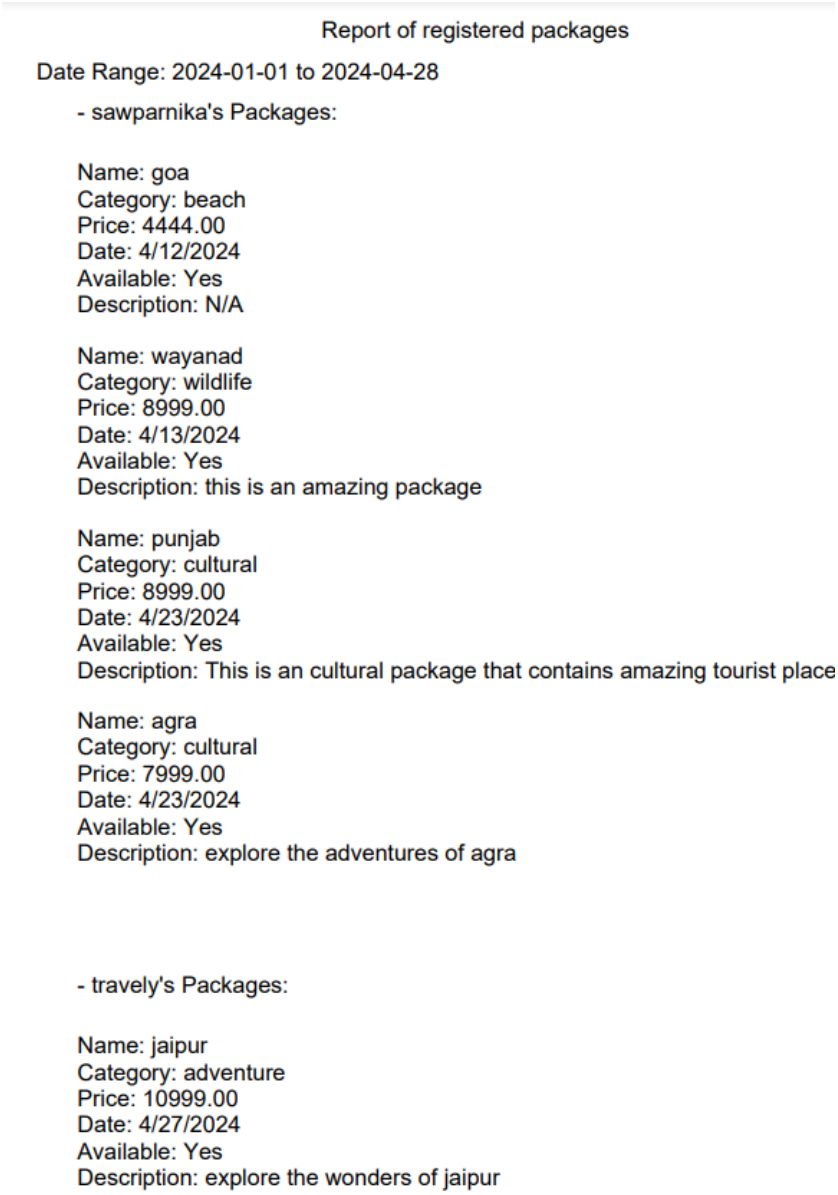
Package: goa

- Booked by: rahul
  No of individuals: 1
  Booked on: 2024-04-27

Package: punjab

- Booked by: don
  No of individuals: 1
  Booked on: 2024-04-27

Figure F.1: report of Bookings made at a date range



Report of registered packages

Date Range: 2024-01-01 to 2024-04-28

- sawparnika's Packages:

Name: goa
Category: beach
Price: 4444.00
Date: 4/12/2024
Available: Yes
Description: N/A

Name: wayanad
Category: wildlife
Price: 8999.00
Date: 4/13/2024
Available: Yes
Description: this is an amazing package

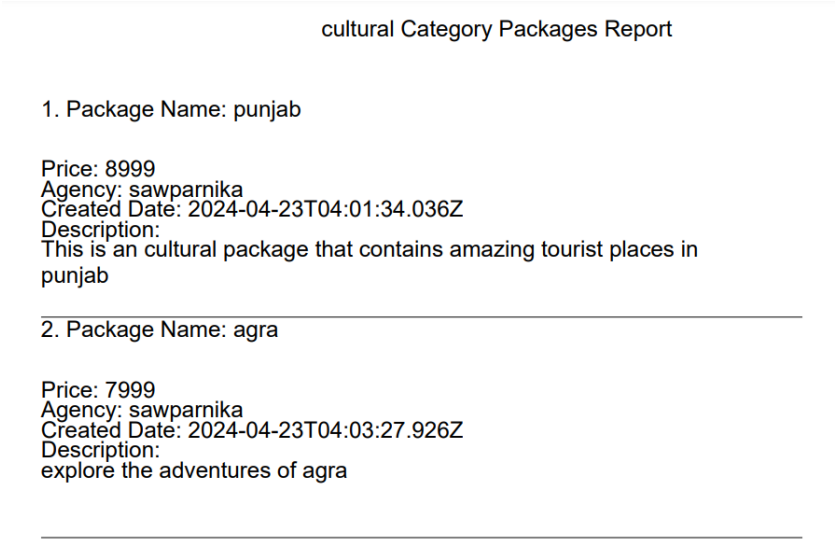
Name: punjab
Category: cultural
Price: 8999.00
Date: 4/23/2024
Available: Yes
Description: This is an cultural package that contains amazing tourist place

Name: agra
Category: cultural
Price: 7999.00
Date: 4/23/2024
Available: Yes
Description: explore the adventures of agra

- travely's Packages:

Name: jaipur
Category: adventure
Price: 10999.00
Date: 4/27/2024
Available: Yes
Description: explore the wonders of jaipur

Figure F.2: report of packages created at a date range


cultural Category Packages Report

1. Package Name: punjab

Price: 8999
Agency: sawparnika
Created Date: 2024-04-23T04:01:34.036Z
Description:
This is an cultural package that contains amazing tourist places in punjab

2. Package Name: agra

Price: 7999
Agency: sawparnika
Created Date: 2024-04-23T04:03:27.926Z
Description:
explore the adventures of agra

Figure F.3: report of packages added in particular category

User Signup report

Date Range: 2024-01-01 to 2024-04-28

Users Registered:

Name: sachin
Email: sachin@gmail.com
UID: 3
User Type: user
Date: 2024-04-12T18:11:53.255Z

Name: don
Email: donthayyur123@gmail.com
UID: 5
User Type: user
Date: 2024-04-23T04:56:04.575Z

Name: rahul
Email: rahul@gmail.com
UID: 6
User Type: user
Date: 2024-04-23T05:14:28.424Z

Agents registered :

Name: sawparnika
Email: sawparnika@gmail.com
UID: 1
User Type: agent
Date: 2024-04-12T18:10:47.038Z

Name: travely
Email: travely@gmail.com
UID: 4
User Type: agent
Date: 2024-04-23T04:11:25.878Z

Figure F.4: report of users registered in date range

Appendix G

sample code

G.1 ShopContext.jsx

```
import React, { createContext, useEffect, useState } from "react";


export const ShopContext = createContext(null);

const getDefaultCart = ()=> {
    let cart = {};
    for (let index = 0; index < 300+1; index++){
        cart[index] = 0;
    }
    return cart;
 }


const ShopContextProvider = (props) => {

    const [all_product,setAll_Product] = useState([]);
    const [cartItems,setCartItems] = useState(getDefaultCart());

    useEffect(()=>{
        fetch('http://localhost:4000/allproducts')
        .then((response)=>response.json())
        .then((data)=>setAll_Product(data))
```

```
    if(localStorage.getItem('auth-token')){
        fetch('http://localhost:4000/getcart',{
        method:'POST',
        headers:{
            Accept:'application/form-data',
            'auth-token':`${localStorage.getItem('auth-token')}`,
            'Content-Type':'application/json',
        },
        body:"",
    }).then((response)=>response.json())
    .then((data)=>setCartItems(data));

}

},[])

const addToCart = (itemId) =>{
    setCartItems((prev)=>({...prev,[itemId]:prev[itemId]+1}));
    if(localStorage.getItem('auth-token')){
        fetch('http://localhost:4000/addtocart',{
            method:'POST',
            headers:{
                Accept:'application/form-data',
                'auth-token':`${localStorage.getItem('auth-token')}`,
                'Content-Type': 'application/json',
            },
            body:JSON.stringify({"itemId":itemId}),
        })
            .then((response)=>response.json())
            .then((data)=>console.log(data));
        }

}
const removeFromCart = (itemId) =>{
    setCartItems((prev)=>({...prev,[itemId]:prev[itemId]-1}));
```

```
        if(localStorage.getItem('auth-token')){
            fetch('http://localhost:4000/removefromcart',{
                method:'POST',
                headers:{
                    Accept:'application/form-data',
                    'auth-token':`${localStorage.getItem('auth-token')}`,
                    'Content-Type': 'application/json',
                },
                body:JSON.stringify({"itemId":itemId}),
            })
                .then((response)=>response.json())
                .then((data)=>console.log(data));
            }

    }

    const getTotalCartAmount = () => {
    let totalAmount = 0;
    for(const item in cartItems)
    {
        if(cartItems[item]>0)
        {
            let itemInfo = all_product.find((product)=>
            product.id===Number(item))
            totalAmount += itemInfo.new_price * cartItems[item];
        }


    }return totalAmount;
}

    const getTotalCartItems = () => {
        let totalItem=0;
        for(const item in cartItems){
            if(cartItems[item]>0)
            {
```

```
                totalItem += cartItems[item];
            }
        }
        return totalItem;
    }

    const contextValue = {getTotalCartItems,getTotalCartAmount,
    all_product,cartItems,addToCart,removeFromCart};
    return (
        <ShopContext.Provider value={contextValue}>
            {props.children}
        </ShopContext.Provider>
    )
}

export default ShopContextProvider;
```

G.2 Approve.jsx

```
import React, { useState, useEffect } from 'react';

const AdminApproval = () => {
    const [users, setUsers] = useState([]);

    useEffect(() => {
        fetchUsersPendingApproval();
    }, []);



    const fetchUsersPendingApproval = async () => {
        try {
            const response = await fetch('http://
            localhost:4000/admin/users/pending');
            const data = await response.json();
```

```
            if (data.success) {
                setUsers(data.users);
            } else {
                console.error('Error fetching users:', data.error);
            }
        } catch (error) {
            console.error('Error fetching users:', error);
        }
    };


    const handleApproveUser = async (userId) => {
        try {
            const response = await fetch(`http://
            localhost:4000/admin/approve/${userId}`, {
                method: 'PATCH',
            });
            const data = await response.json();

            if (data.success) {
                // Remove the approved user from the list
                setUsers(users.filter(user => user._id !== userId));
            } else {
                console.error('Error approving user:', data.error);
            }
        } catch (error) {
            console.error('Error approving user:', error);
        }
    };

    return (
        <div className="admin-approval">
            <h2>Users Pending Approval</h2>
            <ul>
                {users.map(user => (
                    <li key={user._id}>
```

```
                        {user.name} - {user.email}
                        <button onClick={() => handleApproveUser(user._id)}
                        >Approve</button>
                    </li>
                ))}
            </ul>
        </div>
    );
};

export default AdminApproval;

end{verbatim}

\section{Listproduct.jsx}
\begin{verbatim}
import React, { useEffect, useState } from 'react'
import './ListProduct.css'
import cross_icon from '../../assets/cross_icon.png'
import { Link } from 'react-router-dom'

export const ListProduct = () => {
  const [allproducts,setAllProducts] = useState([]);

  const fetchInfo = async ()=>{
    await fetch('http://localhost:4000/allproducts')
    .then((res)=>res.json())
    .then((data)=>{setAllProducts(data)});
  }

  useEffect(()=>{
    fetchInfo();
  },[])

  const remove_product = async (id)=>{
    await fetch('http://localhost:4000/removeproduct',{
```

```
      method:'POST',
      headers:{
        Accept:'application/json',
        'Content-Type':'application/json',
      },
      body:JSON.stringify({id:id})

    })
    await fetchInfo();
  }

  return (

    <div className='list-product'>

      <h1>All Product List</h1>
    <div className="listproduct-format-main">
      <p> products</p>
      <p>Title</p>
      <p>Old price</p>
      <p>New price</p>
      <p>Category</p>
      <p>Agency</p>
      <p>Agent Id</p>
      <p>Remove</p>
    </div>
    <div className="listproduct-allproducts">
      <hr />
    {allproducts.map((product,index)=>{
      return <div key={index} className='listproduct-format
      -main listproduct-format'>
        <img src={product.image} alt=''
        className='listproduct-format-main'></img>
        <p>{product.name}</p>
        <p>{product.old_price}</p>
        <p>{product.new_price}</p>
```

```
        <p>{product.category}</p>
        <p>{product.agentName}</p>
        <p>{product.agentId}</p>
        <img onClick={()=>{remove_product(product.id)}}
         className='listproduct-remove-icon' src={cross_icon} alt="" />
        </div>
    })}
    </div>
    </div>
  )
}

export default ListProduct

end{verbatim}

\section{Listproduct.css}
\begin{verbatim}
.list-product{
    display:flexbox;
    align-items: center;
    width: 100%;
    height: auto;
    padding: 10px 50px;
    margin: 30px;
    border-radius: 6px;
    background: white;
}
.listproduct-format-main{
    display: grid;
    grid-template-columns: 1fr 3fr 1fr 1fr 1fr 1fr 4fr 1fr;
    gap: 10px;
    width: 100%;
    padding: 20px 0px;
    color: cadetblue;
    font-size: 15px;
```

```
    font-weight: 600;

}
.listproduct-remove-icon{
  cursor: pointer;
  margin:30px;

}

end{verbatim}

\section{Hero.jsx}
\begin{verbatim}

import React from 'react'
import './Hero.css'
import plane_icon from '../Aaassets/plane.png'
import arrow_icon from '../Assets/arrow.png'
import hero_image from  '../Aaassets/model.png'


export const Hero = () => {
  return (
    <div className='hero'>
    <div className='hero-left'>
        <h2>choose your destination</h2>
        <div>
            <div className='hero-hand-icon'>
                <p>Exciting</p>
                <img src={plane_icon} alt=''/>
            </div>
            <p>trip packages</p>

        </div>
        <div className='hero-latest-btn'>
            <div>Latest</div>
```

```
            <img src={arrow_icon} alt=''/>
        </div>
    </div>
    <div className='hero-right'>
        <img src={hero_image} alt="" />
    </div>
    </div>
  )
}

end{verbatim}

\section{Hero.css}
\begin{verbatim}
.hero{
    height: 100vh;
    background: linear-gradient(180deg, white, rgb(117, 102, 102) 60%);
    display: flex;
}
.hero-left{
    flex: 1;
    display: flex;
    flex-direction: column;
    justify-content: center;
    gap: 20px;
    padding-left: 180;
    line-height: 1.1;

}
.hero-left h2{
    color: rgb(212, 97, 97);
    font-size: 26px;
    font-weight: 600;
}

.hero-left p{
```

```
    color: black;
    font-size: 100px;
    font-weight: 700;
}

.hero-hand-icon{
    display: flex;
    align-items: center;
    gap: 20px;
}
.hero-hand-icon img{
    width: 105px;
}
.hero-latest-btn{
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 15px;
    width: 310px;
    height: 70px;
    margin-top: 30px;
    border-radius:75px;
    background: red;
    color: aliceblue;
    font-size: 22px;
    font-weight: 500px;
}
.hero-right{
    flex: 1;
    display: flex;
    align-items: center;


}

end{verbatim}
```

```
\section{new_collection.jsx}
\begin{verbatim}
import React, { useEffect, useState } from 'react'
import './NewCollection.css'
import { Item } from '../Item/Item'


export const NewCollection = () => {

  const [new_collection,setNew_collection] = useState([]);

  useEffect(()=>{
  fetch('http://localhost:4000/newcollections')
  .then((response)=>response.json())
  .then((data)=>setNew_collection(data));
  },[])
  return (
    <div className='new-collections'>
        <h1>New Collection</h1>
        <hr/>
        <div className='collection'>
            {new_collection.map((item,i)=>{
                 return <Item key={i} id={item.id} name={item.name} image={it
            })}
        </div>
    </div>
  )
}


end{verbatim}
\section{newcollection.css}
\begin{verbatim}
.new-collections{
    display: flex;
```

```
    flex-direction: column;
    align-items: center;
    gap: 10px;
   margin-bottom: 100px;

}
.new-collections h1{
    color: brown;
    font-size: 50px;
    font-weight: 600;
}
.new-collections hr{
    width: 200px;
    height: 6px;
    border-radius: 10px;
    background: burlywood;
}
.collection{
    margin-top: 50px;
    display: grid;
    gap: 30px;
    grid-template-columns: 1fr 1fr 1fr 1fr;
}

end{verbatim}

\section{Navbar.jsx}
\begin{verbatim}
import React, { useContext, useState, useEffect } from 'react';
import './Navbar.css';
import logo from '../Aaassets/airplane.png';
import cart_icon from '../Aaassets/wishlist.png';
import bookin from '../Assets/dp.png';
import admin from '../Assets/admin.png'
import { Link } from 'react-router-dom';
import { ShopContext } from '../../Context/ShopContext';
```

```
export const Navbar = () => {
  const [menu, setMenu] = useState('home');
  const { getTotalCartItems } = useContext(ShopContext);
  const [Usertype, setUsertype] = useState('');

  useEffect(() => {
    const userType = localStorage.getItem('Usertype');
    setUsertype(userType);
    console.log(Usertype);
  }, []);



  return (
    <div className='navbar'>
      <div className='navlogo'>
        <img src={logo} alt='' />
      </div>
      <div className='navhead'>
        <p>Travel Ally</p>
      </div>

      <ul className='nav-menu'>
        <li onClick={() => { setMenu('home') }}>
          <Link style={{ textDecoration: 'none' }} to='/'>Home</Link>
          {menu === 'home' ? <hr /> : <></>}
        </li>
        <li onClick={() => { setMenu('cultural') }}>
          <Link style={{ textDecoration: 'none' }}
          to='/cat1'>Cultural</Link>
          {menu === 'cultural' ? <hr /> : <></>}
        </li>
        <li onClick={() => { setMenu('wildlife') }}>
          <Link style={{ textDecoration: 'none' }}
           to='/cat2'>Wildlife</Link>
```

```
          {menu === 'wildlife' ? <hr /> : <></>}
        </li>
        <li onClick={() => { setMenu('adventure') }}>
          <Link style={{ textDecoration: 'none' }}
           to='/cat3'>Adventure</Link>
          {menu === 'adventure' ? <hr /> : <></>}
        </li>
        <li onClick={() => { setMenu('beach') }}>
          <Link style={{ textDecoration: 'none' }}
           to='/cat4'>Beach</Link>
          {menu === 'beach' ? <hr /> : <></>}
        </li>
      </ul>

      <div className='navlogincart'>
        {localStorage.getItem('auth-token')
          ? <button onClick={() =>
          { localStorage.removeItem('auth-token');
           window.location.replace('/') }}>Logout</button>
          : <Link to='/login'><button>Login</button></Link>}
        <Link to='/cart'><img src={cart_icon} alt="" />
        {getTotalCartItems()}</Link>
        <Link to='/userdash'><img src={bookin} alt="" /></Link>
        <Link to='/check-usertype'><img src={admin} alt="" /></Link>
      </div>
    </div>
  );
};
```

G.3 Navbar.css

```
.navbar{
    display: flex;
    justify-content: space-around;
    padding: 16px;
```

```
    box-shadow: 0 1px 3px -2px black;
}
.navlogo{
    display:flex;
    gap: 10px;
    height: 120px;
    width: 10px;
    margin-right: 10px;

}
.navhead{
    margin-left: 90px;
    display: flex;
    align-items: center;
    color: brown;
    font-size: 30px;
}
.nav-menu{
    display: flex;
    align-items: center;
    list-style: none;
    gap: 50px;
    color: #ae7d7d;
    font-size: 20px;
    font-weight: 500;
}

.nav-menu li{
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    gap: 3px;

}
```

```
.nav-menu hr{
    border: none;
    width: 80%;
    height: 3px;
    border-radius: 10px;
    background: #f7ec21;

}
.navlogincart{
    display: flex;
    align-items: center;
    gap: 45px;
}
.navlogincart img{
    width: 30px;
}
.navlogincart button{
    width: 150px;
    height: 50px;
    outline: none;
    border: 1px solid  #eb688d;
    border-radius: 15%;
    color: #515151;
    font-size: 20px;
    font-weight: 500;
    background: white;
    cursor: pointer;


}

.navlogincart button:active{
    background: lightblue;
}

.nac-cart-count{
```

```
    width: 22px;
    height: 22px;
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: -35px;
    margin-left: -55px;
    border-radius: 11px;
    font-size: 14px;
    background: red;
    color: white;

}
.addpack button {
    color: #fbf6f6;
    background: #c9a9b2;
    width: 150px;
    height: 50px;
    padding: 1px 20px; /* Adjust padding as needed */
    border: none;
    border-radius: 20px; /* Make the button round */
    cursor: pointer;
    outline: none;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1); /* Add a subtle box shadow */
    transition: background-color 0.3s ease, transform 0.2s ease; /* Smooth tr

    /* Optional: Hover effect */
    &:hover {
        background-color: #b9919d; /* Darken the background on hover */
        transform: translateY(-2px); /* Add a slight lift effect */
    }

    /* Optional: Active effect */
    &:active {
        transform: translateY(1px); /* Add a slight press effect */
    }
```

```
}


@media(max-width:1280px){
    .navbar{
padding: 12px 50px;}
}
```

G.4 AddProduct.jsx

```
import React, { useState } from 'react'
import upload_area from '../assets/upload_area.svg'
import './AddProduct.css'

const AddProduct = () => {

const[image,setImage] = useState(false);
const[productDetails,setProductDetails] = useState({
  name:"",
  image:"",
  category:"beach",
  new_price:"",
  old_price:"",
  description:""
})

const imageHandler = (e) =>{
    setImage(e.target.files[0]);
}

const changeHandler = (e) =>{
  setProductDetails({...productDetails,[e.target.name]:e.target.value})
}

const Add_Product = async ()=>{
  console.log(productDetails);
  let responseData;
```

```
  let product = productDetails;

  let formData = new FormData();
  formData.append('product',image);

  await fetch('http://localhost:4000/upload',{
    method:'POST',
    headers:{
      Accept:'application/json',
    },
    body:formData,
  }).then((resp) => resp.json()).then((data)=>{responseData=data});

if (responseData.success) {
    product.image = responseData.image_url;
    console.log(product);
    await fetch('http://localhost:4000/addproductagent', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'auth-token': localStorage.getItem('auth-token')
        },
        body: JSON.stringify(product),
    })
    .then((resp) => resp.json())
    .then((data) => {
        data.success ? alert("Product Added") : alert
        ("Failed to add product");
    })
    .catch((error) => {
        console.error('Error adding product:', error);
        alert("Failed to add product");
    });
  }
}
```

```
return (
  <div className='add-product'>
      <div className="addproduct-itemfield">
      <p>Product Title</p>
      <input value={productDetails.name}
        onChange={changeHandler} type='text' name='name'
        placeholder='type here'/>
      </div>
      <div className="addproduct-price">
        <div className="addproduct-itemfield">
          <p>Price</p>
          <input value={productDetails.old_price}
          onChange={changeHandler} type='text' name='old_price'
           placeholder='type'/>
          </div>
          <div className="addproduct-itemfield">
          <p>offer price</p>
          <input value={productDetails.new_price} onChange=
          {changeHandler} type='text' name='new_price' placeholder='type'/>
          </div>
      </div>
      <div className="addproduct-itemfields">
          <p>Product Category</p>
          <select value={productDetails.category}
           name='category' onChange={changeHandler}
            className='add-product-selector'>
          <option value='beach'>beach</option>
          <option value='adventure'>adventure</option>
          <option value='wildlife'>wildlife</option>
          <option value='cultural'>cultural</option>
          </select>
      </div>
      <div className="addproduct-itemfield">
          <label htmlFor='file-input'>
            <img src={image?URL.createObjectURL(image):
            upload_area} alt="" />
```

```
            </label>
            <input onChange={imageHandler} type='file'
            name='image' id='file-input' hidden/>
        </div>

        <div className="addproduct-itemfield">
        <p>Description</p>
        <input value={productDetails.description}
        onChange={changeHandler} type='text'
        name='description' placeholder='type'/>
        </div>
        <button onClick={()=>{Add_Product()}}
         className='addproduct-button'>ADD</button>
    </div>

  )
}

export default AddProduct
```

G.5 ProductDisplay.jsx

```
import React, { useContext } from 'react'
import './ProductDisplay.css'
import star_icon from '../Assets/star_icon.png'
import star_dull_icon from '../Assets/star_dull_icon.png'
import { ShopContext } from '../../Context/ShopContext'

export const ProductDisplay = (props) => {
    const {product} = props;
    const {addToCart} =useContext(ShopContext);

    return (
    <div className='productdisplay'>
        <div className="productdisplay-left">
            <div className='productdisplay-img-list'>
                <img src={product.image} alt="" />
```

```
            <img src={product.image} alt="" />
            <img src={product.image} alt="" />
            <img src={product.image} alt="" />
        </div>
        <div className="productdisplay-img">
            <img className='productdisplay-main-img'
             src={product.image} alt="" />
        </div>
    </div>
    <div className="productdisplay-right">
        <h1>{product.name}</h1>
        <div className="productdisplay-right-star">
            <img src={star_icon} alt="" />
            <img src={star_icon} alt="" />
            <img src={star_icon} alt="" />
            <img src={star_icon} alt="" />
            <img src={star_dull_icon} alt="" />
            <p>(122)</p>
        </div>
        <div className="productdisplay_right-prices">
            <div className="productdisplay-right-price-old">
                {product.old_price}
            </div>
            <div className="productdispaly-right-price-new">
                {product.new_price}
            </div>
            <div className="product-right-description">
                {product.description}
            </div>
      <div></div>
        </div>
        <button onClick={()=>{addToCart(product.id)}}
        >ADD TO WISHLIST</button>
        <p className='product-dispaly-right-category'>
        Category : <span></span></p>
        <p className='product-dispaly-right-category'>
```

```
            Tags : <span>popular</span></p>
        </div>
    </div>
  )
}
```

G.6 ShopCategory.jsx

```
import React, { useContext, useState } from 'react';
import './Css/ShopCategory.css';
import { ShopContext } from '../Context/ShopContext';
import Item from '../Components/Item/Item';


export const ShopCategory = (props) => {
  const { all_product } = useContext(ShopContext);
  const [sortedProducts, setSortedProducts] = useState(all_product);
  const [sortBy, setSortBy] = useState('default');

  const handleSortChange = (event) => {
    const selectedSortBy = event.target.value;
    let sortedProductsCopy = [...all_product];

    switch (selectedSortBy) {
      case 'nameAsc':
        sortedProductsCopy.sort((a, b) => a.name.localeCompare(b.name));
        break;
      case 'nameDesc':
        sortedProductsCopy.sort((a, b) => b.name.localeCompare(a.name));
        break;
      case 'priceAsc':
        sortedProductsCopy.sort((a, b) => a.new_price - b.new_price);
        break;
      case 'priceDesc':
        sortedProductsCopy.sort((a, b) => b.new_price - a.new_price);
        break;
      default:
```

```
        // No sorting, keep the default order
        break;
    }

    // Update the sorted products state
    setSortedProducts(sortedProductsCopy);
    setSortBy(selectedSortBy);
    // Update the sortBy state with the selected sorting option
  };

  return (
    <div className='shop-category'>
      <div className='shopcategoty-indexsort'>

        <div className='shopcategory-sort'>
          Sort by{' '}
          <select value={sortBy} onChange={handleSortChange}>
            <option value='default'>Default</option>
            <option value='nameAsc'>Name (A-Z)</option>
            <option value='nameDesc'>Name (Z-A)</option>
            <option value='priceAsc'>Price (Low to High)</option>
            <option value='priceDesc'>Price (High to Low)</option>
          </select>
        </div>
      </div>
      <div className='shopcategory-products'>
        {sortedProducts.map((item, i) => {
          if (props.category === item.category) {
            return <Item key={i} id={item.id} name={item.name}
             image={item.image} new_price={item.new_price}
              old_price={item.old_price} />;
          } else {
            return null;
          }
        })}
      </div>
```

```
      <div className="shopcategory-loadmore">
        Load more
      </div>
    </div>
  );
};
```

G.7 ShopCategory.css

```
.sc-banner{
display: block;
margin: 30x auto;
width:  82%;
}
.shopcategory-indexsort{
    display: flex;
    margin: 0px 170px;
    justify-content:space-between;
    align-items: center;
}
.shopcategory-sort{
padding: 10px 20px;
border-radius: 10px;
border: 1px solid grey;

}
.shopcategory-indexsort p span{
    font-weight: 600;
}
.shopcategory-products{
    margin: 20px 170px;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr 1fr;
    row-gap: 80px;
}
.shopcategory-loadmore{
```

```
    display: flex;
    justify-content: center;
    align-items: center;
    margin:150px auto ;
    width: 233px;
    height: 69px;
    border-radius: 75px;
    background: whitesmoke;
    color: rgb(217, 152, 61);
    font-size: 18px;
    font-weight: 500;
}
```

G.8 LoginSignup.jsx

```
import React, { useState } from 'react';
import './Css/LoginSignup.css';



export const LoginSignup = () => {
  const [state, setState] = useState("Login");
  const [formData, setFormData] = useState({
    username: "",
    password: "",
    email: "",
    usertype:""
  });

  const changeHandler = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const login = async () => {
    console.log("Login function", formData);
```

```
  let responseData;
  await fetch('http://localhost:4000/login', {
    method: 'POST',
    headers: {
      Accept: 'application/form-data',
      'content-type': 'application/json',
    },
    body: JSON.stringify(formData),
  }).then((response) => response.json()).then((data) =>
   responseData = data)

  if (responseData.success) {
    localStorage.setItem('auth-token',
     responseData.token);

    console.log("login fetched");
    window.location.replace("/");
  } else {
    alert(responseData.errors);
  }
};




const signup = async () => {
  console.log("signup function", formData);
  let responseData;
  await fetch('http://localhost:4000/signup', {
    method: 'POST',
    headers: {
      Accept: 'application/form-data',
      'content-type': 'application/json',
    },
    body: JSON.stringify(formData),
  }).then((response) => response.json()).then((data)
```

```
    => responseData = data)

  if (responseData.success) {
    localStorage.setItem('auth-token', responseData.token);

    console.log("login fetched");
    window.location.replace("/");
  } else {
    alert(responseData.errors);
  }
};



return (

  <div className='loginsignup'>
    <div className="loginsignup-container">
      <h1>{state}</h1>
      <div className='loginsignup-fields'>
        {state === "Sign Up" ? <>
        <div className='loginsignup-fields-radio'>
        <label><input type='radio' name='usertype'
         value= 'user'  checked={formData.usertype === 'user'}
          onChange={changeHandler}/>user</label>
        <label><input type='radio' name='usertype'
         value= 'agent'  checked={formData.usertype === 'agent'}
          onChange={changeHandler}/>Agent</label></div>
        <input name="username" value={formData.username}
         onChange={changeHandler} type='text'
          placeholder='Username' /> </>:<></>}
        <input name='email' value={formData.email}
         onChange={changeHandler} type='email'
         placeholder='Email Address' />
        <input name='password' value={formData.password}
```

```
            onChange={changeHandler} type='password'
            placeholder='Password' />
          </div>
          <button onClick={() => { state === "Login" ?
           login() : signup() }}>Continue</button>
          {state === "Sign Up" ? <p className="loginsignup-login">
          Already have an account? <span onClick={() =>
          { setState("Login") }}>
            Login here</span></p> : <p className="loginsignup-login">
            Create an account? <span onClick={() =>
            { setState("Sign Up") }}>Click here</span></p>}
          <div className='loginsignup-agree'>

            <input type='checkbox' name='' id='' />
            <p>By continuing, I agree to the terms
            of use & privacy policy</p>
          </div>
        </div>
      </div>
      );

};
```

G.9 Loginsignup.css

```
.loginsignup{
    width: 100%;
    height: 100vh;
    background: white;

    display: flex;
    justify-content:start; /* Horizontally center the content */
    align-items: center;
```

```
}
.loginsignup-container{
    width: 580px;
    height: auto;
    background: rgb(247, 230, 230);
    margin:auto;
    padding: 40px 60px;
    text-align: center;

}
.loginsignup-container h1{
    margin: 20px 0px;
    font-size: 30px;
    text-decoration:underline;
}
.loginsignup-fields{
    display: flex;
    flex-direction: column;
    gap: 29px;
    margin-top: 30px;
}
.loginsignup-fields-radio {
    display: flex;
    gap: 40px;
}

.loginsignup-fields-radio label {
    margin-right: 20px;
    font-size: 25px;
    color: #141414; /* Choose your desired text color */
}

.loginsignup-fields-radio input[type='radio'] {
    margin-right: 20px;
    cursor: pointer;
    width: 30px;
```

```
    height: 30px;
}

.loginsignup-fields-radio input[type='radio']:checked + label {
    font-weight: bold;
    color: #007bff; /* Choose your desired color for the checked label */
}

.loginsignup-fields input{
    height: 72px;
    width: 100%;
    padding-left: 20px;
    border: 1px solid rgb(174, 163, 163);
    outline: none;
    color:grey ;
    font-size: 18px;
}
.loginsignup-container button{
    width: 580px;
    height: 72px;
    color: aliceblue;
    background: rgb(244, 89, 89);
    margin-top: 30px;
    border: none;
    font-size: 24px;
    font-weight: 500;
    cursor: pointer;
}
.loginsignup-login{
    margin-top: 20px;
    color: blueviolet;
    font-size: 18px;
    font-weight: 500;

}
.loginsignup-login span{
```

```
    color: rgb(243, 84, 84);
    font-weight: 600;
    cursor: pointer;

}
.loginsignup-agree{
    display: flex;
    align-items: center;
    margin-top: 25px;
    gap: 20px;
    color: cadetblue;
    font-size: 18px;
    font-weight: 500;
}
```

G.10 index.js

```
const port = 4000;
const express = require("express");
const app = express();
const mongoose = require("mongoose");
const jwt = require("jsonwebtoken");
const multer = require("multer");
const path = require("path");
const cors = require("cors");


app.use(express.json())
app.use(cors());

//database connection with mongodb

mongoose.connect("mongodb://127.0.0.1:27017")

//api creation

app.get("/",(req,res)=>{
```

```
    res.send("express app is running")
})

// image storage engine

const storage = multer.diskStorage({
  destination:'./upload/images',
  filename:(req,file,cb)=>{
    return cb(null,`${file.fieldname}_${Date.now()}
    ${path.extname(file.originalname)}`)
  }

})

const upload = multer({storage:storage})

//creating upload endpoint for images

app.use('/images',express.static('upload/images'))

app.post('/upload',upload.single('product'),(req,res)=>{
    res.json({
        success:1,
        image_url:`http://localhost:${port}/images/${req.file.filename}`
    })
})

//create schema for adding products

const Product = mongoose.model("Product",{
    id:{
        type: Number,
        required:true,
    },
    name:{
        type: String,
```

```
        required:true,
    },
    image:{
        type: String,
        required:true,
    },
    category:{
        type: String,
        required:true,
    },
    new_price:{
        type:Number,
        required:true,
    },
    old_price:{
        type:Number,
        required:true,
    },
    date:{
        type:Date,
        default:Date.now,
    },
    available:{
        type:Boolean,
        default:true,
    },
    agentId:{
        type: String,
        required:true,
    },
    agentName:{
        type: String,
        required:true,
    },
    description:{
        type:String,
```

```
        required:false,
    }
    });
```

```
// Endpoint to fetch and show all users with their
usertype (separate admin and regular users)

app.get('/allusers', async (req,res)=>{
    let users = await Users.find({});
    console.log("All user fetched");
    res.send(users);
})

// Backend endpoint to remove a user by ID

app.post('/removeuser', async (req, res) => {
    try {
        const user = await Users.findOneAndDelete({ uid: req.body.uid });
        if (!user) {
            return res.status(404).json({ success: false, error:
             'User not found' });
        }
        console.log('User removed:', user.name);
        res.status(200).json({ success: true, name: user.name });
    } catch (error) {
        console.error('Error removing user:', error.message);
        res.status(500).json({ success: false, error:
         'Internal server error' });
    }
});
```

```
//schema for users

const Users = mongoose .model('Users',{
    uid:{
        type: Number,
        required:true,
    },
    name:{
        type:String,
        required:true,
    },
    email:{
        type:String,
        required:true,
    },
    password:{
        type:String,
        required:true,
    },
    cartData:{
        type:Object,
    },
    uploadData:{
        type:Object,

    },
    date:{
        type:Date,
        default:Date.now,
    },
    usertype:{
```

```
      type:String,
      required:true,

    },
    approved: {
        type: Boolean,
        default: false, // Set default to false for new users
    },
})


//creating endpont for registration
app.post('/signup', async (req, res) => {
    const { username, email, password, usertype } = req.body;

    // Check if email or password is null
    if (!email || !password || !username || !usertype) {
        return res.status(400).json({ success: false, errors:
         "All fields are required" });
    }

    // Check if user with the same email already exists
    let check = await Users.findOne({ email });
    if (check) {
        return res.status(400).json({ success: false, errors:
         "An existing user with the same email ID already exists" });
    }

    let cart = {};
    for (let i = 0; i < 300; i++) {
        cart[i] = 0;
    }

    let upload = {};
    for (let i = 0; i < 150; i++) {
        upload[i] = 0;
```

```
}

let approved = false; // Default approval status

// Check if the user type is admin, if yes,
set approval to false (requires approval)
if (usertype === 'agent') {
    approved = false;
} else {
    approved = true; // For non-admin users, set approval to true
}

let users = await Users.find({});
let _Id;
    if (users.length > 0) {
        let lastUser = users[users.length - 1];
        _Id = lastUser.uid + 1;
    } else {
        _Id = 1;
    }

const user = new Users({
    uid: _Id,
    name: req.body.username,
    email: req.body.email,
    password: req.body.password,
    cartData: cart,
    usertype: req.body.usertype,
    uploadData: upload,
    approved: approved, // Set approval status based on user type
});

await user.save();

const data = {
    user: {
```

```
            id: user.id
        }
    }

    const token = jwt.sign(data, 'secret_ecom');
    res.json({ success: true, token });
});


  // Example backend endpoint for fetching users pending approval
app.get('/admin/users/pending', async (req, res) => {
    try {
        const pendingUsers = await Users.find({ approved: false });
        res.json({ success: true, users: pendingUsers });
    } catch (error) {
        console.error('Error fetching pending users:', error);
        res.status(500).json({ success: false, error:
        'Internal server error' });
    }
});


// Create an admin approval endpoint
app.patch('/admin/approve/:userId', async (req, res) => {
    const userId = req.params.userId;

    try {
        const user = await Users.findByIdAndUpdate(userId,
         { approved: true }, { new: true });

        if (!user) {
            return res.status(404).json({ success: false,
            error: 'User not found' });
        }

        res.json({ success: true, message:
```

```
            'User approved successfully' });
    } catch (error) {
        console.error('Error approving user:', error);
        res.status(500).json({ success: false, error:
         'Internal server error' });
    }
});

//creating endpoint for user login
app.post('/login',async(req,res)=>{
    let user = await Users.findOne({email:req.body.email});
    if(user){
        const passCompare = req.body.password ===
         user.password;
        if(passCompare){
            const data = {
                user:{
                    id:user.id,

                }
            }
            const token = jwt.sign(data,'secret_ecom');
            res.json({success:true,token});
        }
        else{
            res.json({success:false,errors:"wrong password"});
        }

    } else{
        res.json({success:false,errors:"wrong email-id"});
    }
})



const bookingSchema = new mongoose.Schema({
```

```
    userId: {
        type:String,
        required: true
    },
    userName: {
        type:String,
        required: true
    },
        productName: {
            type: String,
            required: true
        },
        Number: {
            type: Number,
            required: true
        },
        agentId: {
            type: String,
            required: true
        },
        agentName: {
            type: String,
            required: true
        },
        approved: {
            type: Boolean,
            default: false,  // Set default to false for new users
        },

    created_at: {
      type: Date,
      default: Date.now,
    },
});
const Booking = mongoose.model('Booking', bookingSchema);
```

```
app.post('/bookings', async (req, res) => {
  const token = req.header('auth-token');
  if (!token) {
      return res.status(405).send({ errors:
       "Please authenticate using a valid token" });
  }

  try {
      const data = jwt.verify(token, 'secret_ecom');
      const user = await Users.findOne({ _id: data.user.id });
      if (!user) {
          return res.status(404).send({ errors: "User not found" });
      }

      const userId = user.id;
      const userName = user.name; // Assuming name is the user ID

      // Create the booking
      const booking = new Booking({
          userId,
          userName,
          productName:req.body.productName,
          Number:req.body.Number,
          agentId: req.body.agentId,
          agentName: req.body.agentName

      });

      await booking.save();

      res.status(201).json({ success: true, message:
      'Bookings created successfully' });
  } catch (error) {
      console.error('Error creating booking:', error);
      res.status(500).json({ error: 'Internal server error' });
  }
```

```
});



// Define a route to find all bookings by agent ID
app.get('/bookings/agent', async (req, res) => {
    const token = req.header('auth-token');
    if (!token) {
        return res.status(405).send({ error:
         "Please authenticate using a valid token" });
    }

    try {
        // Verify the token to get the user's ID
        const data = jwt.verify(token, 'secret_ecom');
        const user = await Users.findOne({ _id: data.user.id });
        if (!user) {
            return res.status(404).send({ error:
             "User not found" });
        }

        // Get the agent ID from the user data
        const agentId = user.id;

        // Find all bookings with the specified agentId
        const bookings = await Booking.find({ agentId });

        // Check if there are no bookings found
        if (!bookings || bookings.length === 0) {
            return res.status(404).json({ error:
             'No bookings found for this agent' });
        }

        // Send the bookings in the response
        res.status(200).json({ success: true, bookings });
```

```
    } catch (error) {
        console.error('Error finding bookings:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});



// Define a route to approve or reject a booking
app.put('/bookings/:bookingId/approve', async (req, res) => {
    const token = req.header('auth-token');
    if (!token) {
        return res.status(405).send({ error:
         "Please authenticate using a valid token" });
    }

    try {
        // Verify the token to get the user's ID
        const data = jwt.verify(token, 'secret_ecom');
        const user = await Users.findOne({ _id: data.user.id });
        if (!user) {
            return res.status(404).send({ error: "User not found" });
        }


        // Get the booking ID from the request params
        const bookingId = req.params.bookingId;

        // Find the booking by ID
        const booking = await Booking.findOne({ _id: bookingId });
        if (!booking) {
            return res.status(404).send({ error: "Booking not found" });
        }

        // Update the approved status based on the request body
        booking.approved = req.body.approved;
```

```
        // Save the updated booking
        await booking.save();

        // Send success response
        res.status(200).json({ success: true, message:
         'Booking status updated successfully' });
    } catch (error) {
        console.error('Error updating booking status:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

//to get user details on profile
app.get('/user-details', async (req, res) => {
    const token = req.header('auth-token');
    if (!token) {
        return res.status(405).send({ errors:
         "Please authenticate using a valid token" });
    }

    try {
        const data = jwt.verify(token, 'secret_ecom');
        const user = await Users.findOne({ _id: data.user.id });
        if (!user) {
            return res.status(404).send({ errors: "User not found" });
        }

        const userDetails = {
            id: user._id, // Assuming user ID is stored in _id field
            email: user.email,
            username: user.name,
            usertype: user.usertype,
            // Add other user details as needed
        };
        res.status(200).json({ success: true, user: userDetails });
```

```
    } catch (error) {
        console.error('Error fetching user:', error);
        res.status(402).send({ errors:
        "Please authenticate using valid tokens" });
    }
});

//each users booked item
app.get('/bookings/user', async (req, res) => {
    const token = req.header('auth-token');
    if (!token) {
        return res.status(405).send({ error:
         "Please authenticate using a valid token" });
    }

    try {
        // Verify the token to get the user's ID
        const data = jwt.verify(token, 'secret_ecom');

        // Find all bookings with the specified user ID
        const bookings = await Booking.find({ userId:
         data.user.id });

        // Check if there are no bookings found
        if (!bookings || bookings.length === 0) {
            return res.status(404).json({ error:
             'No bookings found for this user' });
        }

        // Send the bookings in the response
        res.status(200).json({ success: true, bookings });
    } catch (error) {
        console.error('Error fetching user bookings:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});
```

```
//creating middleware to fetch user

const fetchUser = async (req,res,next)=>{
    const token = req.header('auth-token');
    if(!token) {
        res.status(405).send({errors:
        "please authenticate using valid token"});
    }
    else{
        try {
            const data = jwt.verify(token,'secret_ecom');
            req.user = data.user;
            next();
        } catch (error){
                res.status(402).send({errors:
                "please authenticate using valid tokens"})
        }
    }
}
```

```
// Function to check user type based on the decoded
 user object in the request
const checkUserType = async (req, res, next) => {
    try {
        let user = await Users.findById(req.user.id);
      if (user.usertype === 'agent'&& user.approved) {
        next();
```

```
      } else {
        return res.status(403).send({
            success:false,
            message:"UnAuthorised access"
        });
      }
    } catch (error) {
      console.log(error);
    }
  };

  // Function to check user type based on the
   decoded user object in the request
const User = async (req, res, next) => {
    try {
        let user = await Users.findById(req.user.id);
        if (user.usertype === 'user') {
            next();
        } else {
            return res.status(403).send({
                success: false,
                message: "Unauthorized access"
            });
        }
    } catch (error) {
        console.log(error);
    }
};

  app.get('/check-usertype', fetchUser, checkUserType,
   (req, res) => {
    try {
      res.send(ok=true);
    } catch (error) {
      console.log(error);
      res.status(500).send({ errors: 'Internal server error' });
```

```
    }
  });



// Agent adds product
app.post('/addproductagent', fetchUser, async (req, res) => {
    try {
        const token = req.header('auth-token');
        // Get the JWT token from the request header
        if (!token) {
            return res.status(401).json({ error: 'Unauthorized' });
        }

        const data = jwt.verify(token, 'secret_ecom');
        // Verify the JWT token
        if (!data || !data.user || !data.user.id) {
            return res.status(401).json({ error: 'Invalid token' });
        }

        const agent = await Users.findOne({ _id: data.user.id });
        if (!agent) {
            return res.status(404).json({ error: 'Agent not found' });
        }

        const agentId = agent.id;
        const agentName = agent.name;

        let products = await Product.find({});
        let id;
        if (products.length > 0) {
            let lastProduct = products[products.length - 1];
            id = lastProduct.id + 1;
        } else {
            id = 1;
```

```
}

const product = new Product({
    id: id,
    agentId: agentId ,
    agentName: agentName,
    name: req.body.name,
    image: req.body.image,
    category: req.body.category,
    new_price: req.body.new_price,
    old_price: req.body.old_price,
    description: req.body.description
});

// Save the product to the database
await product.save();


// Update user's uploadData with the new product ID
let userData = await Users.findOne({ _id: req.user.id });
if (userData) {
    // Initialize uploadData as an object if it doesn't exist
    if (!userData.uploadData) {
        userData.uploadData = {};
    }
    // Increment the count of the uploaded
     product in uploadData
    userData.uploadData[product.id] =
     (userData.uploadData[product.id] || 0) + 1;
    // Save the updated userData
    await Users.findOneAndUpdate({ _id: req.user.id },
    { uploadData: userData.uploadData });
    res.json({ success: true, product: product });
} else {
    res.status(404).json({ error: 'User not found' });
}
```

```
        console.log("Product added and uploaded by agent:",
         agentName); // Log the admin's name
    } catch (error) {
        console.error('Error adding product:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});



app.post('/getuploadedproducts', fetchUser, async (req, res) => {
    console.log("Get Uploaded Products");
    try {
        let userData = await Users.findOne({ _id: req.user.id });
        if (userData) {
            // Transform uploadData object into an array of products
            const uploadedProducts =
            Object.values(userData.uploadData || {});
            res.json({ success: true, uploadedProducts });
        } else {
            res.status(404).json({ success: false, error:
             'User not found' });
        }
    } catch (error) {
        console.error('Error fetching uploaded products:', error);
        res.status(500).json({ success: false, error:
        'Internal server error' });
    }
})



//Admin adding and removing products
```

```
app.post('/addproduct', fetchUser, async (req, res) => {
    try {
        const { id: uploaderId } = req.user;
        console.log('Uploader ID:', uploaderId);
         // Check if uploaderId is correct

        let products = await Product.find({});
        let id;
        if (products.length > 0) {
            let last_product_array = products.slice(-1);
            let last_product = last_product_array[0];
            id = last_product.id + 1;
        } else {
            id = 1;
        }

        const product = new Product({
            id: id,
            name: req.body.name,
            image: req.body.image,
            category: req.body.category,
            new_price: req.body.new_price,
            old_price: req.body.old_price,

        });
        await product.save();
        console.log("Product added by user:", req.user.name);
        res.json({ success: true, name: req.body.name });
    } catch (error) {
        console.error('Error adding product:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});
```

```
//remove api

app.post('/removeproduct',async (req,res)=>{
await Product.findOneAndDelete({id:req.body.id});
console.log("removed");
res.json({
    success:true,
    name:req.body.name
})
})

//all products api

app.get('/allproducts', async (req,res)=>{
    let products = await Product.find({});
    console.log("All products fetched");
    res.send(products);
})




//creating endpoint for newcollection data
app.get('/newcollections',async (req,res)=>{
    let products = await Product.find({});
    let newcollection = products.slice(0).slice(-8);
    console.log("NewCollection Fetched");
    res.send(newcollection);
})

//creating endpoint for popular in women section
app.get('/popularinwildlife',async (req,res)=>{
    let products = await Product.find({category:"wildlife"});
    let popular_in_wildlife = products.slice(0,4);
```

```
    console.log("Popular in wildlife fetched");
    res.send(popular_in_wildlife);
})



// creating endpoint for products in cartdata
app.post('/addtocart',fetchUser,User, async (req,res)=>{
    console.log("Added",req.body.itemId)
    let userData = await Users.findOne({_id:req.user.id});
    userData.cartData[req.body.itemId] += 1
    await Users.findOneAndUpdate({_id:req.user.id},
    {cartData:userData.cartData});
    res.send('Added');
})

//creating endpoint to remove product from cart
app.post('/removefromcart',fetchUser,async (req,res)=>{
    console.log("removed",req.body.itemId)
let userData = await Users.findOne({_id:req.user.id});
if(userData.cartData[req.body.itemId]>0)
userData.cartData[req.body.itemId] -= 1
await Users.findOneAndUpdate({_id:req.user.id},
{cartData:userData.cartData});
res.send('removed');
})

//creating endpoint to get cartdata
app.post('/getcart',fetchUser,User, async (req,res)=>{
    console.log("GetCart");
    let userData= await Users.findOne({_id:req.user.id});
    res.json(userData.cartData);
})

app.listen(port,(error)=>{
    if (!error) {
```

```
        console.log(`Server running on port ${port}`);
    }
    else{
        console.log("Error :" +error)
    }
})
```

Bibliography

[1] *First Lessons in LaTeX*, by Dr. V N Krishnachandran, Vidya Academy of Science & Technology, Thrissur - 680 501, 2011.

[2] *https://www.w3schools.com/REACT/DEFAULT.ASP*

[3] *https://www.geeksforgeeks.org/mern-stack/*

[4] *https://www.w3schools.com/mongodb/*

[5] *https://youtu.be/ZVyIIyZJutM?si=ATymSWbUXyt5blBW*

[6] *https://youtu.be/y99YgaQjgx4?si=tU4Xk76FvMM9zodC*

Department of Computer Applications
Vidya Academy of Science & Technology
Thalakkottukara, Thrissur - 680 501
(http://www.vidyaacademy.ac.in)