ParkEase – Node.js Backend

Description of all APIs used in the ParkEase backend, including their purpose, location, and a simple explanation of how each works.

Github link : https://github.com/avi-0605/ParkEase_CFA
Deploy link : https://park-ease-cfa.vercel.app/

Authentication APIs

backend > src > routes > JS authRoutes.js > ...

```
const express = require('express');
const { register, login, getMe, updateDetails } = require('../controllers/authController');
const { protect } = require('../middlewares/authMiddleware');

const router = express.Router();

router.post('/register', register);
router.post('/login', login);
router.get('/me', protect, getMe);
router.put('/updatedetails', protect, updateDetails);

module.exports = router;
```

backend > src > controllers > JS authController.js > ...

```
const User = require('../models/User');
const jwt = require('jsonwebtoken');

// Generate JWT
const generateToken = (id) => {
    return jwt.sign({ id }, process.env.JWT_SECRET, {
        expiresIn: process.env.JWT_EXPIRE,
    });
};

// @desc    Register user
// @route   POST /api/auth/register
// @access  Public
exports.register = async (req, res, next) => {
    try {
        const { name, email, password, role } = req.body;

        // Check if user exists
        const userExists = await User.findOne({ email });

        if (userExists) {
            return res.status(400).json({ success: false, error: 'User already exists' });
        }

        // Create user
        const user = await User.create({
            name,
            email,
            password,
```

1. POST /api/auth/register

- Purpose: Register a new user

- Description:
  Allows a new user to create an account by providing name, email, and password. On successful registration, a login token is generated.

2. POST /api/auth/login

- Purpose: User login
- Description:
  Authenticates a user using email and password and returns a token for accessing protected routes.

3. GET /api/auth/me

- Purpose: Get logged-in user details
- Description:
  Returns the profile information of the currently authenticated user.

4. PUT /api/auth/updatedetails

- Purpose: Update user profile
- Description:
  Allows users to update personal details such as name or vehicle information.

Parking Lot APIs

backend > src > routes > JS parkingRoutes.js > ...

```
const express = require('express');
const {
    getParkingLots,
    getParkingLot,
    createParkingLot,
    updateParkingLot,
    deleteParkingLot,
    toggleStatus,
    archiveLot
} = require('../controllers/parkingController');

const { protect } = require('../middlewares/authMiddleware');
const { authorize } = require('../middlewares/roleMiddleware');

const router = express.Router();

router.route('/')
    .get(getParkingLots)
    .post(protect, authorize('owner', 'admin'), createParkingLot);

router.route('/:id')
    .get(getParkingLot)
    .put(protect, authorize('owner', 'admin'), updateParkingLot)
    .delete(protect, authorize('owner', 'admin'), deleteParkingLot);

router.route('/:id/toggle').put(protect, authorize('owner', 'admin'), toggleStatus);
router.route('/:id/archive').put(protect, authorize('owner', 'admin'), archiveLot);

module.exports = router;
```

backend > src > controllers > JS parkingController.js > ...

```
const ParkingLot = require('../models/ParkingLot');
const Slot = require('../models/Slot');
const Review = require('../models/Review');
const ActivityLog = require('../models/ActivityLog');

// Helper to get parking lot stats (availability & rating & pricing)
const getLotStats = async (lotId, basePrice, totalSlots) => {
    const availableSlots = await Slot.countDocuments({ lotId: lotId, status: 'available' });
    const occupationRate = (totalSlots - availableSlots) / totalSlots;

    // Dynamic Pricing Logic: Surge if > 80% occupied
    const isSurge = occupationRate > 0.8;
    const priceMultiplier = isSurge ? 1.5 : 1;
    const currentPrice = basePrice * priceMultiplier;

    // Calculate Average Rating
    const result = await Review.aggregate([
        { $match: { parkingLotId: lotId } },
        {
            $group: {
                _id: '$parkingLotId',
                averageRating: { $avg: '$rating' },
                count: { $sum: 1 }
            }
        }
    ]);

    const stats = result[0] || { averageRating: 0, count: 0 };
    return {
```

5. GET /api/parking-lots

- Purpose: Get all parking lots
- Description:
  Displays all active parking lots with pricing and availability details. Accessible to all users.

6. GET /api/parking-lots/:id

- Purpose: Get a single parking lot
- Description:
  Returns detailed information of a specific parking lot including address, price, and available slots.

7. POST /api/parking-lots

- Purpose: Create a parking lot
- Description:
  Allows admins or owners to add a new parking lot with details like name, address, price, and total slots.

8. PUT /api/parking-lots/:id

- Purpose: Update parking lot details
- Description:
  Enables admins or owners to modify parking lot information such as pricing or slot count.

9. DELETE /api/parking-lots/:id

- Purpose: Delete a parking lot
- Description:
  Removes a parking lot permanently from the system.

10. PUT /api/parking-lots/:id/toggle

- Purpose: Enable or disable parking lot
- Description:
  Temporarily activates or deactivates a parking lot from public view.

11. PUT /api/parking-lots/:id/archive

- Purpose: Archive a parking lot
- Description:
  Admin-only action to hide a parking lot permanently without deleting it.

Slot APIs

backend > src > routes > JS slotRoutes.js > ...

```
const express = require('express');
const {
    getSlotsByLot,
    updateSlotStatus
} = require('../controllers/slotController');

const { protect } = require('../middlewares/authMiddleware');
const { authorize } = require('../middlewares/roleMiddleware');

const router = express.Router();

// GET /api/slots/:lotId
router.get('/:lotId', protect, getSlotsByLot);

// PUT /api/slots/:slotId/status
router.put('/:slotId/status', protect, authorize('owner', 'admin'), updateSlotStatus);

module.exports = router;
```

const router: Router

backend > src > controllers > JS slotController.js > ...

```
const Slot = require('../models/Slot');
const ActivityLog = require('../models/ActivityLog');

// @desc    Get slots for a specific parking lot
// @route   GET /api/slots/:lotId
// @access  Private (or Public depending on requirements, usually public for searching
exports.getSlotsByLot = async (req, res, next) => {
    try {
        const slots = await Slot.find({ lotId: req.params.lotId });

        res.status(200).json({
            success: true,
            count: slots.length,
            data: slots
        });
    } catch (err) {
        next(err);
    }
};

// @desc    Update slot status
// @route   PUT /api/slots/:slotId/status
// @access  Private (Owner/Admin or System)
exports.updateSlotStatus = async (req, res, next) => {
    try {
        const { status } = req.body;

        let slot = await Slot.findById(req.params.slotId);

        if (!slot) {
```

12. GET /api/slots/:lotId

- Purpose: Get all slots in a parking lot
- Description:
  Displays all parking slots (A1, A2, B1, etc.) and their availability status.

13. PUT /api/slots/:slotId/status

- Purpose: Update slot status or type
- Description:
  Allows admins or owners to update slot availability or change slot type (normal / EV).

Booking APIs

backend > src > routes > JS bookingRoutes.js > ...

```js
const express = require('express');
const {
    createBooking,
    getMyBookings,
    extendBooking,
    endBooking
} = require('../controllers/bookingController');

const { protect } = require('../middlewares/authMiddleware');
const { authorize } = require('../middlewares/roleMiddleware');

const router = express.Router();

router.use(protect); // All booking routes are protected

router.post('/', authorize('driver'), createBooking);
router.get('/my', getMyBookings);
router.put('/extend/:id', authorize('driver'), extendBooking);

router.route('/end/:id').put(protect, authorize('driver', 'admin'), endBooking);

module.exports = router;

```

backend > src > controllers > JS bookingController.js > ...

```js
const Booking = require('../models/Booking');
const Slot = require('../models/Slot');
const ParkingLot = require('../models/ParkingLot');

// @desc    Create new booking
// @route   POST /api/bookings
// @access  Private
// @desc    Create new booking
// @route   POST /api/bookings
// @access  Private (Driver)
exports.createBooking = async (req, res, next) => {
    try {
        const { slotId, startTime, endTime } = req.body;

        // 1. Basic Validation
        const start = new Date(startTime);
        const end = new Date(endTime);
        const now = new Date();

        if (start >= end) {
            return res.status(400).json({ success: false, error: 'End time must be after start time' });
        }

        if (start < now) {
            // Allow small buffer (e.g., 2 mins) for network latency, else reject
            if ((now - start) > 2 * 60 * 1000) {
                return res.status(400).json({ success: false, error: 'Cannot book in the past' });
            }
        }
```

14. POST /api/bookings

- Purpose: Create a booking
- Description:
  Creates a new booking based on selected slot and time duration. Calculates total price automatically.

15. GET /api/bookings/my

- Purpose: View user bookings
- Description:
  Shows booking history of the logged-in user, including past and active bookings.

16. PUT /api/bookings/extend/:id

- Purpose: Extend booking time
- Description:
  Allows users to extend their parking duration with updated pricing.

17. PUT /api/bookings/end/:id

- Purpose: End booking early
- Description:
  Ends an active booking early and frees up the slot for other users.

Payment APIs

backend > src > routes > JS paymentRoutes.js > ...

```
const express = require('express');
const { createPayment } = require('../controllers/paymentController');
const { protect } = require('../middlewares/authMiddleware');

const { authorize } = require('../middlewares/roleMiddleware');

const router = express.Router();

router.post('/', protect, authorize('driver'), createPayment);

module.exports = router;

```

backend > src > controllers > JS paymentController.js > ...

```
const Payment = require('../models/Payment');
const Booking = require('../models/Booking');

// @desc    Process payment
// @route   POST /api/payments
// @access  Private
exports.createPayment = async (req, res, next) => {
    try {
        const { bookingId, amount } = req.body;

        const booking = await Booking.findById(bookingId);

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

        // Verify booking belongs to user
        if (booking.userId.toString() !== req.user.id) {
            return res.status(403).json({ success: false, error: 'Not authorized' });
        }

        // Simulate payment success
        const payment = await Payment.create({
            bookingId,
            amount,
            paymentStatus: 'success',
            paymentMode: 'simulated'
        });

        res.status(201).json({
```

18. POST `/api/payments`

- Purpose: Record payment
- Description:
  Stores payment details for a booking. Payment is currently simulated (no real gateway).

Review APIs

backend > src > routes > JS reviewRoutes.js > ...

```
const express = require('express');
const { createReview, getReviews } = require('../controllers/reviewController');
const { protect } = require('../middlewares/authMiddleware');

const { authorize } = require('../middlewares/roleMiddleware');

const router = express.Router();

router.post('/', protect, authorize('driver'), createReview);
router.get('/:lotId', getReviews);

module.exports = router;

```

backend > src > controllers > JS reviewController.js > ...

```
const Review = require('../models/Review');
const ParkingLot = require('../models/ParkingLot');

// @desc    Create review
// @route   POST /api/reviews
// @access  Private
exports.createReview = async (req, res, next) => {
    try {
        const { parkingLotId, rating, comment, issueReported } = req.body;

        // Check if parking lot exists
        const parkingLot = await ParkingLot.findById(parkingLotId);
        if (!parkingLot) {
            return res.status(404).json({ success: false, error: 'Parking lot not found' });
        }

        const review = await Review.create({
            userId: req.user.id,
            parkingLotId,
            rating,
            comment,
            issueReported
        });

        res.status(201).json({
            success: true,
            data: review
        });
    } catch (err) {
        next(err);
```

19. POST /api/reviews

- Purpose: Add a review

- Description:
  Allows users to submit ratings and feedback after using a parking lot.

20. GET /api/reviews/:lotId

- Purpose: View parking lot reviews
- Description:
  Displays all reviews and ratings for a specific parking lot.

Admin APIs

backend > src > routes > JS adminRoutes.js > ...

```
const express = require('express');
const { getPeakHours, getSystemAlerts, getDashboardStats, getActivityLogs, getReviews } = require('../controllers/ad
const { getAllParkingLots } = require('../controllers/parkingController');
const { protect } = require('../middlewares/authMiddleware');
const { authorize } = require('../middlewares/roleMiddleware');

const router = express.Router();

router.use(protect);
router.use(authorize('admin', 'owner'));

router.get('/analytics/peak-hours', getPeakHours);
router.get('/alerts', getSystemAlerts);
router.get('/stats', getDashboardStats);
router.get('/logs', getActivityLogs);
router.get('/reviews', getReviews);
router.get('/parking-lots', getAllParkingLots);

module.exports = router;

```

backend > src > controllers > JS adminController.js > ...

```
const Booking = require('../models/Booking');
const Slot = require('../models/Slot');
const ParkingLot = require('../models/ParkingLot');
const ActivityLog = require('../models/ActivityLog');
const Review = require('../models/Review');

// @desc    Get Peak Parking Hours based on historical bookings
// @route   GET /api/admin/analytics/peak-hours
// @access  Private (Admin)
exports.getPeakHours = async (req, res, next) => {
    try {
        const peakHours = await Booking.aggregate([
            {
                $project: {
                    hour: { $hour: "$startTime" } // Extract hour from startTime
                }
            },
            {
                $group: {
                    _id: "$hour",
                    count: { $sum: 1 } // Count bookings per hour
                }
            },
            {
                $sort: { count: -1 } // Sort by busiest
            }
        ]);

        // Transform for frontend
        const result = peakHours.map(item => {
```

Note: All admin APIs require **admin or owner** privileges.

21. GET /api/admin/stats

- Purpose: Dashboard statistics

- Description:
  Shows overall system stats like total users, bookings, revenue, and occupied slots.

22. GET /api/admin/alerts

- Purpose: System alerts
- Description:
  Displays alerts for low availability, poor ratings, or system issues.

23. GET /api/admin/logs

- Purpose: Activity logs
- Description:
  Shows a complete activity history of actions performed in the system.

24. GET /api/admin/reviews

- Purpose: View all reviews
- Description:

  Allows admins to see reviews from all parking lots in one place.

25. GET /api/admin/parking-lots

- Purpose: View all parking lots
- Description:
  Displays all parking lots including inactive and archived ones.

26. GET /api/admin/analytics/peak-hours

- Purpose: Peak hour analytics
- Description:
  Shows the busiest booking hours to help with planning and optimization.

Root API

backend > JS server.js > ...

```
const express = require('express');
const dotenv = require('dotenv');
const cors = require('cors');
const http = require('http');
const { Server } = require('socket.io');
const cron = require('node-cron');
const connectDB = require('./src/config/db');
const errorHandler = require('./src/middlewares/errorMiddleware');
const checkAndReleaseSlots = require('./src/utils/autoRelease');

// Load env vars
dotenv.config();

// Connect to database
connectDB();

const app = express();
const server = http.createServer(app);

// Initialize Socket.io
const io = new Server(server, {
    cors: {
        origin: "*",
        methods: ["GET", "POST", "PUT", "DELETE"]
    }
});

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

// Pass io to request
app.use((req, res, next) => {
    req.io = io;
    next();
});

// Routes
app.use('/api/auth', require('./src/routes/authRoutes'));
app.use('/api/parking-lots', require('./src/routes/parkingRoutes'));
app.use('/api/slots', require('./src/routes/slotRoutes'));
app.use('/api/bookings', require('./src/routes/bookingRoutes'));
app.use('/api/payments', require('./src/routes/paymentRoutes'));
app.use('/api/reviews', require('./src/routes/reviewRoutes'));
```

27. GET /

- Purpose: API health check
- Description:
  Confirms the backend server is running and returns a simple status message.

Summary

Total APIs: 27

- Authentication: 4
- Parking Lots: 7
- Slots: 2
- Bookings: 4
- Payments: 1
- Reviews: 2
- Admin: 6
- Root: 1