Parking Slot Management System - Development Phase 2 Report

---

a. Requirements

System Overview:

The Parking Slot Management System is designed to facilitate parking slot booking, management, and reporting. It includes authentication, role-based access control, booking features, and automated ticketing.

Functional Requirements:

User Authentication

- Users can register, log in, and log out.
- JWT-based authentication for secure access.
- Role-based access control (Admin, Moderator, User).
- Zod validation is implemented.

Parking Slot Management

- Admins can create, update, and delete parking slots.
- Users can view available slots and book them.
- Admins and moderators can manage slot statuses (Available, Occupied, Reserved).

Booking Management

- Users can create, view, and delete their bookings.
- Admins and moderators can view all bookings.
- Booking statuses: Pending, Confirmed, Cancelled.

User Management

- Admins can view, update, and delete users.
- Admins can update user roles (User, Moderator, Admin).

Multi-Section Management

- Admins can create, update, and delete parking lots, levels, and slots.

Slot Availability Checking

- Users can view available slots by selecting date, lot, level, and slot.
- Collision Avoidance System: Users cannot book unavailable slots.

Overstay Fine Calculation

- Users can view overstay fines.
- Admins and moderators can manage overstay statuses (Mark Complete, Fine Paid).

Parking History & Reports

- Users can view/download booking reports.
- Admins and moderators can view all bookings and download reports.
- Admins and moderators can view system statistics (Total Lots, Floors, Slots, Active Bookings, Monthly Revenue).

User Feedback & Ratings

- Users, Admins, and Moderators can view ratings.
- Users can rate and review completed bookings.

Automated Ticketing System

- Users, Admins, and Moderators can view and download booking tickets.

Security Requirements

- CORS middleware for cross-origin requests.
- Secure cookie settings for authentication tokens.
- Input validation using express-validator.

Database Requirements

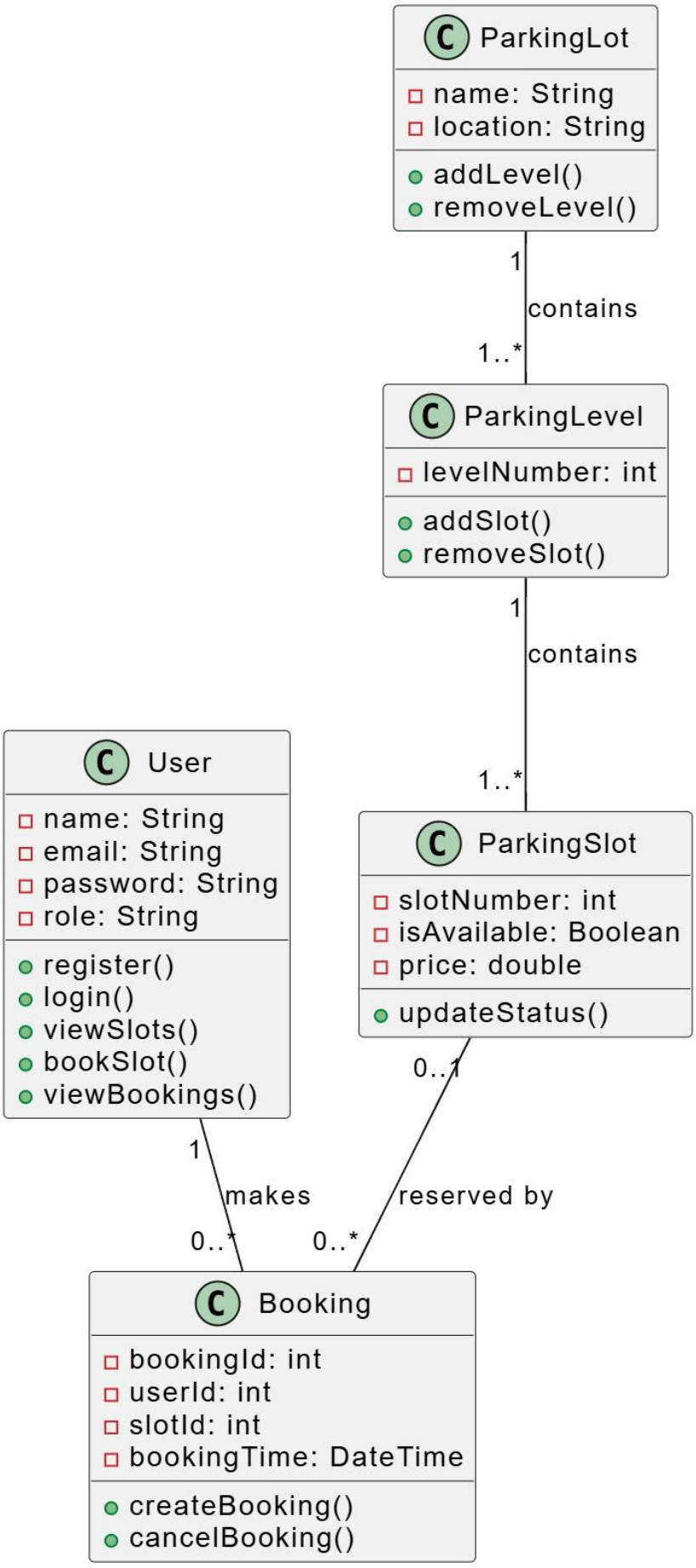
- MongoDB used as the primary database.
- Models: User, Parking, Booking with appropriate validations and relationships.

---

b. UML Design

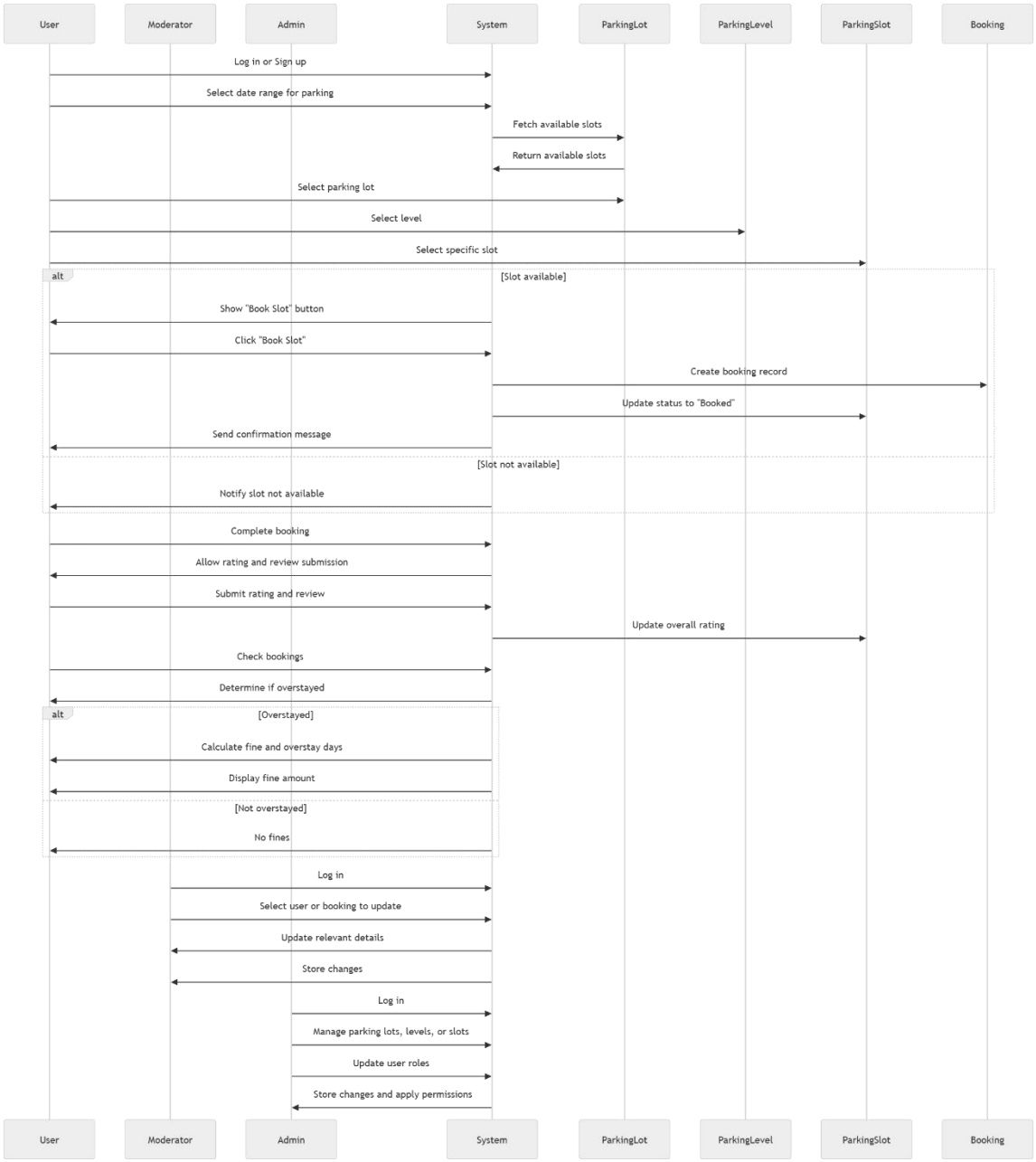
Class Diagram

- Represents entities like User, ParkingLot, ParkingLevel, ParkingSlot, and Booking.

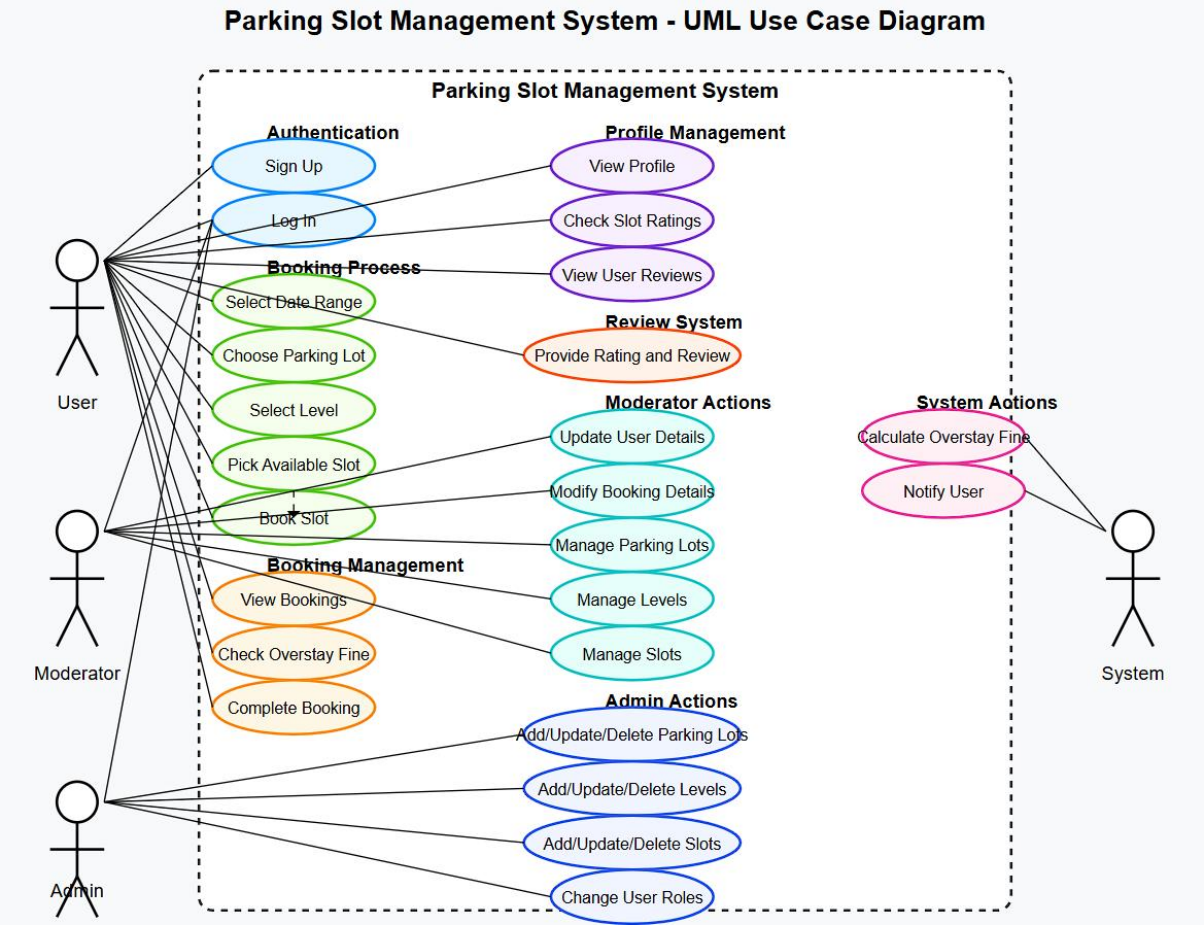

ParkingLot
name: String
location: String
addLevel()
removeLevel()
1
contains
1..*
ParkingLevel
levelNumber: int
addSlot()
removeSlot()
1
contains
1..*
User
name: String
email: String
password: String
role: String
register()
login()
viewSlots()
bookSlot()
viewBookings()
ParkingSlot
slotNumber: int
isAvailable: Boolean
price: double
updateStatus()
0..1
1
makes
0..*
reserved by
0..*
Booking
bookingId: int
userId: int
slotId: int
bookingTime: DateTime
createBooking()
cancelBooking()

Sequence Diagram

- Includes interactions for booking a parking slot and handling an overstay fine.



Use Case Diagram



- One normal case (successful booking)
- One error case (booking a slot that is already occupied)

---

c. Test Cases (Unit Tests)

Test Case 1: User Registration

Description: Tests if a new user can register successfully. **Inputs:** Username, Email, Password. **Expected Output:** Registration success message.

Request URL: http://localhost:5001/api/auth/signup

Request Method: POST

payload: {"firstName":"user04","lastName":"
04","email":"user4@gmail.com","password":"Aq1234","confirmPassword":"Aq1234","vehicle":"Od5
566","contact":"1234567890"}

Response: {

"success": true,

"user": {

"email": "user4@gmail.com",

"password": "$2b$12$UnLsELtCZ0r4oe2nAiZFhen6NjL4emu8.2I4ucfGLBZW09HjowkQu",

"firstName": "user04",

"lastName": " 04",

"role": "user",

"contact": "1234567890",

"verified": "false",

"vehicle": "Od5566",

"_id": "67ea7ea0601c2072097fe90e",

"createdAt": "2025-03-31T11:38:08.133Z",

"updatedAt": "2025-03-31T11:38:08.133Z",

"__v": 0,

"id": "67ea7ea0601c2072097fe90e"

},

"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3ZWE3ZWEwNjAxYzIwNzIwOTdmZTkwZSIsImVtYWlsIjoidXNlcjRAZ21haWwuY29tIiwicm9sZSI6InVzZXIiLCJpYXQiOjE3NDM0MjEwODgsImV4cCI6MTc0MzUwNzQ4OH0.MEGaBT2YgOy7rzIS-MJh9yxb4hWvyxB9ES3n3Lv9fM4",

"message": "Successfully Registered and Logged In!"

}

Test Case 2: Booking a Parking Slot

Description: Ensures users can book available slots. **Inputs:** Date, Parking Lot, Level, Slot. **Expected Output:** Booking confirmation message.

Request URL: http://localhost:5001/api/parking/slots/67e985a6c097fb65b3cbe763/book

Request Method: POST

Payload: {"fromDate":"2025-03-31T11:43:21.685Z","toDate":"2025-04-01T11:43:21.685Z","vehicleNumber":"Od2222"}

Response: {

"message": "Booking successful",

"booking": {

"parkingSlot": "67e985a6c097fb65b3cbe763",

"bookedBy": "67ea7ea0601c2072097fe90e",

"vehicleNumber": "Od2222",

```
    "fromDate": "2025-03-31T11:43:21.685Z",
    "toDate": "2025-04-01T11:43:21.685Z",
    "status": "Confirmed",
    "fineAmount": 0,
    "isFinePaid": false,
    "overstayDays": 0,
    "dailyRate": 300,
    "_id": "67ea8090601c2072097fe935",
    "createdAt": "2025-03-31T11:46:24.833Z",
    "updatedAt": "2025-03-31T11:46:24.834Z",
    "__v": 0
  },
  "receipt": {
    "slotNumber": "Zone001",
    "totalDays": 1,
    "dailyRate": 300,
    "totalAmount": 300
  }
}
```


Test Case 3: View Bookings

Description: Checks if the user gets booking data correctly. **Inputs:** Booking id, **Expected Output:** Display booking.

Request URL:http://localhost:5001/api/bookings/user/67ea7ea0601c2072097fe90e

Request Method: GET

Authorization: Bearer {Token}

Response: [


```
  {
    "_id": "67ea8090601c2072097fe935",
    "parkingSlot": {
      "_id": "67e985a6c097fb65b3cbe763",
      "slotNumber": "Zone001",
```

```
      "parkingLevel": {
        "_id": "67e98546c097fb65b3cbe759",
        "name": "Floor003",
        "parkingLot": {
          "_id": "67e9819b827293941821134a",
          "name": "Parking 2",
          "address": "Demo "
        }
      },
      "status": "Available",
      "price": 300
    },
    "bookedBy": {
      "_id": "67ea7ea0601c2072097fe90e",
      "email": "user4@gmail.com",
      "firstName": "user04"
    },
    "vehicleNumber": "Od2222",
    "fromDate": "2025-03-31T11:43:21.685Z",
    "toDate": "2025-04-01T11:43:21.685Z",
    "status": "Confirmed",
    "fineAmount": 0,
    "isFinePaid": false,
    "overstayDays": 0,
    "dailyRate": 300,
    "createdAt": "2025-03-31T11:46:24.833Z",
    "updatedAt": "2025-03-31T11:46:24.834Z",
    "__v": 0
  }
]
```

Test Case 4: Create Parking Slot

Description: Checks if the Admin parking spot correctly. **Inputs:** lot, level, from date. to date, price, fineamount, **Expected Output:** Display parking spot.

Request URL:http://localhost:5001/api/parking/slots

Request Method:POST

Authorization: Bearer {Token}

Payload:{"slotNumber":"Z06","parkingLevel":"67e98546c097fb65b3cbe759","location":"","price":"400","fineAmount":"500"}

Response: {

"slotNumber": "Z06",

"parkingLevel": "67e98546c097fb65b3cbe759",

"status": "Available",

"averageRating": 0,

"price": 400,

"fineAmount": 500,

"_id": "67ea84cf96e56c493420418e",

"reviews": [],

"createdAt": "2025-03-31T12:04:31.399Z",

"__v": 0

}

---

d. User Manual

Login & Registration

1. Navigate to the login page.
2. Enter your email and password.
3. Click on "Login" to access the system.
4. New users can click on "Register" and fill in the form.

Booking a Parking Slot

1. Log in to your account.
2. Navigate to "Find a Slot."
3. Select date, parking lot, level, and slot.
4. Click "Book Now."

Viewing Parking History

1. Navigate to "My Bookings."
2. Download reports as needed.

Admin Panel Management

1. Log in as an Admin.
2. Navigate to "Parking Management" to add/edit slots.
3. Check user bookings and generate reports.

---

e. Compilation & Execution Instructions

Prerequisites

- Node.js installed
- MongoDB database configured

Running the Project

1. Clone the repository.
2. Install dependencies using npm install.
3. Start the backend server with npm run dev.
4. Start the frontend with npm run start.

Running Test Cases

- Execute unit tests with npm test.

---

f. Code Inspection Feedback

Feedback Received:

- Improve error handling for booking failures.
- Implement more detailed logging for slot management.

Actions Taken:

- Added enhanced error messages.
- Implemented logging for slot status changes.

---

g. Reflection

What Went Well

- Successfully implemented user authentication and role-based access.

- Booking management system is fully functional.
- Automated ticketing system and reporting features work as expected.

Areas for Improvement

- UI can be enhanced for a better user experience.
- Consider adding payment integration for online slot reservations.
- Implement notifications for booking confirmations and fines.

---

h. Members Contribution Table

Team Member	Responsibilities	Overall Contribution (%)
Jaswanth	- Set up the entire backend folder structure and integrated MongoDB with Mongoose for schema-based design. - Configured Express middleware and routing for modular API development. - Built the statistics controller to analyze slot usage, revenue, and user activity. - Developed token verification middleware using JWT for securing APIs. - Created cronjobs to automatically check for overstays and calculate fines in real-time. - Implemented `index.js` to initialize the server with middleware, routes, and DB connection. - Designed and implemented the Parking Slot model for managing individual slot status.	12%
Jaya	- Developed authentication modules including Sign In, Sign Up, and Logout APIs with secure password hashing and JWT. - Added frontend validations by using Zod to enhance security and UX for Sign In/Sign Up forms. - Built the Parking Lot model to define metadata like name, location, total levels.	11%

	- Implemented reusable UI components like the Navbar and Sidebar. - Created utility request handler `requestMethod.js` to abstract HTTP logic. - Initialized the React frontend using `main.jsx` and managed route-level setup. - Designed and connected Booking and Parking routes to backend services.	
Anurag	- Created the User model with role-based schema (admin, moderator, user). - Developed controller functions for fetching, updating, and deleting user data. - Configured routes to enable secure access to user-related APIs. - Collaborated on Parking Controller functions like slot updates, availability, and status changes. - Built Booking Controller to manage user bookings, cancellations, and queries.	11%
Abhishek	- Developed the Parking model defining structure for each parking space in the lot. - Built the Parking Level model to define each level with number of slots and identifiers. - Implemented backend Parking Controllers to manage CRUD operations across lots, levels, and slots.	11%
Deepika	- Designed and implemented the Booking model including details like user, slot, timings, payment, and status. - Developed Booking Controller to handle bookings, overstay tracking, and updates.	11%
Sashank	- Set up the entire frontend dashboard layout, user profile page, and routing base. - Developed interactive dialogs for slot addition, booking	11%

	confirmation, editing details, and review submission. - Ensured UI consistency and responsive behavior across components.	
Diresh	- Implemented Redux store with slices for auth, bookings, parking, and user data. - Developed logic for Sign In/Sign Up state management. - Integrated components like Home, UserSection, and PaymentSuccessModal with Redux and API services. - Handled all API interactions via centralized functions in `apicalls.js`. - Built and configured the main store in `store.js`.	11%
Nikhil	- Developed UserSection to allow users to manage profiles, bookings, and view history. - Contributed to Parking Management UI and integration with backend APIs.	11%
Jishna	- Built Booking Management module for viewing, filtering, and administrating user bookings. - Helped implement Parking Management functionalities including slot-level control and fine management.	11%