Testing Report: PCS 25 - 09

Ishita Srivastava 2100290120088

Gurpreet Kaur 2100290120082

Kritika 2100290120098

1 Introduction

A. Test Strategies

a. Functional Testing:

- Ensures that each feature of the application works according to the requirements.
- Includes login, booking, payment, notifications, and review features.

b. Integration Testing:

- Verifies the interactions between different modules, such as the connection between booking and payment systems or user notifications.
- Includes complex workflows like booking creation, payment processing, and notification delivery.

c. Boundary Value Analysis:

- Tests the application at the boundaries of input ranges to uncover edge case issues.

d. Equivalence Partitioning:

- Groups input data into valid and invalid equivalence classes for efficient testing.

e. Usability Testing:

- Validates that the user interface (UI) is intuitive and provides a smooth user experience.

B. Test Process

a. Requirement Analysis:

- Understand project requirements from functional specifications and user stories.
- Define acceptance criteria for each feature.

b. Test Planning:

- Develop a test plan outlining objectives, scope, resources, timelines, and deliverables.

- Identify tools and techniques for testing (e.g., Selenium, Postman).

c. **Test Design**:
   - Create test cases for individual modules, integration workflows, and edge cases.
   - Include boundary value and equivalence test scenarios.

d. **Test Environment Setup**:
   - Configure a test environment that mirrors production conditions.
   - Integrate the database, APIs, and cloud services for end-to-end testing.

e. **Test Execution**:
   - Execute test cases manually and through automation tools.

f. **Test Closure**:
   - Summarize testing activities, including the number of test cases executed, passed, failed, and blocked.
   - Prepare a final test report with recommendations for improvement.

C. Workflow

a. **Development Phase**:
   - Unit testing while developing.

b. **QA Testing Phase**:
   - Execute planned functional and integration tests.

c. **Bug Fixing and Verification**:
   - Defects are resolved and verified.
   - Regression tests ensure no existing functionality is broken.

d. **User Acceptance Testing (UAT)**:
   - End-users test the application to validate real-world scenarios.

e. **Deployment Phase**:
   - Conduct final testing in the production environment.

D. Methodologies Used

a. **Agile Methodology**:
   - Testing is integrated into the development cycle, allowing continuous feedback and iterative improvements.
   - Testing activities are planned for each sprint, ensuring timely validation of features.

b. **Automation Testing**:
   - Automation tools like Selenium and JUnit are used for repetitive tasks such as regression and performance testing.

c. **Black-Box Testing**:
   - Focuses on testing the application's functionality without knowledge of its internal code structure.

d. **Risk-Based Testing**:
   - Prioritizes testing areas critical to user experience and business functionality, such as payments and booking modules.

e. **Exploratory Testing**:
   - Testers explore the application to identify unexpected behavior and usability issues.

E. Key Tools and Technologies

a. **Testing Tools**: Selenium, Postman (for API testing).

b. **Security Tools**: Burp Suite.

1.1 Scope

A. In Scope

Functional Requirements to be Tested:

a. **User Authentication and Management**:
   - User registration for vehicle owners and area providers.
   - Login/logout functionality with secure password hashing.
   - User profile updates (email, phone number, password, etc.).

- Verification and validation of user inputs during registration and login.

b. **Area Providing Module**:

- Parking area registration with details (name, address, total spaces, price per hour, etc.).
- Editing, updating, and deleting parking area details.
- Real-time updates of available parking spaces.

c. **Area Booking Module**:

- Searching for parking areas based on geolocation and availability.
- Booking functionality with start and end times.
- Real-time updates to availability upon booking confirmation or cancellation.

d. **Payment Processing**:

- Integration with payment gateways for secure transactions.
- Payment methods: credit/debit cards, UPI, net banking.
- Payment status updates and refund mechanisms.

e. **Notification System**:

- Sending notifications for booking confirmations, cancellations, and payment updates.
- Marking notifications as read/unread.

f. **Reviews and Ratings**:

- Users can provide ratings and reviews for parking areas.
- Moderation and retrieval of reviews.

<u>Non-Functional Requirements to be Tested:</u>

a. **Performance**:

- Response time for booking searches and payment processing.
- Scalability to handle high user traffic during peak hours.

b. **Usability**:

- Intuitive user interface for both vehicle owners and area providers.

- Accessibility for diverse users, including responsive design for mobile devices.

c. **Security**:

- Data encryption for sensitive information (e.g., passwords, payment details).
- Protection against vulnerabilities like SQL injection, XSS, and CSRF.
- Secure API communication using HTTPS.

d. **Compatibility**:

- Cross-browser testing (e.g., Chrome, Firefox, Safari).
- Testing on multiple devices (Android, iOS, desktops).

e. **Reliability**:

- Ensuring system stability under various scenarios (e.g., high traffic, unexpected inputs).
- Database integrity with consistent updates to available spaces and user data.

f. **Maintainability**:

- Testing the modular structure for easy updates and fixes.
- Validation of automated test cases to facilitate future regression testing.

B. Out-of-Scope Testing

a. Hardware compatibility testing as the platform relies on third-party devices for access.

1.2 Quality Objective

a. **Conformance to Functional and Non-Functional Requirements**:
Ensure the application adheres to the specified functional and non-functional requirements, including core features such as user authentication, area booking, payment processing, and notifications.

b. **Delivering Quality as Defined by Stakeholders**:
Validate that the **Application Under Test (AUT)** meets the quality standards

and expectations set by the client. This includes functionality, usability, performance, and security benchmarks to provide a seamless user experience.

c. **Identifying and Resolving Bugs Before Deployment**:
Proactively identify documents and resolve defects or issues in the application to ensure it is stable and error-free before going live.

d. **User Satisfaction**:
Guarantee that the platform delivers a satisfying experience to both vehicle owners and area providers, with intuitive workflows, responsive performance, and secure transactions.

e. **System Reliability and Scalability**:
Confirm that the platform can handle peak loads, large user bases, and extensive interactions between its modules without performance degradation.

f. **Integration and Interoperability**:
Validate the seamless integration of various modules (e.g., user management, booking, payments) and their ability to operate cohesively across different devices, browsers, and environments.

g. **Data Integrity and Security**:
Ensure that sensitive user data is securely stored, transmitted, and protected from unauthorized access or breaches.

h. **Regulatory and Compliance Adherence**:
To build trust with users and area providers, verify compliance with relevant legal, financial, and data protection standards (e.g., GDPR, PCI DSS).

i. **Minimizing Risk**:
Mitigate risks associated with functionality failures, security breaches, or performance issues by conducting comprehensive testing.

j. **Readiness for Future Enhancements**:
Lay a foundation for the maintainability and scalability of the application to accommodate future updates or feature additions seamlessly.

1.3 Roles and Responsibilities

A. QA Analyst (Quality Assurance Analyst)

- **Role**: Ensures the quality of the application through rigorous testing.
- **Responsibilities**:
    - Create and execute test cases based on functional and non-functional requirements.

- Perform various types of testing such as functional, regression, usability, and performance testing.
- Log and track defects in the defect management tool.
- Validate defect fixes and perform retesting.
- Collaborate with developers to understand issues and resolve them efficiently.
- Prepare test summary reports and share findings with the team.

B. Developers

- **Role**: Build and maintain the application based on requirements and feedback.
- **Responsibilities**:
    - Develop features and functionalities as outlined in the project specifications.
    - Fix bugs identified during testing.
    - Perform unit testing and code reviews to ensure code quality.
    - Collaborate with QA analysts to understand and resolve defects.
    - Optimize the application for performance, security, and scalability.
    - Provide technical documentation for the developed modules.

C. UI/UX Designer

- **Role**: Designs the user interface and user experience to ensure a seamless user journey.
- **Responsibilities**:
    - Create wireframes, mockups, and prototypes based on requirements.
    - Ensure the design aligns with the target audience's preferences.
    - Collaborate with developers to ensure the design is implemented correctly.
    - Conduct usability testing and make adjustments based on feedback.
    - Maintain consistency in design elements throughout the application.

D. Database Administrator (DBA*)*

- **Role**: Manages the database to ensure data integrity, security, and performance.
- **Responsibilities**:

- Design and implement the database schema.
- Optimize database queries for performance.
- Ensure data backup and recovery mechanisms are in place.
- Monitor database health and resolve any issues.
- Enforce security measures to protect sensitive data.

2 Test Methodology

2.1 Overview

Selected Methodology: Agile

Agile methodology was chosen for the Project project due to the following factors:

a. Nature of the Project

- **Dynamic Requirements**: The requirements for Project are expected to evolve based on user feedback, emerging trends, and stakeholder inputs. Agile allows for flexible adjustments during development.
- **Module-Based Approach**: The project is divided into modules (e.g., User Management, Area Booking, Payments, Notifications), which can be developed and tested incrementally.

b. Collaboration Needs

- **Cross-Functional Teams**: Agile promotes collaboration between developers, QA analysts, business analysts, and other stakeholders, which is essential for this multifaceted project.

c. Focus on Quality

- **Continuous Testing**: Agile integrates testing throughout the development lifecycle, ensuring that bugs are identified and addressed early.
- **User-Centric Development**: By incorporating user feedback in every sprint, Agile ensures the application meets functional and non-functional requirements.

d. Risk Mitigation

- **Frequent Releases**: Regular releases reduce the risk of delivering a product that doesn't meet expectations.

- **Early Issue Identification**: With iterative development and testing, potential risks and issues are identified early, minimizing delays and cost overruns.

e. Project Complexity

- **Integration of Multiple Modules**: Agile supports the seamless integration of complex, interdependent modules like geolocation, payment systems, and notifications.
- **High Scalability**: Agile can accommodate the growing scope of features and functionalities as the project evolves.

f. Comparison with Other Methodologies

- **Waterfall**: Waterfall is rigid and doesn't accommodate changes well. For a dynamic project like Project, this would lead to delays and increased costs.
- **Iterative**: While iterative allows some flexibility, it lacks the collaboration and frequent feedback loops provided by Agile.
- **Extreme Programming (XP)**: XP's focus on engineering practices might not align with the broader business and client collaboration needs of the project.

2.2 Test Levels

a. Unit Testing

- **Purpose**: To validate individual components or modules of the application in isolation.
- **Scope**:
  - Ensure that methods, functions, and classes for modules like User Management, Area Booking, and Payment Processing work as intended.
  - Test critical functions such as user authentication, vehicle registration, and booking creation.
- **Responsible Team**: Developers and QA
- **Tools Used**: JUnit (for Java-based testing).
- **Example**:
  - Verify that the validateUserCredentials () method correctly authenticates users.

    - Test if the calculateTotalCost () function calculates the booking cost accurately.

b. Integration Testing

- **Purpose**: To test interactions between modules and ensure seamless data flow.
- **Scope**:
    - Verify integrations between modules like:
        - User Management ↔ Area Booking.
        - Booking ↔ Payment.
        - Notifications ↔ Booking and Payments.
    - Ensure APIs and database queries are working correctly when modules communicate.
- **Responsible Team**: QA
- **Tools Used**: Postman (API Testing), Selenium (for end-to-end flows).
- **Example**:
    - Check if a successful booking triggers a notification.
    - Test payment success updates the booking status to "Confirmed."

c. Acceptance Testing

- **Purpose**: To validate the application meets client expectations and is ready for deployment.
- **Scope**:
    - Conduct User Acceptance Testing (UAT) with client and end-users.
    - Validate real-world scenarios, such as a user booking a parking spot at a specific location using their registered vehicle.
- **Responsible Team**: QA and End-User Representatives.
- **Tools Used**: Manual Testing, or tools such as BrowserStack for cross-platform compatibility testing.
- **Example**:

    - Validate if the area owner can see booking and payment details for their parking area.

d. Regression Testing

- **Purpose**: To ensure new features or changes do not negatively affect existing functionality.
- **Scope**:
    - Rerun test cases for critical modules (e.g., User Management, Payment Processing) after introducing new features or fixes.
- **Responsible Team**: QA
- **Tools Used**: Selenium (for automation).
- **Example**:
    - Verify that adding a new payment method does not disrupt the booking workflow.

2.3 Test Completeness

a. Test Coverage

- **Requirement**:
    - Achieve 100% coverage for all critical modules and workflows.
    - All functional and non-functional requirements should be covered by test cases.
    - Verify edge cases, boundary conditions, and real-world scenarios.
- **Verification**:
    - Review test coverage reports to ensure every functionality and integration is tested

b. Execution of Test Cases

- **Requirement**:
    - Execute all manual and automated test cases planned during test design.
    - Ensure that tests for major workflows, such as user registration, booking, payment, and notification, are successfully executed.

- **Verification**:
  - Review the execution status in the test management tool (e.g., Jira, TestRail).
  - Confirm that test cases marked as critical and high priority are executed without errors.

c. Bug Resolution

- **Requirement**:
  - All critical and high-priority bugs identified during testing are resolved.
  - Medium and low-priority bugs should either be resolved or deferred with client approval for future releases.
- **Verification**:
  - No open critical bugs remain.
  - Medium and low-priority issues are documented and scheduled for subsequent releases if not fixed.

d. Performance Benchmarks

- **Requirement**:
  - Meet all performance benchmarks, including response time, load capacity, and scalability.
  - Confirm that the application performs efficiently under peak load conditions.
- **Verification**:
  - Review performance testing reports from tools like JMeter to ensure all benchmarks are met.

e. Security Validation

- **Requirement**:
  - Address all identified vulnerabilities from penetration testing and security audits.
  - Ensure data encryption, secure payment processing, and proper access controls.
- **Verification**:

- Review security test results and confirm no critical vulnerabilities remain unaddressed.

f. **User Acceptance Testing (UAT)**

- **Requirement**:
  - UAT feedback from the client and end-users is positive.
  - All reported issues during UAT are resolved or documented for future consideration.
- **Verification**:
  - Validate that UAT sign-off is obtained from stakeholders.

g. **Regression Testing**

- **Requirement**:
  - Complete regression testing to ensure that new changes have not affected existing functionalities.
- **Verification**:
  - Review the regression testing logs and confirm that no major issues were introduced during recent updates.

h. **Documentation Completion**

- **Requirement**:
  - All test artifacts, including test cases, defect reports, and test execution reports, are documented and shared with stakeholders.
  - User manuals and training documents are updated based on the final application.
- **Verification**:
  - Perform a final review of test documentation to ensure completeness and accuracy.

A. Test cases:

1. User Authentication Module

Test Case ID	Scenario	Input	Expected Output
UA-001	Login with valid	Valid	Success: User logged in

	credentials	username/password	
UA-002	Login with invalid credentials	Invalid username/password	Error: "Invalid credentials"
UA-003	Login with empty fields	Empty username/password	Error: "Fields cannot be empty"
UA-004	Password length boundary (min)	Password = 6 characters	Success: Logged in
UA-005	Password length boundary (max)	Password = 20 characters	Success: Logged in
UA-006	Account lock after 5 failed attempts	5 invalid login attempts	Error: "Account locked for security reasons"

2. Area Management Module

Test Case ID	Scenario	Input	Expected Output
AM-001	Add new area with valid details	Valid area details	Success: Area added
AM-002	Add new area with missing fields	Missing address/total spaces	Error: "All fields are required"
AM-003	Update area details (price per hour)	Valid price	Success: Details updated
AM-004	Update area with invalid price	Negative price	Error: "Invalid price value"
AM-005	Delete area with active bookings	Area with bookings	Error: "Cannot delete area with active bookings"

3. Booking Module

Test Case ID	Scenario	Input	Expected Output
BM-001	Book a parking space (valid)	Valid user, area, vehicle	Success: Booking created
BM-002	Book a space with invalid area ID	Invalid area ID	Error: "Area not found"
BM-003	Book space when no slots are available	Area with 0 available slots	Error: "No slots available"
BM-004	Cancel a booking (valid)	Booking ID	Success: Booking canceled
BM-005	Cancel a booking (expired)	Booking with past end time	Error: "Cannot cancel expired booking"

4. Payment Module

Test Case ID	Scenario	Input	Expected Output
PM-001	Process payment (valid)	Valid booking, amount	Success: Payment confirmed
PM-002	Process payment (failed gateway)	Valid details, gateway fails	Error: "Payment failed. Try again."
PM-003	Payment with partial amount	Amount < Total Cost	Error: "Insufficient payment amount"
PM-004	Payment refund on cancellation	Valid cancellation	Success: Amount refunded
PM-005	Payment using coupon (valid)	Valid coupon applied	Success: Discount applied, payment processed

---

5. Notifications Module

Test Case ID	Scenario	Input	Expected Output
NT-001	Send booking confirmation notification	Valid booking	Notification sent to user
NT-002	Send notification on failed payment	Payment fails	Notification sent with retry instructions
NT-003	Notify area owner on low rating	Area rating drops below 3	Notification sent to owner
NT-004	Notification status update (read)	User reads notification	Status updated to "Read"

6. Reviews and Ratings Module

Test Case ID	Scenario	Input	Expected Output
RR-001	Add review with valid details	Valid area, rating, review text	Success: Review added
RR-002	Add review without rating	Missing rating	Error: "Rating is required"
RR-003	Add review with invalid rating	Rating > 5 or < 1	Error: "Rating out of range"
RR-004	Duplicate review submission	User submits a second review	Error: "Review already submitted"

7. Reports and Logs Module

Test Case ID	Scenario	Input	Expected Output

RL-001	Generate report for bookings	Valid date range	Success: Report generated
RL-002	Generate report for payments	Valid filters (date, status)	Success: Report generated
RL-003	Log login attempts	Failed login	Entry added to system logs
RL-004	Log payment gateway response	Payment processed	Gateway response logged

8. Geolocation Module

Test Case ID	Scenario	Input	Expected Output
GL-001	Search parking near valid location	Latitude, Longitude	Success: List of parking areas returned
GL-002	Search parking with invalid location	Out-of-bounds coordinates	Error: "Location not supported"
GL-003	Calculate distance from user to area	Valid coordinates	Success: Distance calculated

B. Equivalence Testing:

1. User Authentication

Test Case ID	Input Field	Equivalence Classes	Expected Result
EP-001	Username	Valid: [1–50 chars], Invalid: [0, >50]	Valid: Accept; Invalid: Reject
EP-002	Password	Valid: [8–20 chars], Invalid: [<8, >20]	Valid: Accept; Invalid: Reject

2. Area Availability

Test Case ID	Input Field	Equivalence Classes	Expected Result
EP-003	Area Availability	Valid: Available slots > 0	Show available areas
EP-004		Invalid: Available slots = 0	Show "No slots available"

3. Payment

Test Case ID	Input Field	Equivalence Classes	Expected Result
EP-005	Card Number	Valid: 16 digits	Valid: Accept; Invalid: Reject
EP-006	Amount	Valid: [1–5000 INR], Invalid: [<1, >5000]	Valid: Accept; Invalid: Reject

C. Decision Table:

1. Decision Table for Complex Module Interactions

Condition	C1	C2	C3	C4	Outcome
Booking status is "Confirmed"	Yes	Yes	No	No	Notify user
Payment processed successfully	Yes	No	Yes	No	Update booking status to "Paid"
Area availability updated	Yes	Yes	Yes	No	Notify interested users
Expected Outcome	Notify	Deny	Update	Log Error	

2. Booking Cancellation and Refund

Condition	C1	C2	C3	C4	Action
Booking status is "Booked"	Yes	Yes	No	No	Process cancellation
Cancellation request within 1 hour	Yes	No	Yes	No	Refund full amount
Refund request outside 1 hour	No	Yes	No	Yes	Partial refund
Expected Outcome	Refund	No Refund	No Action	No Action	

3. Parking Availability Update

Condition	C1	C2	C3	C4	Action
Total spaces > 0	Yes	Yes	No	No	Allow booking
Available spaces > 0	Yes	No	Yes	No	Allow booking
Booking cancelled	No	Yes	No	Yes	Increase available spaces
Expected Outcome	Allow	Deny	Deny	Update	

3 Resource & Environment Needs

3.1 Testing Tools

a. Automation Tools

- **Purpose**: To automate repetitive test scenarios for regression, performance, and functional testing.
- **Tools**:
  - **Selenium**: For functional and regression testing of the web application.
  - **Appium**: For testing mobile versions of the application.
  - **Postman**: For API testing and validation.
  - **JMeter**: For performance and load testing.

b. Collaboration Tools

- **Purpose**: To facilitate communication and collaboration among team members.
- **Tools**:
  - **Slack**: For team communication.
  - **Confluence**: For maintaining centralized documentation and knowledge sharing.

c. Version Control Tool

- **Purpose**: To track changes to test scripts, automation code, and documentation.
- **Tool**:
  - **Git**: For version control and collaboration.
  - **GitHub/GitLab**: For hosting repositories.

d. Reporting Tools

- **Purpose**: To generate and share test execution and defect tracking reports.
- **Tools**:

- **Excel/Google Sheets**: For simple reporting needs.

3.2 Test Environment

A. Hardware Requirements

a. Test Machines (Desktops/Laptops)

- Processor: Intel i5 or higher / AMD Ryzen 5 or higher
- RAM: Minimum 8 GB (16 GB recommended for automation and performance testing)
- Storage: Minimum 256 GB SSD (512 GB SSD or higher recommended for faster performance)
- Screen Resolution: 1920x1080 (Full HD) or higher
- Network: Stable broadband connection with minimum 50 Mbps speed
- Graphics: Integrated graphics (dedicated GPU if load testing requires graphical simulations)

b. Mobile Devices

- Android: Devices running Android 8 (Oreo) or higher
- Screen Sizes: A mix of devices covering small, medium, and large screen sizes for responsive testing

c. Servers (for Environment Hosting)

- CPU: 4-core or higher
- RAM: 16 GB or higher
- Disk Space: Minimum 1 TB (for database, logs, and backups)
- Network: High-speed Ethernet connection for local servers

B. Software Requirements

a. Operating System

- Windows 8 and above (recommended: Windows 10 or 11)
- macOS (for compatibility testing)
- Linux distributions (for backend testing and server environment)

b. Office Suite

- Microsoft Office 2013 and above
- Alternative: LibreOffice or Google Workspace

c. Email and Collaboration

- Microsoft Exchange for email management
- Slack, Microsoft Teams, or Zoom for team collaboration

d. Browsers (for Cross-Browser Testing)

- Chrome: Latest version
- Firefox: Latest version
- Microsoft Edge: Latest version
- Safari: Latest version (for macOS and iOS devices)

e. Test and Development Tools

- IDE: IntelliJ IDEA or Visual Studio Code
- Database: MySQL Workbench for database management
- Automation: Selenium WebDriver, Postman, JMeter

4 Terms/Acronyms

Category	Term/Acronym	Description
General Terms	AUT (Application Under Test)	Refers to the Project application being tested.
	QA (Quality Assurance)	Activities and processes to ensure the quality of the product.
	UAT (User Acceptance Testing)	Testing conducted by end-users to ensure the application meets their requirements.
	SDLC (Software Development Life Cycle)	The process of planning, creating, testing, and deploying software.
	STLC (Software Testing Life Cycle)	The process of testing the software, including requirement analysis, test planning, test execution, and reporting.
	SRS (Software Requirements Specification)	A document detailing the functional and non-functional requirements of the application.
	BRS (Business	A document describing the high-level business

	Requirements Specification)	requirements and objectives.
Testing Terms	Test Case	A set of actions executed to verify a specific functionality of the AUT.
	Test Suite	A collection of test cases intended to test a system or its components.
	Regression Testing	Testing existing functionalities to ensure no new bugs are introduced after code changes.
	Boundary Value Analysis (BVA)	A testing technique that focuses on the values at the edge of input domains.
	Equivalence Partitioning (EP)	A technique that divides inputs into equivalent groups to reduce the number of test cases.
	Defect/Bug	A flaw in the application causing incorrect or unexpected results.
	Defect Density	The number of defects identified in a specific module or application size.
	Smoke Testing	A preliminary test to ensure the basic functionality of the application works.
	Sanity Testing	A narrow and deep test to verify specific functionality after minor changes.
Tools and Techniques	Selenium	A tool for browser-based automation testing.
	Appium	An open-source tool for mobile application testing.
	JMeter	A tool used for performance testing.
	Postman	A tool for API testing.
	MySQL Workbench	A tool for managing and testing the database.
Project-Specific Terms	Vehicle Owner	A user who books parking spaces in the Project application.
	Area Owner	A user who provides parking spaces to be listed in the Project application.
	Booking Module	The system handling the reservation of parking spaces.
	Area Module	The system managing parking space details like availability and pricing.
	Notifications Module	The system responsible for sending alerts and updates to users.
	Payment Gateway	The integrated system for processing online payments.
Acronyms	API (Application Programming Interface)	A set of functions allowing applications to interact with other software.
	UI (User Interface)	The space where user interactions with the application occur.
	UX (User Experience)	The overall experience of a user when interacting

		with the application.
	KPIs (Key Performance Indicators)	Metrics used to measure the effectiveness of the application or testing process.
	DBMS (Database Management System)	Software for creating, managing, and interacting with databases.