This a very basic example of the backend code for a parking tracker system using Node.js, Express.js , and MongoDB as the database.

```
// Import necessary modules
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

// Initialize Express app
const app = express();

// Set up middleware
app.use(bodyParser.json());

// Connect to MongoDB
mongoose.connect('mongodb://localhost/parkingtracker', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});

// Define a schema for parking spaces
const parkingSpaceSchema = new mongoose.Schema({
  location: String,
  isOccupied: Boolean
});

const ParkingSpace = mongoose.model('ParkingSpace', parkingSpaceSchema);

// Define API routes

// Get all parking spaces
app.get('/api/parking', async (req, res) => {
  try {
    const parkingSpaces = await ParkingSpace.find();
    res.json(parkingSpaces);
  } catch (error) {
    res.status(500).json({ error: 'Server error' });
  }
});

// Update parking space status
app.put('/api/parking/:id', async (req, res) => {
  try {
    const parkingSpace = await ParkingSpace.findByIdAndUpdate(
      req.params.id,
      { isOccupied: req.body.isOccupied },
      { new: true }
    );
    res.json(parkingSpace);
```

```
  } catch (error) {
    res.status(500).json({ error: 'Server error' });
  }
});

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

The code sets up an Express.js server, connects to a MongoDB database, defines a schema for parking s paces, and provides two API routes: one to get all parking spaces and another to update a parking spaces occupancy status. You'll need to install the necessary dependencies (express, body-parser, mongoose) using npm before running this code.

Please note that this example is minimal and lacks proper error handling, security measures, authenticati on, and more advanced features that a real-world parking tracker system would require. Additionally, the frontend code for user interaction, including the reservation system and real-time updates, is not included in this basic example.