Final Project

Kihoon-Car-Rent-Application

PROG3271- Open Source Web Programming -Sec1

Jerome Ramsay

Kihoon Kim

Computer Programming

8781294

Oct 15, 2025

0. Project Repository and Jira Ticketing

- GitHub Reposity:
  https://github.com/kkim8781294/kihoon-car-rent-application.git
- Jira Board:
  https://kkim1294.atlassian.net/jira/software/projects/CAR/boards/36?atlOrigin=eyJpIjoiMjQyNDc3NDM0OTZiNGFiM2I4MDFiODZmMWE3MTZjM2YiLCJwIjoiaiJ9

1. Introduction

This project implements a backend system for a car rental booking website. It follows the Clean Architecture design pattern and demonstrates key backend development concepts such as authentication, booking logic, data persistence, and administrative control. The system is built using Node.js, Express, TypeScript, and MongoDB, and achieves over 80% test coverage using Jest.

2. Objective

- Design and implement a backend using Clean Architecture principles.
- Apply authentication, authorization, and RESTful design.
- Use JWT for secure access control.
- Logging all pipeline
- Ensure modular, testable components and ≥80% Jest coverage.
- Validate all endpoints with Bruno API test suite.
- Manage development progress with Jira and Git version control

3. System Overview

3.1 Project Type

- Option2 Booking Website: Car Rental Backend

3.2 Core Functionalities

Module	Description
Auth & Management	Registration, login, token generate/refresh

Cars	Admin car creation and public car listing
Booking	Car availability check and booking creation registered users and guests
Multiuser booking	Allow multiple users or guests to make bookings on the same car or site
Admin	Booking listing, approval, decline, and editing
Testing	Bruno API test and Jest based unit-test (80%+coverage)

4. System Architecture

4.1 Clean Architecture Layers

Layer	Description	Example Directory
Core	Domain entities and business logic	src/core
Infrastructure	Database models and repositories	src/infrastructure
Presentation	Controllers and rotues	src/controller src/port
Middleware	Authentication and request validation	src/middleware
Testing	Unit Test	src/tests

4.2 Flow of Execution

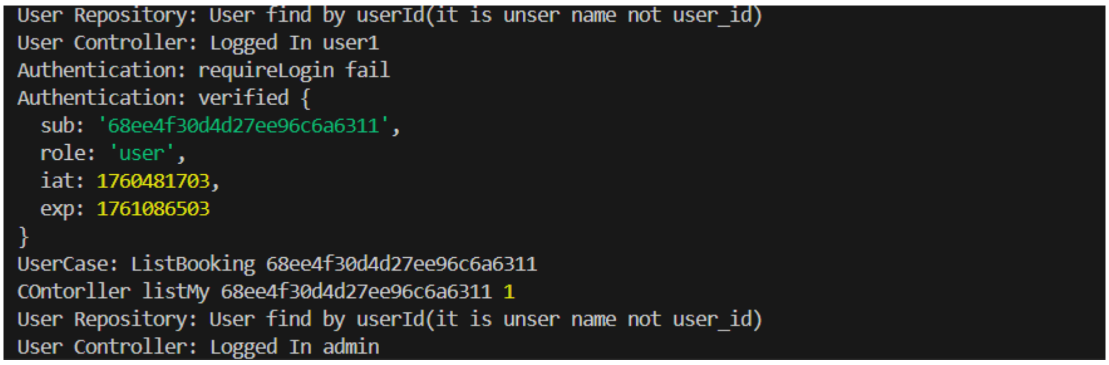
- Request → Route → Controller → Usecase → Repository → MongoDB

4.3 Technologies Used

- Architecture: Clean Architecture (Core 4)
- Backend Framework: Node.js, Express
- Language: TypeScript
- Database: MongoDB
- Authentication: JWT
- Testing: Bruno, Jest
- Project Management: Jira, Git

4.4 Logging

- Logging was implemented throughout all layers to trace both successful operations and failure events.
- Each use case, controller, and repository logs key events using console.log, providing clear visibility into request flow and system behavior
- Example

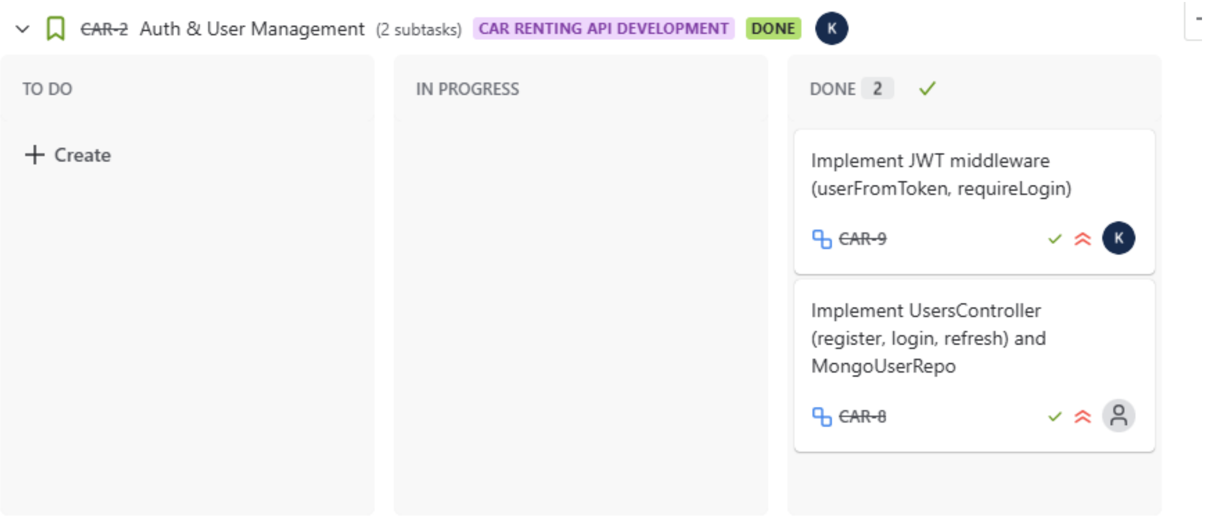


```
User Repository: User find by userId(it is unser name not user_id)
User Controller: Logged In user1
Authentication: requireLogin fail
Authentication: verified {
  sub: '68ee4f30d4d27ee96c6a6311',
  role: 'user',
  iat: 1760481703,
  exp: 1761086503
}
UserCase: ListBooking 68ee4f30d4d27ee96c6a6311
COntorller listMy 68ee4f30d4d27ee96c6a6311 1
User Repository: User find by userId(it is unser name not user_id)
User Controller: Logged In admin
```

5. Implementation Details

5.1 Auth & User management

- **Jira Story: CAR-2 – Auth & User Management**



~~CAR-2~~ Auth & User Management (2 subtasks) CAR RENTING API DEVELOPMENT DONE K

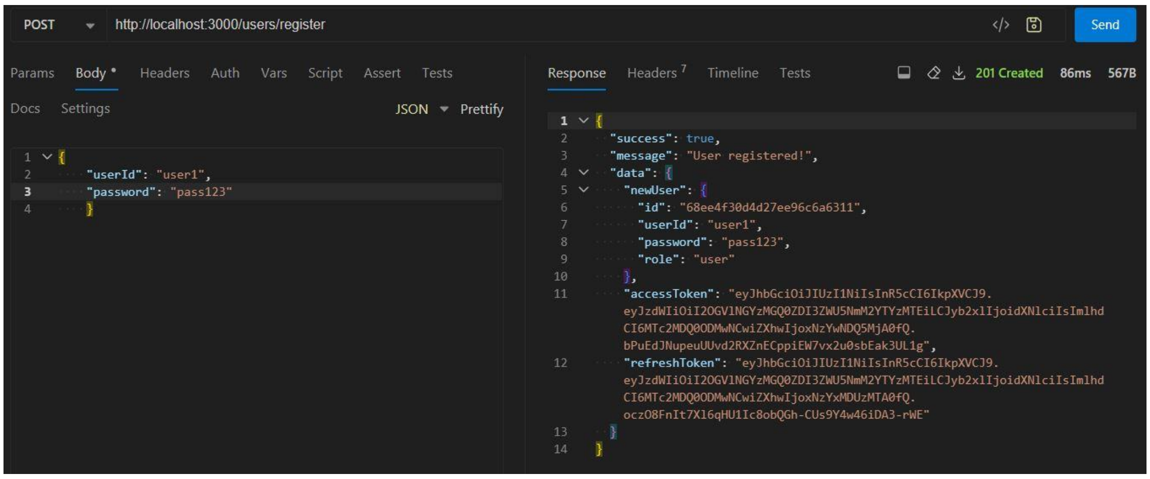
TO DO	IN PROGRESS	DONE 2 ✓
+ Create		Implement JWT middleware (userFromToken, requireLogin) ~~CAR-9~~
		Implement UsersController (register, login, refresh) and MongoUserRepo ~~CAR-8~~

- **Jira Subtask**
  - CAR-8: Implement UsersController (register, login, refresh) and MongoUserRepo
  - CAR-9: Implement JWT middleware (userFromToken, requireLogin)

- **Implementation Summary**
    - Implemented UsersController with endpoints /register, /login and JWT-based authentication and middleware functions (userFromToken, requireLogin) manage access control. MongoUserRepo handles persistence and data retrieval.
- **Bruno Test Results**

Endpoint	Method	Description
/user/register	POST	Registers a new user
/user/login	POST	Logs in existing user
/user/seed-admin	POST	Seed admin and make admin auth token

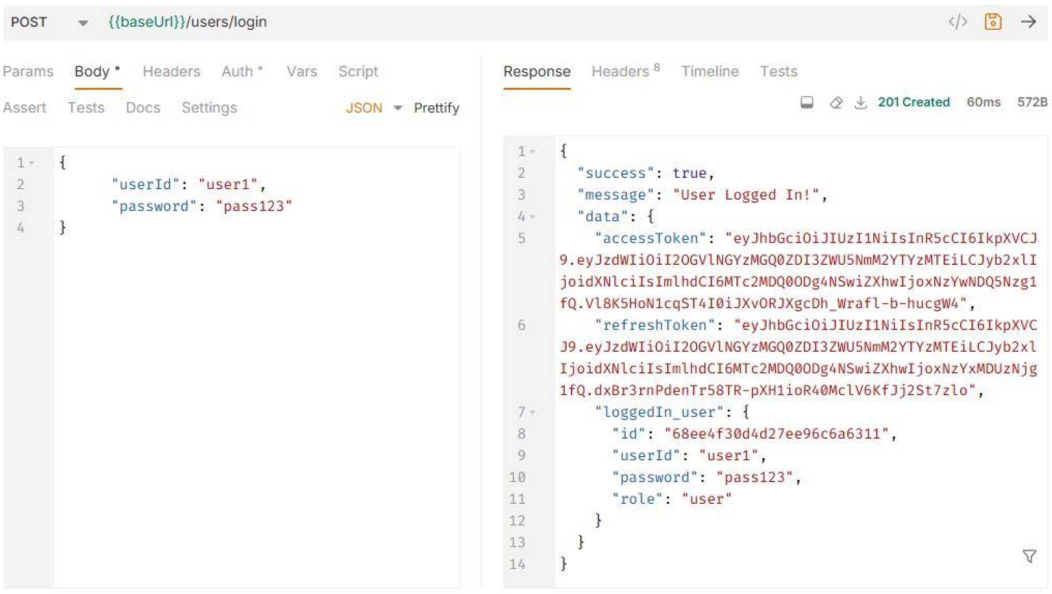
    - POST: /user/register


POST
http://localhost:3000/users/register
Send
Params Body • Headers Auth Vars Script Assert Tests
Docs Settings
JSON Prettify
Response Headers 7 Timeline Tests
201 Created 86ms 567B

```
{
    "userId": "user1",
    "password": "pass123"
}
```

```
{
  "success": true,
  "message": "User registered!",
  "data": {
    "newUser": {
      "id": "68ee4f30d4d27ee96c6a6311",
      "userId": "user1",
      "password": "pass123",
      "role": "user"
    },
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiI2OGVlNGYzMGQ0ZDI3ZWU5NmM2YTYzMTEiLCJyb2xlIjoidXNlciIsImlhd
    CI6MTc2MDQ0ODMwNCwiZXhwIjoxNzYwNDQ5MjA0fQ.
    bPuEdJNupeuUUvd2RXZnECppiEW7vx2u0sbEak3UL1g",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiI2OGVlNGYzMGQ0ZDI3ZWU5NmM2YTYzMTEiLCJyb2xlIjoidXNlciIsImlhd
    CI6MTc2MDQ0ODMwNCwiZXhwIjoxNzYxMDUzMTA0fQ.
    ocz08FnIt7Xl6qHU1Ic8obQGh-CUs9Y4w46iDA3-rWE"
  }
}
```

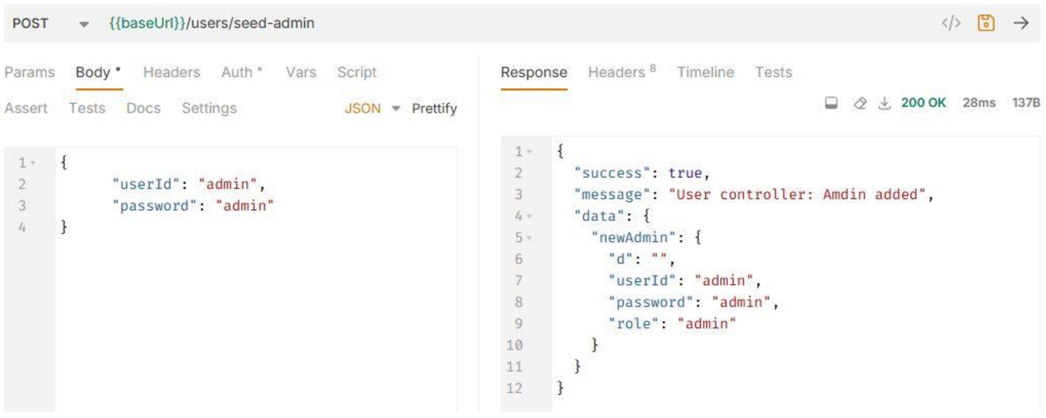
- POST:/user/login



```
{
      "userId": "user1",
      "password": "pass123"
}
```

```
{
  "success": true,
  "message": "User Logged In!",
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ
9.eyJzdWIiOiI2OGVlNGYzMGQ0ZDI3ZWU5NmM2YTYzMTEiLCJyb2xlI
joidXNlciIsImlhdCI6MTc2MDQ0ODg4NSwiZXhwIjoxNzYwNDQ5Nzg1
fQ.Vl8K5HoN1cqST4I0iJXvORJXgcDh_Wrafl-b-hucgW4",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC
J9.eyJzdWIiOiI2OGVlNGYzMGQ0ZDI3ZWU5NmM2YTYzMTEiLCJyb2xl
IjoidXNlciIsImlhdCI6MTc2MDQ0ODg4NSwiZXhwIjoxNzYxMDUzNjg
1fQ.dxBr3rnPdenTr58TR-pXH1ioR40MclV6KfJj2St7zlo",
    "loggedIn_user": {
      "id": "68ee4f30d4d27ee96c6a6311",
      "userId": "user1",
      "password": "pass123",
      "role": "user"
    }
  }
}
```

- POST:/user/seed-admin

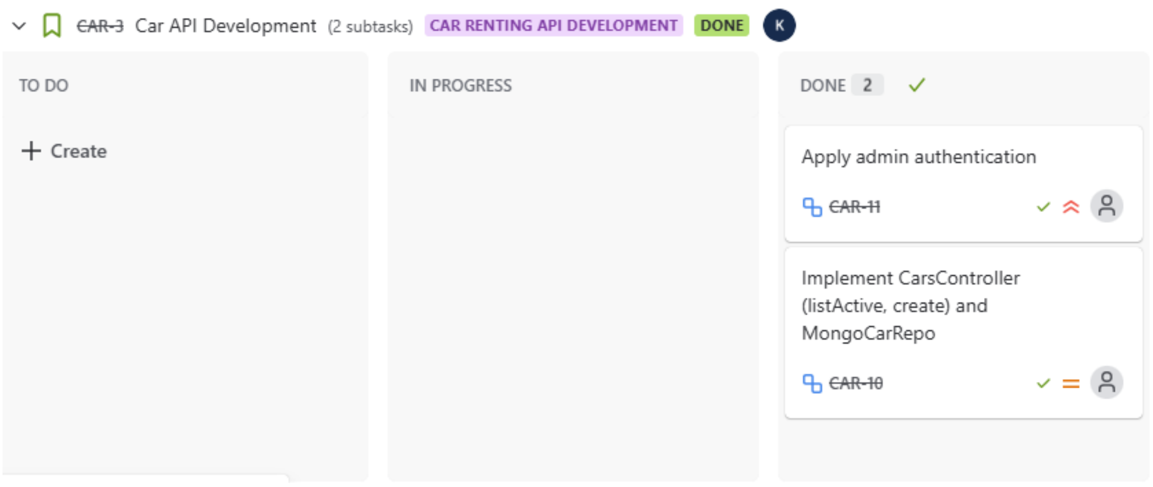


```
{
      "userId": "admin",
      "password": "admin"
}
```

```
{
  "success": true,
  "message": "User controller: Amdin added",
  "data": {
    "newAdmin": {
      "d": "",
      "userId": "admin",
      "password": "admin",
      "role": "admin"
    }
  }
}
```

5.2 Cars API

- **Jira Story: CAR-3- Car API Development**



~~CAR-3~~ Car API Development (2 subtasks) CAR RENTING API DEVELOPMENT DONE K

TO DO

+ Create

IN PROGRESS

DONE 2 ✓

Apply admin authentication

~~CAR-11~~

Implement CarsController (listActive, create) and MongoCarRepo

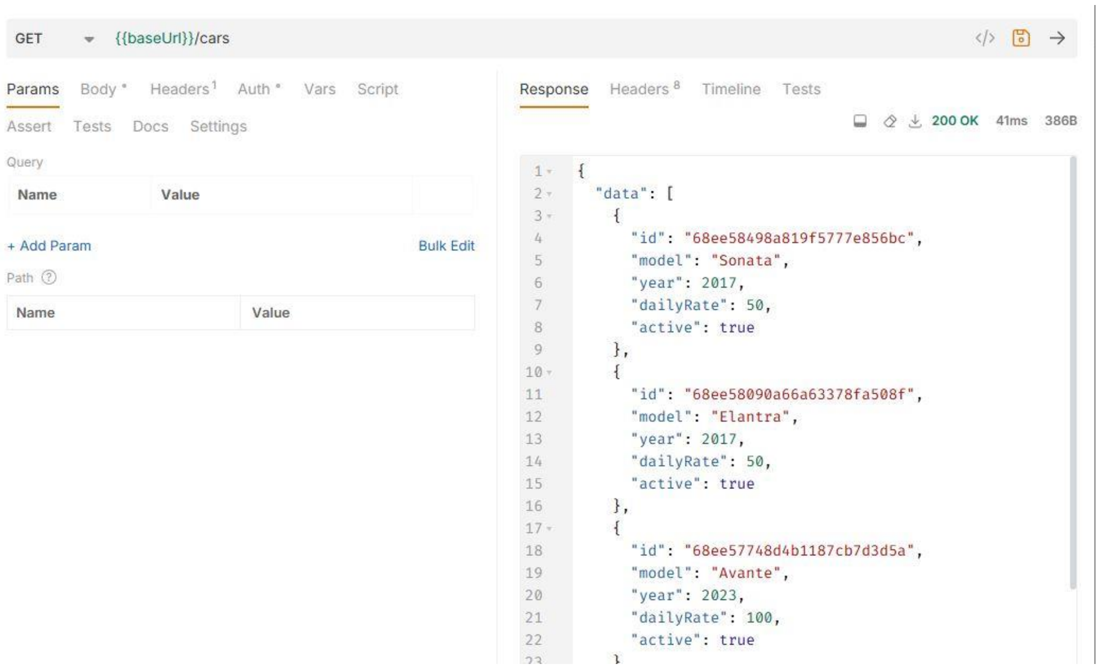
~~CAR-10~~

- **Jira Sub-tasks**
    - CAR-10: Implement CarsController (listActive, create) and MongoCarRepo
    - CAR-11: Apply admin authentication
- **Implementation Summary**
    - Implemented CarsController for listing and creating cars including returning active car and allowing only admin users to add new cars and been Applied requireAdmin middleware for access restriction and MongoCarRepo for persistence

- **Bruno Test Result**

Endpoint	Method	Description
/cars	POST	Retrieve all active cars
/cars	POST(admin)	Create new car(executed successfully with admin and non-admin tokens)

- GET: /cars – list active car



GET {{baseUrl}}/cars

Params Body Headers 1 Auth Vars Script Assert Tests Docs Settings

Query

Name	Value

+ Add Param Bulk Edit

Path

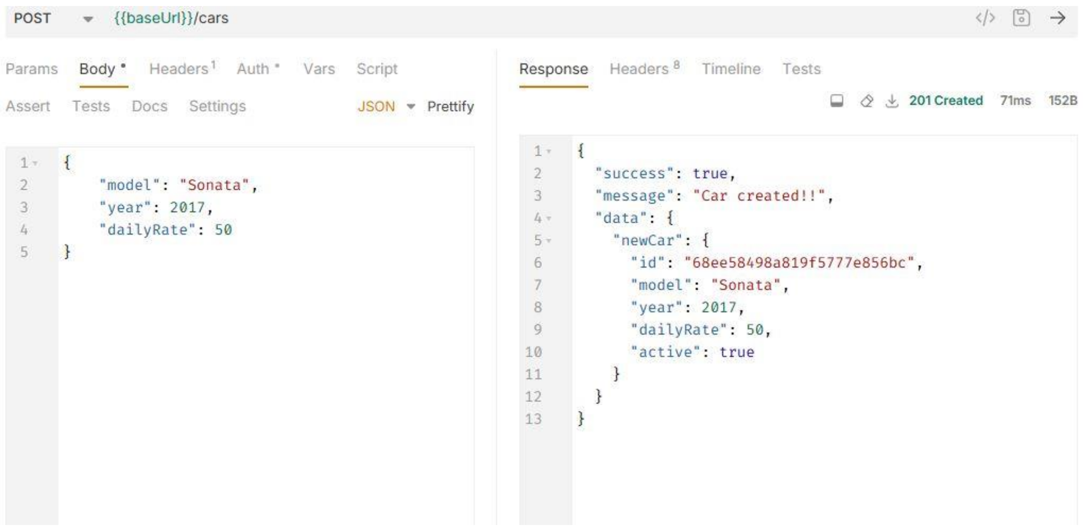
Name	Value

Response Headers 8 Timeline Tests

200 OK 41ms 386B

```
{
  "data": [
    {
      "id": "68ee58498a819f5777e856bc",
      "model": "Sonata",
      "year": 2017,
      "dailyRate": 50,
      "active": true
    },
    {
      "id": "68ee58090a66a63378fa508f",
      "model": "Elantra",
      "year": 2017,
      "dailyRate": 50,
      "active": true
    },
    {
      "id": "68ee57748d4b1187cb7d3d5a",
      "model": "Avante",
      "year": 2023,
      "dailyRate": 100,
      "active": true
    }
```

- POST:/cars – create car (admin only)



POST {{baseUrl}}/cars

Params Body Headers 1 Auth Vars Script Assert Tests Docs Settings JSON Prettify

```
{
    "model": "Sonata",
    "year": 2017,
    "dailyRate": 50
}
```

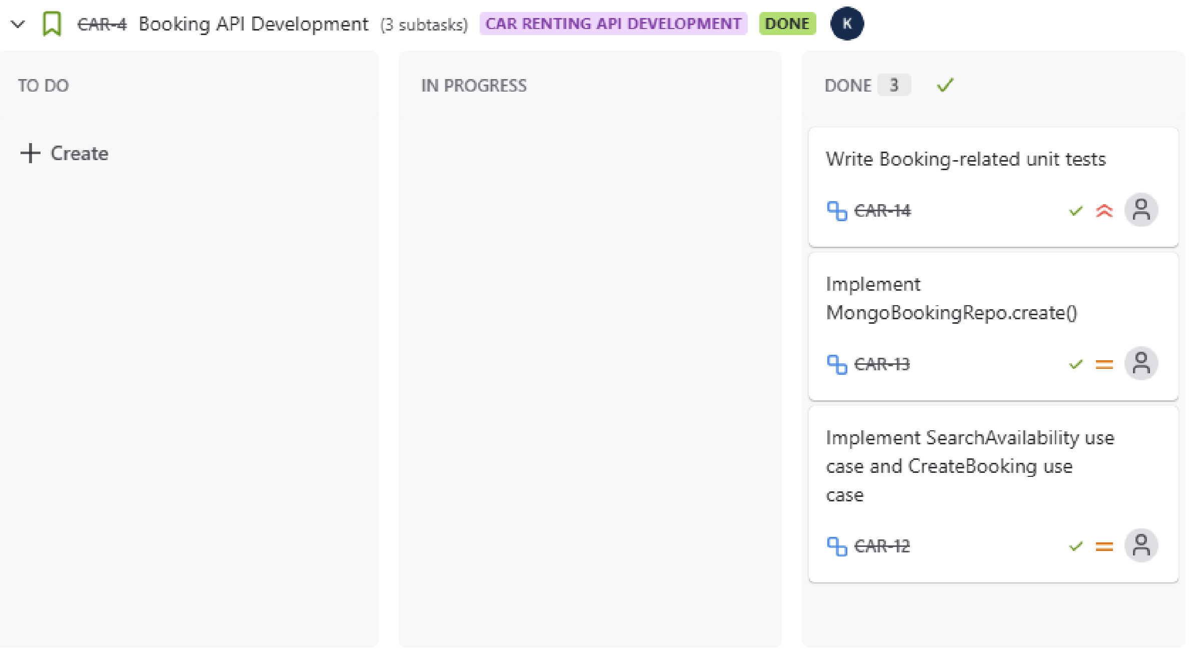
Response Headers 8 Timeline Tests

201 Created 71ms 152B

```
{
  "success": true,
  "message": "Car created!!",
  "data": {
    "newCar": {
      "id": "68ee58498a819f5777e856bc",
      "model": "Sonata",
      "year": 2017,
      "dailyRate": 50,
      "active": true
    }
  }
}
```

5.3 Bookings API

- **Jira Story: CAR-4- Car API Development**



~~CAR-4~~ Booking API Development (3 subtasks) CAR RENTING API DEVELOPMENT DONE K

TO DO

+ Create

IN PROGRESS

DONE 3 ✓

Write Booking-related unit tests
~~CAR-14~~

Implement MongoBookingRepo.create()
~~CAR-13~~

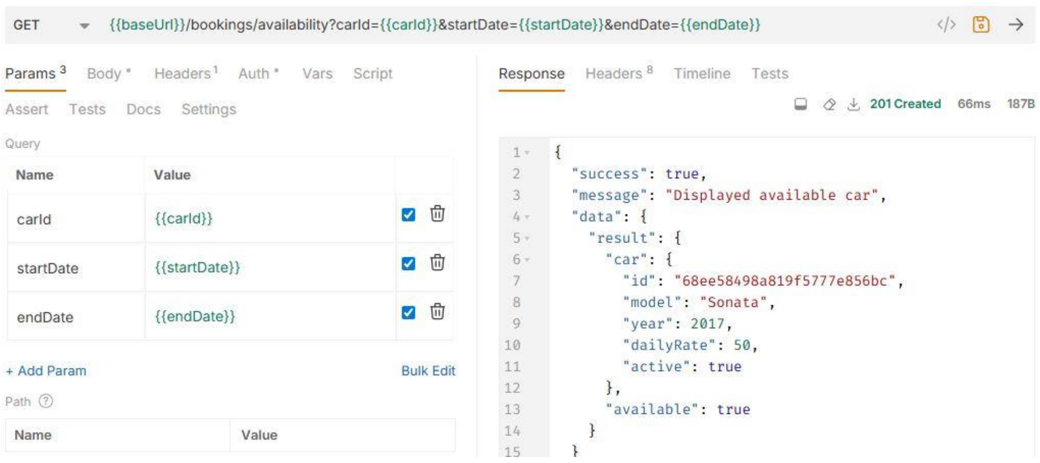
Implement SearchAvailability use case and CreateBooking use case
~~CAR-12~~

- **Jira Sub-tasks**
  - CAR-12: Implement SearchAvailability use case and CreateBooking use case
  - CAR-13: Implement MongoBookingRepo.create()
  - CAR-14: Write Booking-related unit tests
- **Implementation Summary**
  - Check availably car based on date
  - Booking for both registered users (authenticated bookings) and guests (email-based bookings).
    - For guest users, guestEmail is required; for registered users, their userId is automatically attached from the JWT payload.
  - The system also supports multiple users booking different cars simultaneously,
    - while preventing double booking of the same car for overlapping dates using date-fns interval checks.
  - The total cost is computed as dailyRate × number_of_days.

- **Bruno Test Result**

Endpoint	Method	Description
/bookings/availability	GET	Check car availability
/bookings	POST	Guest booking with email and authenticated user booking
/bookings (overlap)	POST	Overlapping booking prevented
/bookings/my	GET	Return only user's own booking

  - GET: /bookings/availability



```
GET {{baseUrl}}/bookings/availability?carId={{carId}}&startDate={{startDate}}&endDate={{endDate}}
```

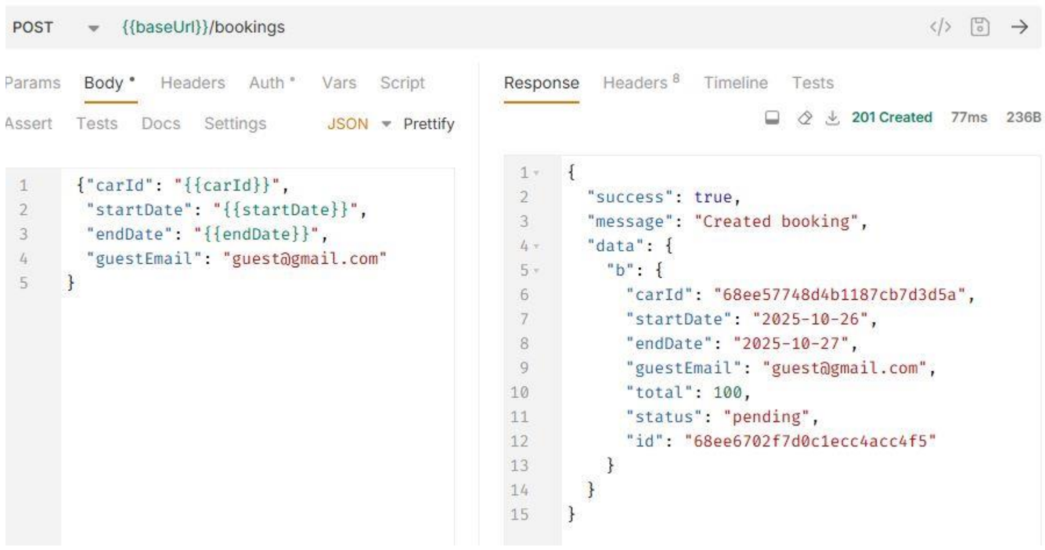
Query

Name	Value
carId	{{carId}}
startDate	{{startDate}}
endDate	{{endDate}}

Response: 201 Created 66ms 187B

```
{
  "success": true,
  "message": "Displayed available car",
  "data": {
    "result": {
      "car": {
        "id": "68ee58498a819f5777e856bc",
        "model": "Sonata",
        "year": 2017,
        "dailyRate": 50,
        "active": true
      },
      "available": true
    }
  }
```

  - POST: /bookings (for guest)



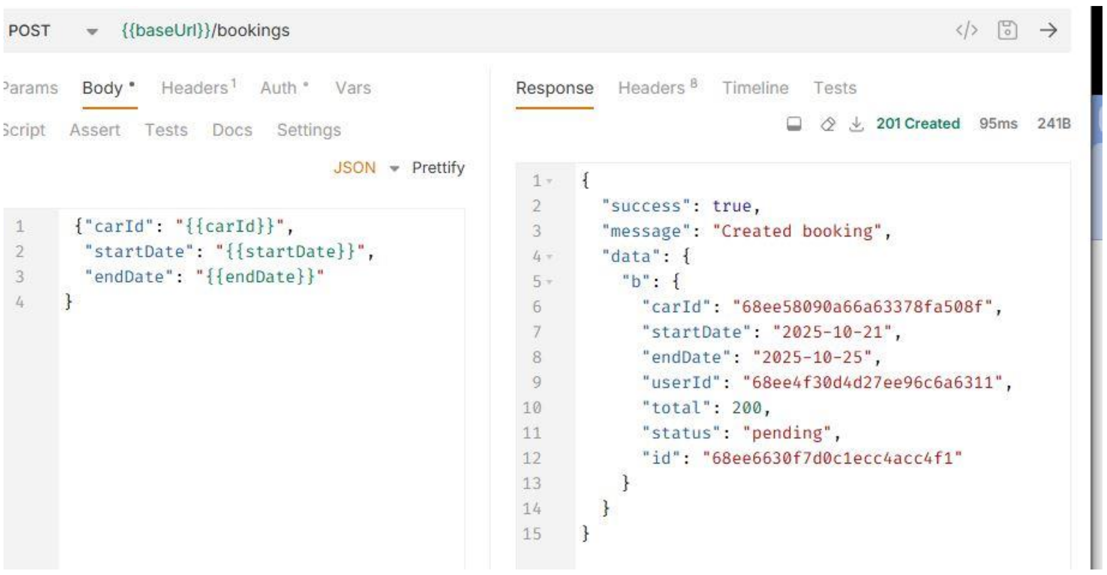
```
POST {{baseUrl}}/bookings
```

```
{"carId": "{{carId}}",
 "startDate": "{{startDate}}",
 "endDate": "{{endDate}}",
 "guestEmail": "guest@gmail.com"
}
```

Response: 201 Created 77ms 236B

```
{
  "success": true,
  "message": "Created booking",
  "data": {
    "b": {
      "carId": "68ee57748d4b1187cb7d3d5a",
      "startDate": "2025-10-26",
      "endDate": "2025-10-27",
      "guestEmail": "guest@gmail.com",
      "total": 100,
      "status": "pending",
      "id": "68ee6702f7d0c1ecc4acc4f5"
    }
  }
}
```

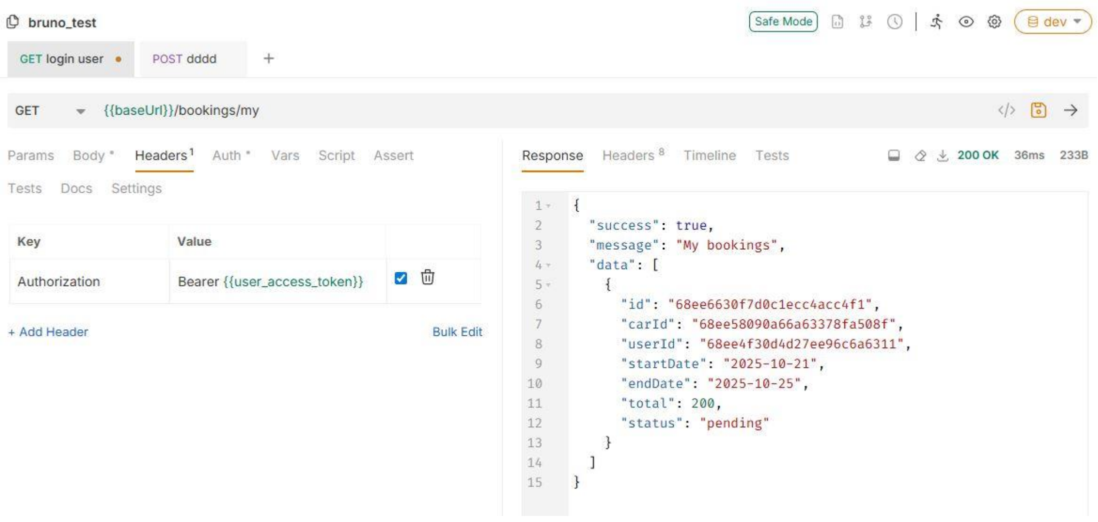
- POST: /bookings (for user)


POST {{baseUrl}}/bookings

```
{"carId": "{{carId}}",
  "startDate": "{{startDate}}",
  "endDate": "{{endDate}}"
}
```

```
{
  "success": true,
  "message": "Created booking",
  "data": {
    "b": {
      "carId": "68ee58090a66a63378fa508f",
      "startDate": "2025-10-21",
      "endDate": "2025-10-25",
      "userId": "68ee4f30d4d27ee96c6a6311",
      "total": 200,
      "status": "pending",
      "id": "68ee6630f7d0c1ecc4acc4f1"
    }
  }
}
```

- GET: /bookings/my

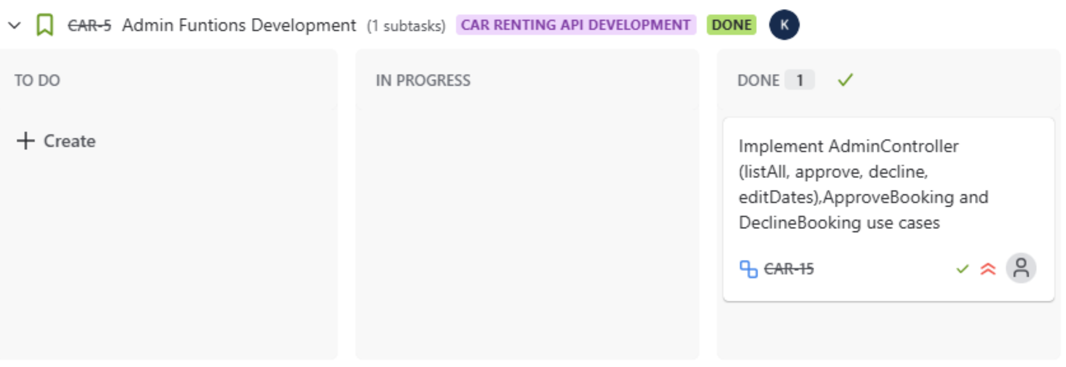

GET {{baseUrl}}/bookings/my

Key	Value
Authorization	Bearer {{user_access_token}}

```
{
  "success": true,
  "message": "My bookings",
  "data": [
    {
      "id": "68ee6630f7d0c1ecc4acc4f1",
      "carId": "68ee58090a66a63378fa508f",
      "userId": "68ee4f30d4d27ee96c6a6311",
      "startDate": "2025-10-21",
      "endDate": "2025-10-25",
      "total": 200,
      "status": "pending"
    }
  ]
}
```

5.4 Admin Functions

- **Jira Story: CAR-5- Admin Functions Development**


~~CAR-5~~ Admin Funtions Development (1 subtasks) CAR RENTING API DEVELOPMENT DONE

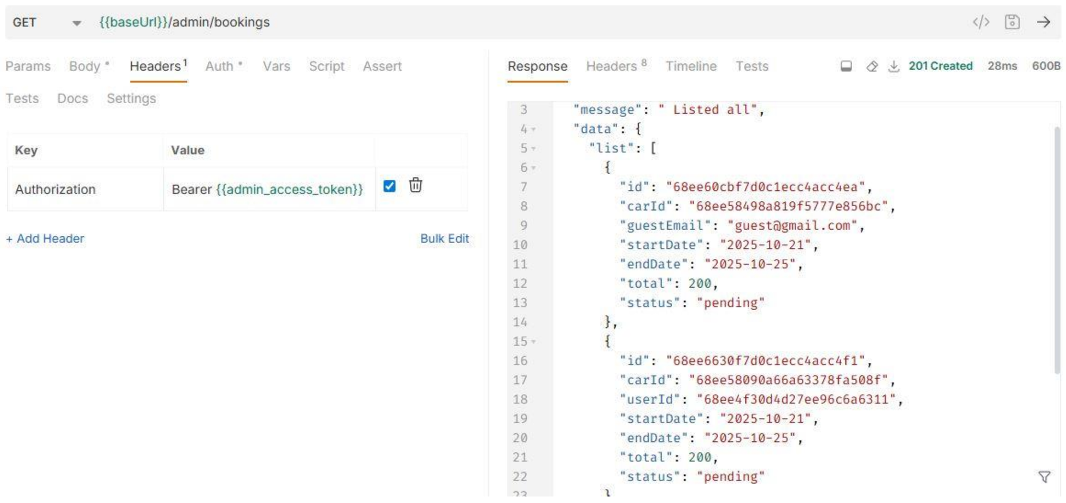
Implement AdminController (listAll, approve, decline, editDates),ApproveBooking and DeclineBooking use cases

~~CAR-15~~

- **Jira Sub-tasks**
    - CAR-15: Implement AdminController (listAll, approve, decline, editDates),ApproveBooking and DeclineBooking use cases
- **Implementation Summary**
    - All functions are restricted exclusively to administrators.
        - List all bookings
        - Approve booking
        - Decline booking
        - Edit booking
    - Authorization is enforced using the requireAdmin middleware, which validates the user's JWT role before allowing the request.
- **Bruno Test Result**

Endpoint	Method	Description
/admin/bookings	GET	Retrieve all bookings
/admin/bookings/:id/approve	PATCH	Approve booking
/admin/bookings/:id/decline	PATCH	Decline booking
/admin/bookings/:id/edit	PATCH	Edit booking

- GET: /admin/bookings



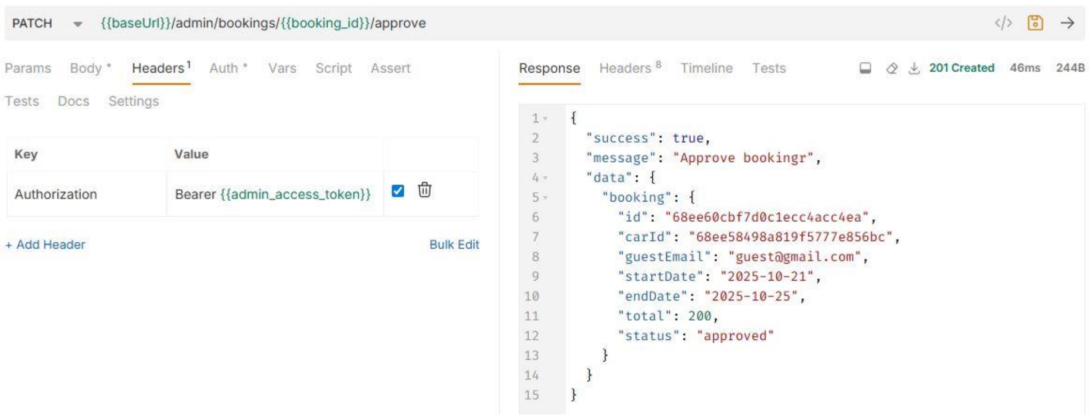
GET {{baseUrl}}/admin/bookings

Key	Value
Authorization	Bearer {{admin_access_token}}

201 Created 28ms 600B

```
  "message": " Listed all",
  "data": {
    "list": [
      {
        "id": "68ee60cbf7d0c1ecc4acc4ea",
        "carId": "68ee58498a819f5777e856bc",
        "guestEmail": "guest@gmail.com",
        "startDate": "2025-10-21",
        "endDate": "2025-10-25",
        "total": 200,
        "status": "pending"
      },
      {
        "id": "68ee6630f7d0c1ecc4acc4f1",
        "carId": "68ee58090a66a63378fa508f",
        "userId": "68ee4f30d4d27ee96c6a6311",
        "startDate": "2025-10-21",
        "endDate": "2025-10-25",
        "total": 200,
        "status": "pending"
```

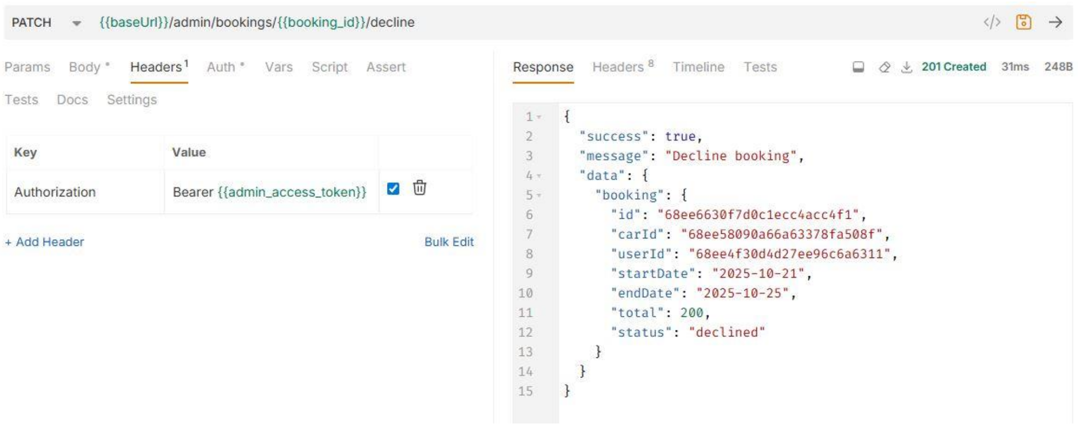
- PATCH: /admin/bookings/:id/approve



Key	Value
Authorization	Bearer {{admin_access_token}}

```
{
  "success": true,
  "message": "Approve bookingr",
  "data": {
    "booking": {
      "id": "68ee60cbf7d0c1ecc4acc4ea",
      "carId": "68ee58498a819f5777e856bc",
      "guestEmail": "guest@gmail.com",
      "startDate": "2025-10-21",
      "endDate": "2025-10-25",
      "total": 200,
      "status": "approved"
    }
  }
}
```

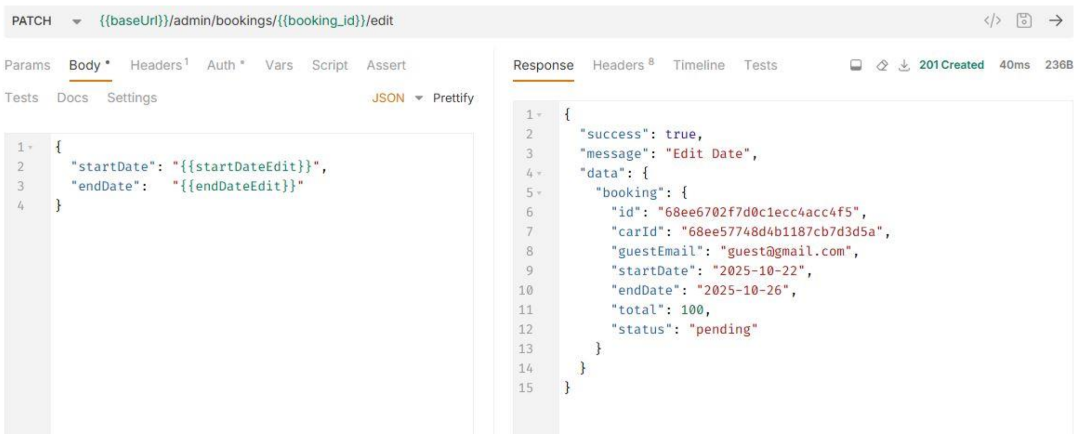
- PATCH: /admin/bookings/:id/decline



Key	Value
Authorization	Bearer {{admin_access_token}}

```
{
  "success": true,
  "message": "Decline booking",
  "data": {
    "booking": {
      "id": "68ee6630f7d0c1ecc4acc4f1",
      "carId": "68ee58090a66a63378fa508f",
      "userId": "68ee4f30d4d27ee96c6a6311",
      "startDate": "2025-10-21",
      "endDate": "2025-10-25",
      "total": 200,
      "status": "declined"
    }
  }
}
```

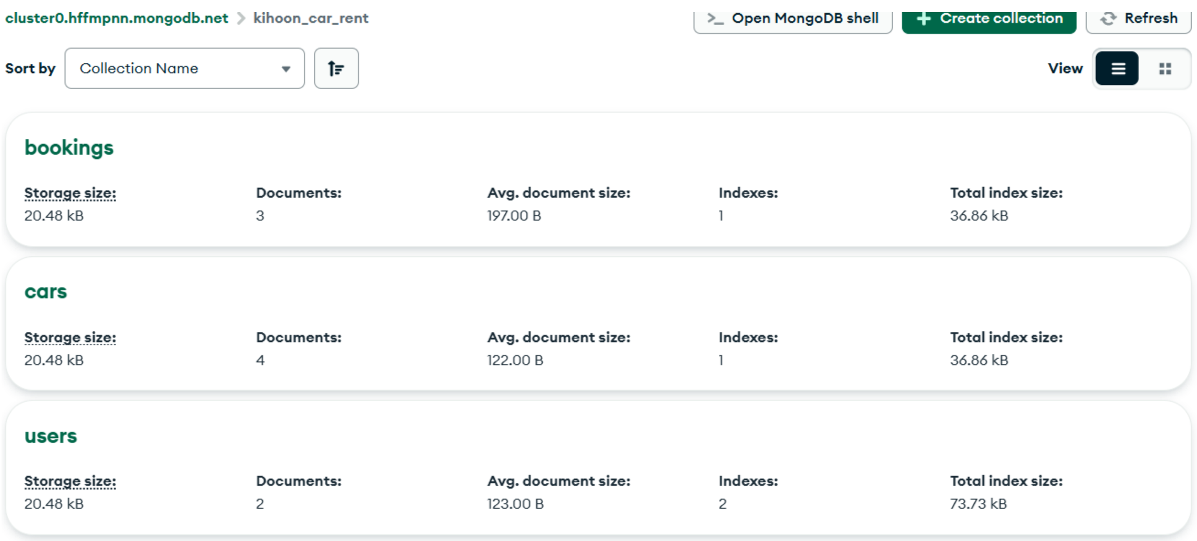
- PATCH: /admin/bookings/:id/edit



```
{
  "startDate": "{{startDateEdit}}",
  "endDate":   "{{endDateEdit}}"
}
```

```
{
  "success": true,
  "message": "Edit Date",
  "data": {
    "booking": {
      "id": "68ee6702f7d0c1ecc4acc4f5",
      "carId": "68ee57748d4b1187cb7d3d5a",
      "guestEmail": "guest@gmail.com",
      "startDate": "2025-10-22",
      "endDate": "2025-10-26",
      "total": 100,
      "status": "pending"
    }
  }
}
```

5.5 Database Structure



bookings

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	3	197.00 B	1	36.86 kB

cars

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	4	122.00 B	1	36.86 kB

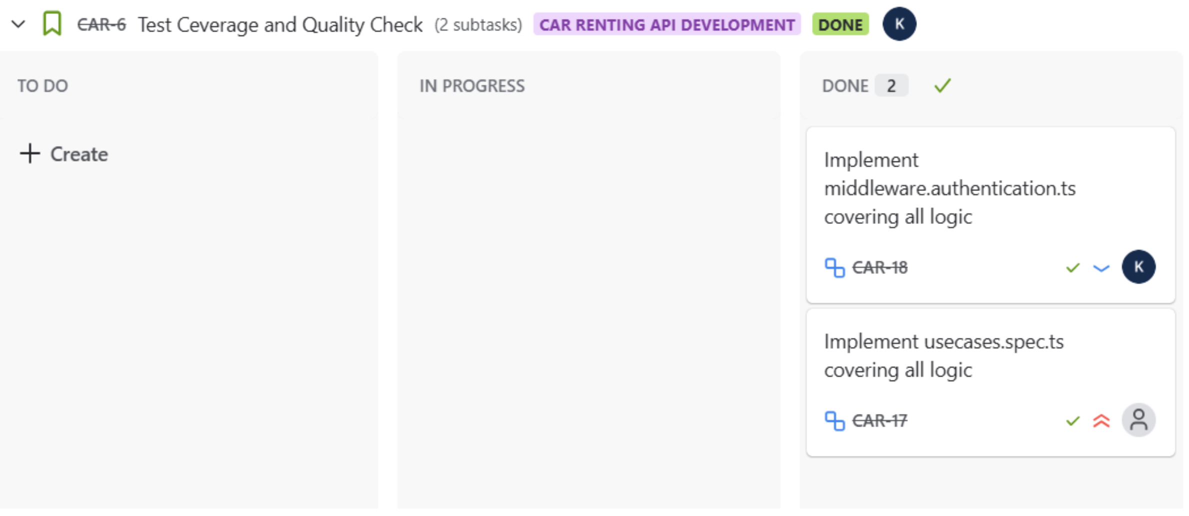
users

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	2	123.00 B	2	73.73 kB

- The backend uses MongoDB for data storage.
- Three main collections are implemented: Users, Cars, and Bookings.
- Users include two roles like admin and user. these are distinguished by the role field for authorization control.
- Each booking document references a car and a user (or guest email), maintaining relational consistency through ObjectId references

6. Testing and Coverage

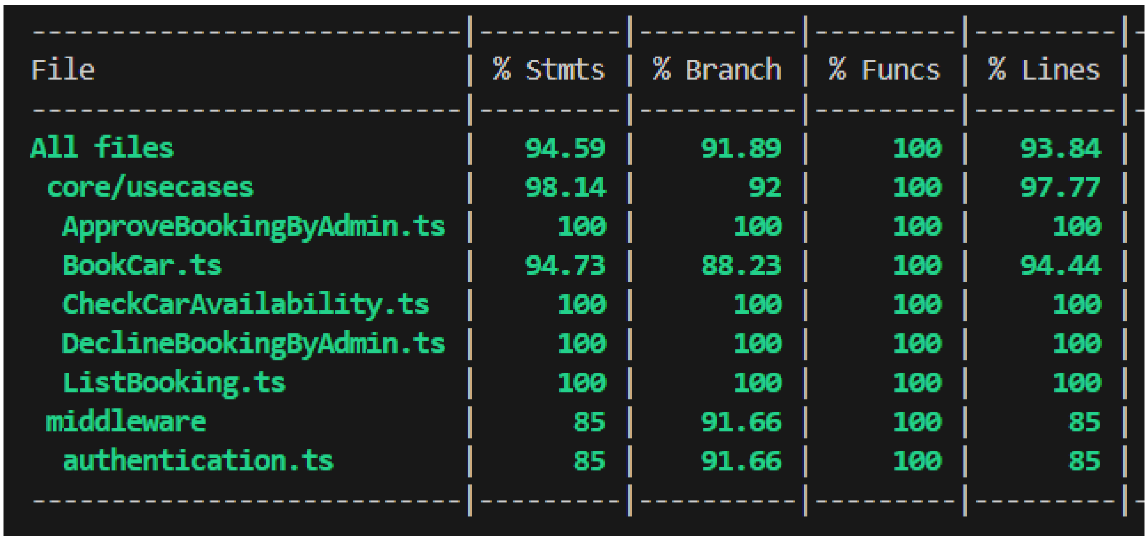
- Jira Story: CAR-6 Test Coverge and Quality Check


~~CAR-6~~ Test Ceverage and Quality Check (2 subtasks)
CAR RENTING API DEVELOPMENT
DONE
K
TO DO
+ Create
IN PROGRESS
DONE 2

Implement middleware.authentication.ts covering all logic
~~CAR-18~~

Implement usecases.spec.ts covering all logic
~~CAR-17~~

- Testing was divided into two key areas: Authentication Testing and Usecase Testing. All tests were executed using Jest. The Jest configuration collected coverage data from “/src/core” and “src/middleware” to evaluate business logic precisely.



```
---------------------------|---------|----------|---------|---------|-
File                       | % Stmts | % Branch | % Funcs | % Lines |
---------------------------|---------|----------|---------|---------|-
All files                  |   94.59 |    91.89 |     100 |   93.84 |
 core/usecases             |   98.14 |       92 |     100 |   97.77 |
  ApproveBookingByAdmin.ts |     100 |      100 |     100 |     100 |
  BookCar.ts               |   94.73 |    88.23 |     100 |   94.44 |
  CheckCarAvailability.ts  |     100 |      100 |     100 |     100 |
  DeclineBookingByAdmin.ts |     100 |      100 |     100 |     100 |
  ListBooking.ts           |     100 |      100 |     100 |     100 |
 middleware                |      85 |    91.66 |     100 |      85 |
  authentication.ts        |      85 |    91.66 |     100 |      85 |
---------------------------|---------|----------|---------|---------|-
```

6.1 Authentication Test

- Tests covered login, token refresh, role validation, and access control for different user types (guest, user, admin).
    - Login and Token Validation: Verified /users/register, /users/login, and /users/refresh endpoints returned valid JWTs.
    - Role-Based Access Control: Confirmed that only admins could access /admin/* routes, while users and guests received 403 Forbidden.
    - Console Logging: Authentication middleware logs events
- All authentication scenarios passed as expected, validating both JWT handling and role-based authorization

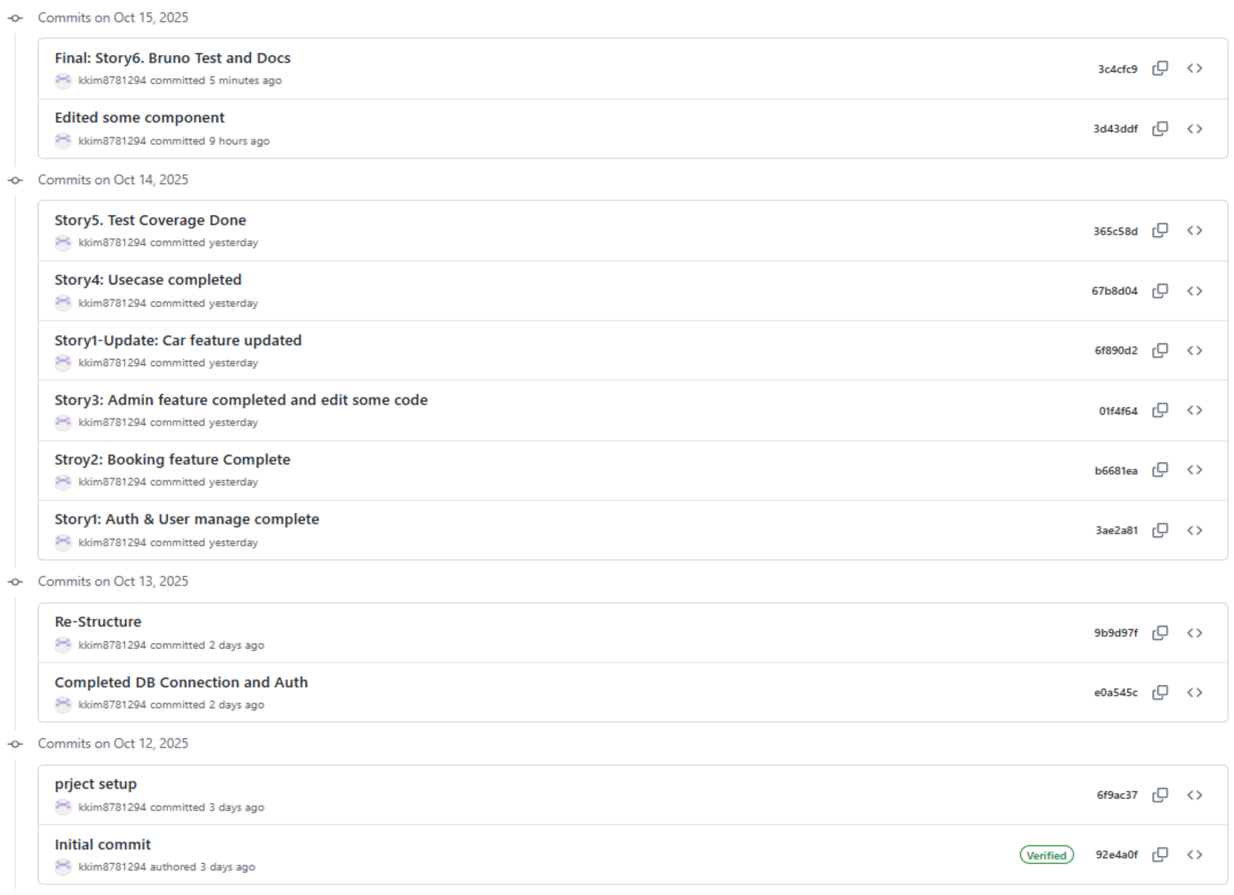
6.2 Usecase Testing

- All core business usecases were tested using Jest and test repositories(no database connection) ensuring predictable and isolated results.
    - CheckCarAvailability: Confirmed available=true when no overlap, and false otherwise.
    - BookCar: Verified total cost calculation and prevented overlapping bookings.

    - ListBookings: Returned only the user's own reservations.
    - Approve/Decline/EditingBooking: Correctly updated booking status.
- Both positive (valid data) and negative (invalid/overlapping dates) cases were tested.
- This methodical testing approach resulted in high reliability and complete coverage of all business logic.

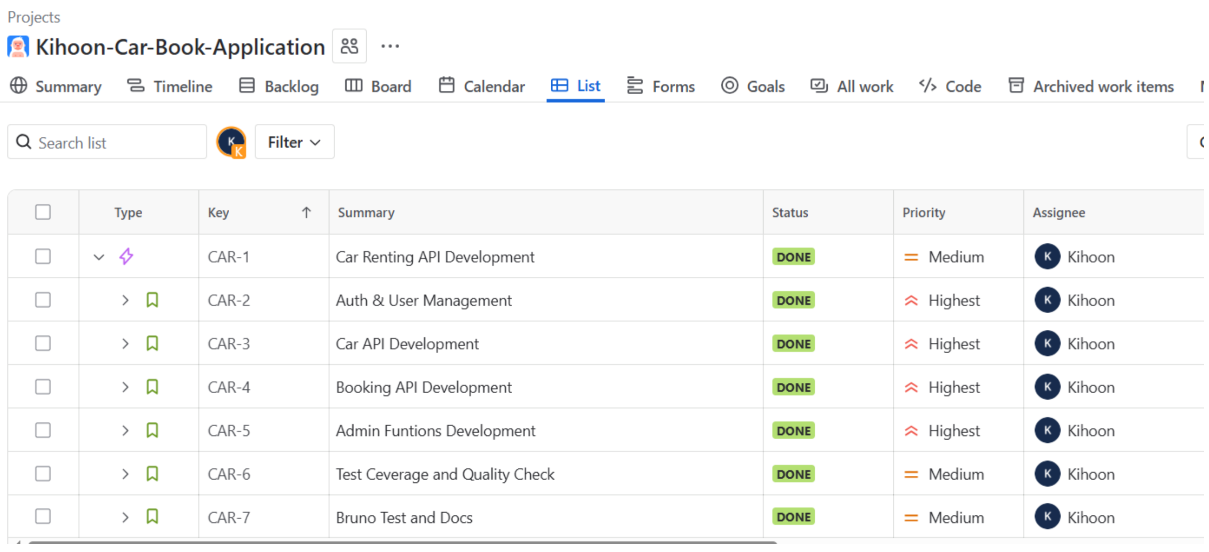
7. Project Management

- Managed under one Epic: "Kihoon-Car-Book-Application Development (4-Day Sprint)"
- Six Stories (Auth, Cars, Bookings, Admin, Test Coverage, Documentation) were planned and tracked through Jira's Kanban board.
- Each task moved from To Do → In Progress → Done across the 4-day sprint.
- Git history shows incremental commits for each feature branch.
- Git Log



```
Commits on Oct 15, 2025

Final: Story6. Bruno Test and Docs                                3c4cfc9
kkim8781294 committed 5 minutes ago

Edited some component                                             3d43ddf
kkim8781294 committed 9 hours ago

Commits on Oct 14, 2025

Story5. Test Coverage Done                                        365c58d
kkim8781294 committed yesterday

Story4: Usecase completed                                         67b8d04
kkim8781294 committed yesterday

Story1-Update: Car feature updated                                6f890d2
kkim8781294 committed yesterday

Story3: Admin feature completed and edit some code                01f4f64
kkim8781294 committed yesterday

Stroy2: Booking feature Complete                                  b6681ea
kkim8781294 committed yesterday

Story1: Auth & User manage complete                               3ae2a81
kkim8781294 committed yesterday

Commits on Oct 13, 2025

Re-Structure                                                      9b9d97f
kkim8781294 committed 2 days ago

Completed DB Connection and Auth                                  e0a545c
kkim8781294 committed 2 days ago

Commits on Oct 12, 2025

prject setup                                                      6f9ac37
kkim8781294 committed 3 days ago

Initial commit                                          Verified  92e4a0f
kkim8781294 authored 3 days ago
```

- Jira List View



Type	Key	Summary	Status	Priority	Assignee
	CAR-1	Car Renting API Development	DONE	Medium	Kihoon
	CAR-2	Auth & User Management	DONE	Highest	Kihoon
	CAR-3	Car API Development	DONE	Highest	Kihoon
	CAR-4	Booking API Development	DONE	Highest	Kihoon
	CAR-5	Admin Funtions Development	DONE	Highest	Kihoon
	CAR-6	Test Ceverage and Quality Check	DONE	Medium	Kihoon
	CAR-7	Bruno Test and Docs	DONE	Medium	Kihoon

8. Conclusion

Through this project, I gained hands-on experience with the full lifecycle of backend system development. And I think Car Renting API backend successfully implements a clean and modular system meeting all course requirements:

- Clean Architecture applied across all layers
- Authentication and authorization using JWT
- Guest and registered user booking flows supported
- Multiple user scheduling handled concurrently
- Logging and error handling included
- Test coverage exceeded 90%
- Jira and Git used for project tracking
- Bruno tests fully validated for all endpoints

Overall, I strengthened my understanding of backend architecture, API design, and testing workflows that ensure both functionality and maintainability.