PRACTICAL 1

Aim: Project Definition and objective of the specified module and Perform Requirement Engineering Process.

Definition:

The project, **WANDERMATE**, aims to create a travel experience platform focusing on personalized travel itineraries, local guides, and group travel opportunities.

Objective of the Specified Module:

Its primary goal is to provide users with stress-free trip planning while integrating budgets, preferences, and interests. Key features include:

- **Personalized Itineraries**: Tailored to user preferences and budgets, including local attractions, hidden gems, and travel tips.
- **Local Guides**: Detailed profiles for guides, showcasing language proficiency, expertise, and availability.
- **Social Travel**: Facilitating group tours to foster cost-effective and interactive travel experiences.

Requirement Engineering Process:

Gathering user needs through:

- Surveys to understand traveler preferences for destinations, activities, and budget limits.
- Interviews with users to identify pain points in existing travel platforms.
- Feedback collection on features like local guide booking, AI recommendations, and group travel coordination.

Requirement Analysis

Key areas of focus include:

- **Functional Feasibility**: Ensuring features like real-time itinerary updates, booking APIs, and group trip handling are technically achievable.
- **Prioritization**: Giving precedence to critical functionalities like user-friendly itinerary customization and guide matching systems.

Requirement Specification

Documented requirements include:

- **Functional Requirements**:
  - Dynamic travel itinerary generation.
  - Local guide profile creation and booking.
  - Group tour organization and participation features.
- **Non-Functional Requirements**:
  - Scalability to handle high traffic.
  - Robust security for user data.
  - Mobile-friendly, intuitive UI for seamless navigation.

Requirement Validation

Prototyping and iterative user feedback ensure:

- User needs are met accurately.
- The system aligns with the platform's goals.
- Continuous refinement of features like real-time notifications and hidden gem suggestions.

PRACTICAL 2

Aim: Identify Suitable Design and Implementation models from the different software engineering models.

Agile Model

The **Agile Model** is selected for the design and implementation of WANDERMATE due to its adaptability, user-centric focus, and iterative approach. Key reasons include:

- **Dynamic Requirements**: The travel platform demands frequent updates based on user feedback and market trends.
- **Iterative Delivery**: Allows continuous feature enhancement, such as AI-powered recommendations and local guide integration.
- **User Collaboration**: Encourages ongoing input from stakeholders, ensuring the platform meets user expectations.

Implementation Steps:

- **Planning**: Define high-priority features like itinerary generation and group travel coordination.
- **Iteration Development**: Develop features in sprints, such as integrating booking APIs or implementing real-time updates.
- **Testing**: Validate each sprint output through user feedback and testing.
- **Release**: Deploy incremental improvements regularly.

Advantages of Agile Model:

- **Flexibility**: Adapts to changing requirements and evolving user needs.
- **Faster Delivery**: Enables early release of an MVP for quick user engagement.
- **User-Centric**: Regular feedback ensures features align with user expectations.
- **Risk Reduction**: Continuous testing minimizes errors and large-scale failures.
- **Improved Collaboration**: Promotes teamwork and stakeholder involvement.
- **Iterative Progress**: Allows gradual enhancement through manageable increments.
- **Market Responsiveness**: Quick updates ensure the platform remains competitive.

Prototype Model

The **Prototype Model** is used alongside Agile to refine user interface (UI) and experience (UX) designs.

- **Purpose**: Quickly create mock-ups of key features like personalized itineraries and social travel coordination to gather user feedback.
- **Benefits**: Enhances user engagement and ensures the interface is intuitive and visually appealing.

Design Models Used

A. Architectural Design

- **Model**: Layered Architecture.
  - **Reason**: Separation of concerns, with layers for user interface, business logic, and data management.
  - **Example**: UI/UX for travel itinerary customization in the presentation layer, AI algorithms in the business layer, and databases in the data layer.

B. Component-Based Design

- **Reason**: Modular development of independent components such as:
  - User account management.
  - Local guide booking system.
  - Group travel coordination.
  - AI-powered recommendations.

C. User-Centred Design (UCD)

- **Focus**: Prioritizes user needs and usability through:
  - Intuitive navigation.
  - Mobile-responsive layouts.
  - Customizable itinerary management.

Implementation Strategy

- **Technology Stack**: Leveraging modern technologies such as:
  - Frontend: React.js for dynamic and interactive UI.
  - Backend: Node.js with Express.js for scalable server-side logic.
  - Database: MongoDB for flexibility in storing user preferences and itineraries.
  - APIs: Google Maps API for location-based services.
- **DevOps Practices**:
  - CI/CD pipelines for regular updates.
  - Containerization using Docker for consistent environments.

Justification:

- **Dynamic Requirements**
  - Travel industry trends, user preferences, and technological advancements can change rapidly. Agile accommodates these changes without derailing the project.
- **Focus on User Experience (UX)**
  - Features like itinerary customization, hidden gem suggestions, and AI-powered recommendations require continuous feedback to meet diverse traveller needs effectively.
- **Competitive Market**
  - Launching an MVP quickly with core features like personalized itineraries ensures **WANDERMATE** gains an early foothold in the market while enabling incremental improvements.
- **Modular Development Needs**
  - The platform's functionality (e.g., itinerary planning, guide booking, group travel) can be developed and tested in parallel using Agile's iterative approach.
- **Stakeholder Involvement**
  - Close collaboration ensures the platform reflects the vision of stakeholders and evolves based on real-world usage data.

PRACTICAL 3

Aim: Prepare Software Requirement Specification (SRS) for the selected module.

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for the **Itinerary Customization Module** of **WANDERMATE**, a travel experience platform. This module allows users to generate, customize, and manage personalized travel plans based on preferences, budget, and interests.

1.2 Scope

This module aims to:

- Offer tailored travel itineraries including destinations, accommodations, and activities.
- Enable dynamic updates based on real-time events (e.g., weather or availability).
- Integrate with other modules, such as local guide booking, group travel coordination, and **subscription-based premium features**.

1.3 Definitions, Acronyms, and Abbreviations

- **MVP**: Minimum Viable Product
- **UI**: User Interface
- **API**: Application Programming Interface

1.4 References

- Agile Model Documentation
- Requirement Gathering Documents

2. Overall Description

2.1 Product Perspective

The **Itinerary Customization Module** is part of **WANDERMATE's** travel platform and will interact with:

- **User Account Module**: To fetch user preferences and past travel history.
- **Subscription Module**: To unlock premium features such as exclusive itineraries, discounts, and concierge services for subscribers.
- **API Integrations**: For accommodation, travel services, and maps.
- **Payment Gateway**: For paid itinerary features and subscriptions.

2.2 Product Features

- Generate itineraries based on user inputs like destination, travel style, and budget.
- Provide real-time updates, such as event notifications or travel disruptions.
- Enable users to add, remove, or reorder items in the itinerary.
- Suggest nearby attractions, local restaurants, and hidden gems.
- Offer **exclusive content and features** for subscribed users, such as:
    - Premium itineraries.
    - Discounts on local guides and group travel bookings.
    - Early access to seasonal or promotional group tours.

2.3 User Characteristics

- **Primary Users**: Independent travellers, families, group adventurers, and premium subscribers.
- **Technical Knowledge**: Basic familiarity with web or mobile apps.

3. Functional Requirements

3.1 Itinerary Generation

- The system shall allow users to input preferences (e.g., budget, destination, interests).
- The system shall generate an initial itinerary with accommodations, activities, and local guides.
- **For Subscribers**: The system shall offer premium itinerary suggestions with exclusive options.

3.2 Itinerary Customization

- The system shall allow users to edit, reorder, or add new items to the itinerary.
- The system shall integrate with APIs for real-time updates on weather, local events, and availability.
- **For Subscribers**: The system shall provide additional customization options, such as advanced filters for eco-friendly travel or luxury accommodations.

3.3 Recommendations

- The system shall suggest nearby attractions and dining options based on location.
- The system shall recommend group tours or local experiences.
- **For Subscribers**: Priority access to group tours and personalized recommendations based on advanced AI insights.

3.4 Subscription Management

- The system shall allow users to subscribe to a premium plan via the platform.
- The system shall enable subscribers to access exclusive itineraries, discounts, and other benefits.
- The system shall notify users about subscription expiration and renewal options.

4. Non-Functional Requirements

- **Performance**: The system should respond within 2 seconds for itinerary updates, including subscription-based features.
- **Scalability**: The module must support up to 1,000 concurrent users, with priority processing for subscribers.
- **Usability**: The interface should be intuitive and mobile-friendly, with clear access to subscription benefits.
- **Security**: Ensure secure handling of subscription payments and user data using encryption.

5. Assumptions and Dependencies

- Availability of APIs for maps, local guides, and accommodation data.
- Stable internet connection for real-time updates.
- Integration of a robust subscription management system.

PRACTICAL-4

Aim: Develop Software project management planning (SPMP) for the specified module.

1. Introduction

- Purpose: This SPMP outlines the project management approach for the development of the Itinerary Customization Module in WANDERMATE. It defines processes, schedules, resources, and risk management strategies.
- Scope: Covers activities like requirements gathering, design, development, testing, deployment, and subscription feature integration.

2. Project Organization

- **Team Roles:**
  - Project Manager: Oversees project progress and ensures milestones are met.
  - Developers: Build the customization and subscription modules.
  - UI/UX Designers: Design a user-friendly interface.
  - Testers: Ensure quality and functionality.
  - Stakeholders: Provide feedback and approve deliverables.

3. Project Management Process

3.1 Development Approach

- Agile methodology for iterative development with regular feedback cycles.

3.2 Milestones

- Requirements Finalization: Week 1-2.
- Module Design: Week 3-4.
- Development Phase 1 (Core Features): Week 5-8.
- Testing & Debugging: Week 9-10.
- Deployment of MVP: Week 11.
- Subscription Module Integration: Week 12-14.

4. Work Breakdown Structure (WBS)

1. **Requirement Engineering**
   - Gather user stories and functional requirements.
   - Prioritize features (e.g., itinerary customization, subscription benefits).
2. **Design**
   - Create wireframes for core functionalities.
   - Prepare technical architecture.
3. **Implementation**
   - Develop core itinerary customization features.
   - Integrate APIs (maps, travel data).
   - Implement subscription-based features.
4. **Testing**
   - Functional testing of customization.
   - User acceptance testing for premium features.
5. **Deployment**
   - Launch MVP.
   - Add subscription features in post-launch updates.

5. Resources

- **Tools:**
  - Frontend: React or Vue.js.
  - Backend: Node.js or Django.
  - APIs: Google Maps, Travel Booking APIs.
  - Subscription Management: Stripe or Razor pay.
- **Team Members:**
  - 1 Project Manager, 2 Developers, 1 UI/UX Designer, 1 Tester.

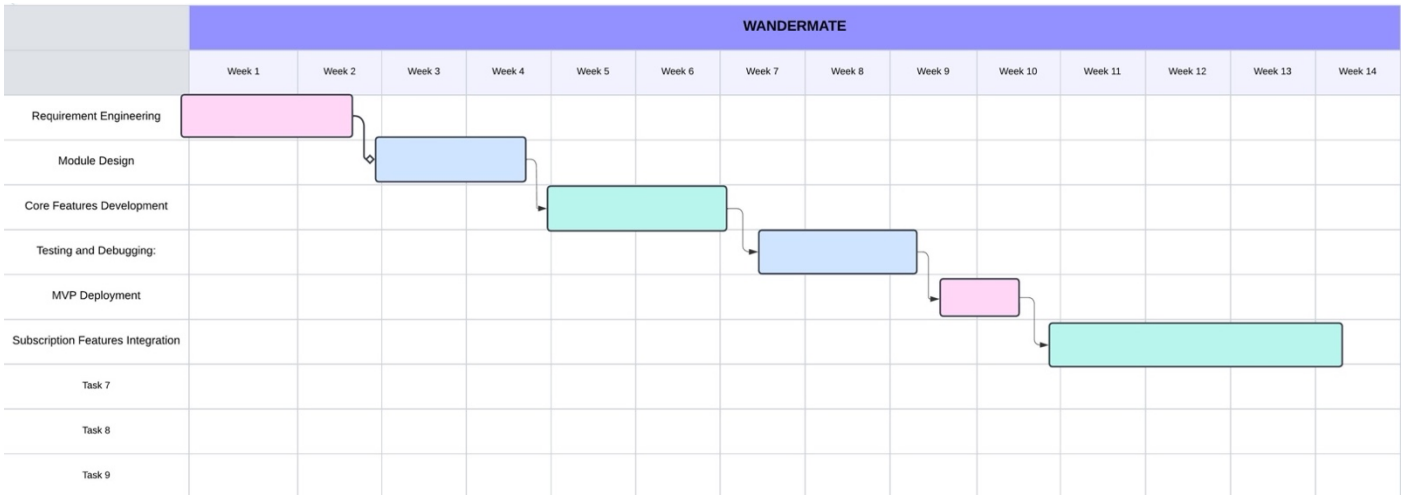
6. Risk Management

- **Risk 1:** Changing user requirements.
  - Mitigation: Use Agile for iterative updates.
- **Risk 2:** API failure or downtime.
  - Mitigation: Implement fallback mechanisms and alternative APIs.
- **Risk 3:** Budget constraints for subscription features.
  - Mitigation: Release features incrementally to manage costs.

7. Schedule

- The total duration is 14 weeks, divided into sprints to focus on incremental delivery.

GAANT CHART:


WANDERMATE
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Requirement Engineering
Module Design
Core Features Development
Testing and Debugging:
MVP Deployment
Subscription Features Integration
Task 7
Task 8
Task 9

PRACTICAL-5

Aim: **Do Cost Estimation use Different Software Cost Estimation Models.**

FUNCTION POINT:

Step 1: Function Counts

Function Type	Count	Complexity Weight	Total
External Inputs	12	6	72
External Outputs	18	7	126
Inquiries	15	6	90
Logical Files	10	15	150
Interface Files	14	10	140

Total Count: 578

Step 2: Adjustment Factors

- Selected Complexity Factors: **f1=4, f3=3, f5=2, f6=2, f12=1, f14=3, f15=5**
- Sum of (fi) = **20**

Step 3: Function Point Calculation

FP=657×[0.65+0.01(21)]
FP=657×0.86FP = 657
FP=565.02FP

Lines of Code (LOC) Estimation:

- **Assumption:** 1 FP = **30 LOC**

LOC=565.02×32=18,080.64LOC
KLOC=18080.64/1000=18.080KLOC

COCOMO MODEL :

Step 1: Effort Calculation:

EFFORT=a1×(KLOC)a2
EFFORT=2.4×(18.080)1.05
EFFORT=50.149 Person-Months

Step 2: Time Development (TDEV) Calculation:

TDEV=b1×(EFFORT)b2
TDEV=2.5×(50.149)0.38
TDEV=11.06 Months

Step 3: Cost Estimation:

COST=TDEV × No. of Employees× Avg Salary
COST=11.06×100×30,000
INR}COST=3,31,80,000 INR

PRACTICAL-6

Aim: **Prepare System Analysis and System Design of Identified Requirement specification using structure design as DFD with Data Dictionary and Structure chart for the specific module.**

System Analysis and System Design for Travel Website:

1. System Analysis (Identified Requirement Specification):
   -> Overview of the System:

The travel website aims to provide a seamless travel planning experience by offering personalized itineraries, local guide bookings, group travel options, and AI-powered recommendations.

Key Features & Requirements:

1. **User Registration & Authentication**
   - Users can register/login using email or social accounts.
   - Secure authentication using JWT or OAuth.
2. **Itinerary Planning**
   - Users can create and customize their travel itinerary.
   - AI-based suggestions for attractions, restaurants, and activities.
3. **Booking System**
   - Users can book hotels, local guides, and group tours.
   - Integration with payment gateways (Stripe, PayPal, etc.).
4. **Payment Integration**
   - Secure transactions via payment gateways.
   - Invoice generation and transaction history.
5. **User Reviews & Ratings**
   - Users can rate and review their travel experiences.
   - Reviews impact recommendations and guide listings.

2. System Design:

2.1 Data Flow Diagram (DFD):

DFD - Level 0 (Context Diagram)

- The system interacts with **Users, Payment Gateway, and External APIs** (for hotels, flights, etc.).
- Main processes include:
  - User Authentication
  - tinerary Management
  - Booking Management
  - Payment Processing
  - Review & Feedback

DFD - Level 1 (Detailed Breakdown)

- **User Authentication** → Register/Login, Update Profile.
- **Itinerary Management** → Create/Edit/Delete Itinerary, View Recommendations.
- **Booking System** → Select Service, Confirm Booking, Process Payment.
- **Review & Feedback** → Submit Review, Display Reviews.

2.2 Data Dictionary:

Entity	Attribute	Data Type	Description
User	UserID	Integer	Unique ID of the user
User	Name	String	Name of the user
User	Email	String	Email address
Itinerary	ItineraryID	Integer	Unique ID of the itinerary
Itinerary	UserID	Integer	ID of the user creating itinerary
Booking	BookingID	Integer	Unique ID of the booking
Booking	ServiceType	String	Type of service (Hotel, Guide, etc.)
Payment	PaymentID	Integer	Unique ID of the payment
Payment	Amount	Float	Transaction amount

2.3 Structure Chart:

The **hierarchical structure** of the system modules is as follows:

- **Main System**
  - **User Management**
    - Register/Login
    - Profile Management
  - **Itinerary Management**
    - Create/Edit/Delete Itinerary
    - View AI Recommendations
  - **Booking System**
    - Book Hotel/Guide/Tour
    - Confirm Payment
  - **Review & Feedback**
    - Submit Review
    - View Ratings & Comments

PRACTICAL-7

Aim: **Designing the module using Object Oriented approach including Use case Diagram with scenarios, Class Diagram and State Diagram, Sequence Diagram and Activity Diagram.**

3. Object-Oriented Design:

3.1 Use Case Diagram with Scenarios:

- **Actors:** User, Admin, Local Guide, Payment Gateway
- **Use Cases:**
  - Register/Login
  - Create/Edit Itinerary
  - Book Service
  - Process Payment
  - Submit/View Reviews

3.2 Class Diagram:

- **Classes:**
  - User (UserID, Name, Email, Password, Role)
  - Itinerary (ItineraryID, UserID, Destination, Activities)
  - Booking (BookingID, UserID, ServiceType, Status)
  - Payment (PaymentID, BookingID, Amount, Status)
  - Review (ReviewID, UserID, Rating, Comments)

3.3 State Diagram:

- **States:**
  - User Registration → Logged In → Itinerary Created → Booking Confirmed → Payment Processed → Review Submitted

3.4 Sequence Diagram:

Booking Flow:

1. User selects service
2. System validates availability
3. User confirms booking
4. Payment gateway processes transaction
5. Booking confirmation sent to user

3.5 Activity Diagram

Itinerary Creation:

1. **User logs in**
2. **Selects destination**
3. **Adds activities**
4. **Saves itinerary**
5. **Views recommendations**

PRACTICAL - 8

AIM : Defining Coding Standards and Walk through.

- **General Coding Standards :**

1. Naming Conventions
    - Use meaningful and descriptive names for variables, functions, classes, and other identifiers.
    - Follow a consistent naming convention (e.g., camel Case, snake_case, Pascal Case).
    - Avoid using single-letter variable names except for loop counters.
2. Code Structure
    - Organize code into functions or methods to promote reusability and readability.
    - Keep functions short and focused on a single task (Single Responsibility Principle).
    - Use consistent indentation (e.g., 2 spaces, 4 spaces, or tabs).
3. Commenting and Documentation
    - Write comments to explain complex logic or important decisions.
    - Use docstrings or comments to describe the purpose and usage of functions and classes.
    - Keep comments up to date with code changes.
4. Error Handling
    - Implement proper error handling using exceptions or error codes.
    - Avoid silent failures; log errors or provide meaningful feedback.

5. Code Formatting
    - Use consistent formatting for braces, parentheses, and whitespace.
    - Limit line length (commonly 80 or 120 characters).
    - Use blank lines to separate logical sections of code.
6. Version Control
    - Use a version control system (e.g., Git) for tracking changes.
    - Write clear and descriptive commit messages.
    - Follow a branching strategy (e.g., Git Flow, feature branches).
7. Testing
    - Write unit tests for critical components of the code.
    - Use test-driven development (TDD) where applicable.
    - Ensure tests are automated and run regularly.
8. Code Reviews
    - Conduct regular code reviews to ensure adherence to standards and improve code quality.
    - Provide constructive feedback and be open to receiving it.
9. Performance Considerations
    - Optimize code for performance where necessary, but prioritize readability and maintainability.
    - Avoid premature optimization; focus on clear and correct code first.
10. Security Practices
    - Validate and sanitize user inputs to prevent security vulnerabilities (e.g., SQL injection, XSS).
    - Use secure coding practices and libraries.

PRACTICAL - 9

AIM : Write the test cases for the identified module.

1. User Authentication Module:

Test Case ID	Test Scenario	Pre-Condition	Test Steps	Expected Result	Status
TC_001	Valid Login	User has a registered account	1. Open the login page 2. Enter valid email and password 3. Click the **Login** button	User successfully logs in and is redirected to the dashboard	Pass/Fail
TC_002	Invalid Login	User has an unregistered or incorrect password	1. Open the login page 2. Enter invalid email/password 3. Click **Login**	Error message **"Invalid credentials"** displayed	Pass/Fail
TC_003	Logout	User is logged in	1. Click on the **Logout** button 2. Confirm logout	User is logged out and redirected to the home page	Pass/Fail
TC_004	Forgot Password	User has a registered email	1. Click on **Forgot Password** 2. Enter a registered email 3. Click **Reset Password**	A reset password link is sent to the email	Pass/Fail

2. Search & Booking Module:

Test Case ID	Test Scenario	Pre-Condition	Test Steps	Expected Result	Status
TC_005	Search Destination	User is logged in	1. Enter destination and travel dates. 2. Click **Search**	List of available travel options is displayed	Pass/Fail
TC_006	Book a Trip	User is logged in and selected a trip	1. Click on a travel package. 2. Enter traveller details. 3. Proceed to payment. 4. Confirm booking	Booking confirmation is generated with a reference number	Pass/Fail
TC_007	Invalid Payment	User is on the payment page	1. Enter invalid card details. 2. Click **Pay Now**	Error message **"Invalid payment details"** displayed	Pass/Fail
TC_008	Cancel Booking	User has an active booking	1. Navigate to **My Bookings** 2. Click **Cancel Booking.** 3. Confirm cancellation	Booking is canceled, and refund status is updated	Pass/Fail

3. User Profile Management:

Test Case ID	Test Scenario	Pre-Condition	Test Steps	Expected Result	Status
TC_009	Update Profile	User is logged in	1. Navigate to **Profile.** 2. Edit name, phone number, or address 3. Click **Save Changes**	Profile updates successfully	Pass/Fail
TC_010	View Booking History	User has previous bookings	1. Navigate to **My Bookings.** 2. Click on **View History**	Past bookings are displayed	Pass/Fail

4. Admin Panel Module:

Test Case ID	Test Scenario	Pre-Condition	Test Steps	Expected Result	Status
TC_011	Add New Travel Package	Admin is logged in	1. Go to the **Admin Panel.** 2. Click **Add New Package.** 3. Enter package details. 4. Click **Submit**	Package is successfully added to the database	Pass/Fail
TC_012	Remove a Package	Admin is logged in and a package exists	1. Go to **Manage Packages.** 2. Select a package. 3. Click **Delete.** 4. Confirm deletion	Package is removed successfully	Pass/Fail

PRACTICAL - 10

AIM : Demonstrate the use of different Testing Tools with comparison.

1. Types of Testing Tools:

Testing tools are classified into the following categories:

1. **Test Management Tools** → Manage test cases, plans, and reports.
2. **Functional Testing Tools** → Perform automated functional testing.
3. **Performance Testing Tools** → Analyse the load, stress, and performance.
4. **Security Testing Tools** → Identify vulnerabilities in the application.
5. **API Testing Tools** → Test the functionality of APIs.
6. **Mobile Testing Tools** → Test mobile applications.

2. Comparison of Popular Testing Tools:

Feature	Selenium (Functional)	JMeter (Performance)	TestRail (Test Management)	Postman (API)	Burp Suite (Security)
Type	Functional Testing (Automation)	Performance & Load Testing	Test Case & Defect Management	API Testing	Security & Vulnerability Testing
Usage	Automating UI tests	Load testing for web & API	Managing test plans & execution	API validation, automation	Penetration testing
Best for	Web application testing	Performance & stress testing	Organizing test cases	REST, SOAP API testing	Finding security loopholes
Scripting Required?	Yes (Java, Python, etc.)	Yes (JMeter scripting)	No	No (Basic tests)	No (Automated scan)
Supports CI/CD?	Yes	Yes	Yes	Yes	Limited
Free Version?	Yes (Open-source)	Yes (Open-source)	No (Paid tool)	Yes	Yes (Limited)
Platform Support	Web (All browsers)	Web, API	Web-based	Web, API	Web, Mobile Apps

3. Demonstration of Each Tool

Selenium - Functional Testing (Example):

Scenario: **Test the login functionality of a travel website.**

```
python
CopyEdit
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()
driver.get("https://travelwebsite.com/login")

username = driver.find_element(By.NAME, "email")
password = driver.find_element(By.NAME, "password")

username.send_keys("testuser@example.com")
password.send_keys("password123")
password.send_keys(Keys.RETURN)

assert "Dashboard" in driver.title
driver.quit()
```

Output: **If the login is successful, the dashboard page will be displayed.**

2. Meter - Performance Testing (Example)

Scenario: **Test the response time of the website under 100 concurrent users.**

- Open **JMeter**
- Add a **Thread Group** with **100 users**
- Add **HTTP Request Sampler** (Enter the website URL)
- Add **Listeners** to view results
- Run the test and analyze **Response Time & Throughput**

Output: Generates graphs showing the server's response under load.

3. TestRail - Test Management (Example)

Scenario: **Manage manual test cases for a travel website.**

- Create a **Test Plan** in TestRail.
- Add **Test Cases** like "Valid Login," "Invalid Login," etc.
- Assign test cases to the team.
- Track test execution & report bugs.

Output: Detailed test execution reports.

4. Postman - API Testing (Example)

Scenario: **Test the** login API **of the travel website.**

- Open **Postman**
- Select **POST** method
- Enter API Endpoint: `https://api.travelwebsite.com/login`
- Add **JSON Body**:

```
json
CopyEdit
{
  "email": "testuser@example.com",
  "password": "password123"
}
```

- Click **Send**

Output: Response `{ "message": "Login Successful", "token": "xyz123" }`

5. Burp Suite - Security Testing (Example)

Scenario: **Identify security vulnerabilities (e.g., SQL Injection).**

- Start **Burp Suite**
- Intercept login requests.
- Modify SQL queries (e.g., enter `" OR 1=1 --` in the username field).
- Check if the system allows unauthorized access.

Output: If successful, the system is vulnerable to SQL Injection.

4. Conclusion: Choosing the Right Tool:

Testing Need	Recommended Tool
UI Automation	Selenium
Performance Testing	JMeter
Test Case Management	TestRail
API Testing	Postman
Security Testing	Burp Suite

PRACTICAL - 11

AIM : Define security and quality aspects of the identified module.

Security and Quality Aspects of the Identified Module:

1. Security Aspects

Security is crucial for ensuring that the identified module is protected against threats such as unauthorized access, data breaches, and malicious attacks. Below are the key security aspects:

🔹 **Authentication & Authorization**

- **Implement strong authentication mechanisms (e.g., OAuth, JWT).**
- **Enforce role-based access control (RBAC) to restrict user permissions.**

🔹 **Data Security & Encryption**

- **Use SSL/TLS encryption for data transmission.**
- **Store passwords using hashing algorithms like bcrypt or Argon2.**
- **Implement end-to-end encryption for sensitive data.**

🔹 **Input Validation & Protection Against Attacks**

- **Prevent SQL Injection using prepared statements.**
- **Sanitize user input to avoid cross-site scripting (XSS) attacks.**
- **Implement CSRF protection to prevent unauthorized requests.**

🔹 **Secure APIs**

- **Use API keys or OAuth for secure API access.**
- **Implement rate limiting to prevent DDoS attacks.**
- **Validate all API requests to ensure data integrity.**

🔹 **Logging & Monitoring**

- **Enable security logging to track login attempts, API calls, and unusual activities.**
- **Use intrusion detection systems (IDS) for real-time threat monitoring.**

🔹 **Secure Deployment & Patching**

- **Keep the server and application up to date with security patches.**
- **Use firewalls and security groups to limit external access.**
- **Implement container security best practices (if using Docker/Kubernetes).**

2. Quality Aspects

Quality ensures that the module meets functional and non-functional requirements, offering reliability and usability.

🔹 **Functional Quality**

- **Ensure the module meets business and user requirements.**
- **Conduct unit testing, integration testing, and system testing.**

🔹 **Performance & Scalability**

- **Optimize code for low latency and high efficiency.**
- **Perform load testing to ensure the system can handle concurrent users.**
- **Implement caching mechanisms for faster data retrieval.**

🔹 **Maintainability & Code Quality**

- **Follow coding standards and best practices (e.g., Clean Code, SOLID principles).**
- **Write modular, reusable, and well-documented code.**
- **Use version control (Git) to track changes efficiently.**

🔹 **Usability & Accessibility**

- **Ensure a user-friendly interface with a seamless experience.**
- **Follow WCAG guidelines for accessibility compliance.**
- **Provide error messages and tooltips for better user guidance.**

🔹 **Reliability & Fault Tolerance**

- **Implement automated backups and failover mechanisms.**
- **Use error handling and exception management for stability.**
- **Monitor uptime using logging and observability tools like Prometheus.**

3. Summary Table: Security vs. Quality Aspects:

Aspect	Security Measures	Quality Measures
Authentication	Strong passwords, OAuth, MFA	Ensure login functionality works correctly
Data Protection	Encryption, Secure storage	Maintain data integrity and consistency
Code Quality	Prevent SQL Injection, XSS	Follow coding best practices
Performance	Secure API endpoints, Rate limiting	Load testing, Optimization
Monitoring	Logging, IDS, Intrusion alerts	Performance tracking, Error handling