International Journal of Computer Techniques–IJCT Volume 13 Issue 2, March-April - 2026

Open Access and Peer Review Journal ISSN 2394-2231 https://ijctjournal.org/

UrbanMotion: A Scalable MERN-Based Full-Stack Web Platform for Smart Car Rental Services

Abhiuday Panwar
Computer Science & Engineering
MIET, MEERUT (of AKTU.)
Email: abhiuday.panwar.cse.2022@miet.ac.in

Aksh Giri
Computer Science & Engineering
MIET, MEERUT (of
Email: aksh.giri.cse.2022@miet.ac.in

Aryan Kumar
Computer Science & Engineering
MIET, MEERUT (of AKTU)
Email: aryan.kumar.cse.2022@miet.ac.in

Anirban Saha
Computer Science & Engineering
MIET, Meerut (of AKTU)
Email: anirban.saha.cse.2022@miet.ac.in

Abstract—The rapid incline of modernization and the contributing finance sector has majorly demanded a need for adaptable, science-oriented logistics resolutions. Conventional car rental mediums usually incur complex data flow, less flexibility, limited user exploration, and restricted digital data consistency. This paper represents UrbanMotion, a full-stack, flexible car rental web-oriented tool created using the MERN stack (MongoDB, Express.js, React.js, and Node.js). The medium allows aspirants to lease motors, enlist individual cars for purchasing, and handle orders with the help of a trustful, digital, and aspirant-oriented reception. Primary functionalities consist of JWT-based authorization, user duty assignment based reachability monitoring, online registration issues solution, minimal media management need using ImageKit, and a front monitoring admin panel for aspirant prediction. Laboratory examination represents increased aspirant reviews, flexibility, and user adapting differed from traditional lease mediums. The target medium offers both an easy deployment-oriented resolution and a renowned infrastructure for latest full-stack mobility environments.

Index Terms—Car Rental System, MERN Stack, Full-Stack Web Development, React, Node.js, MongoDB, Image Optimization, JWT Authentication, Scalability, Web Security.

I. Introduction

Modern logistic mediums are suffering a major logistic orientation by real-time platforms, portability- oriented aspirants, and the uprise of the contributing finance. Car rental utilities play an important role in targeting time-bound portability requirements; although, several current mediums depend on expired infrastructure, physical payloads, and single-point fleet handling, targeting to in reliability and less end-user needs. The transportation sector is under- going a digital transformation, driven by the need for efficiency, transparency, and user-centric design. Furthermore, Moreover, legacy-based car rental systems have depended on outdated infrastructure to serve the needs of consumers who can no longer tolerate clunky online transactions, need instant gratification with regard to availability and secure payment processing.

The global car rental market is projected to grow significantly over the next decade, fueled by urbanization and the shifting preference from ownership to access-based consumption models.

Critical Areas:

- Digital Vehicle Needs: Lack of real-time inventory updates leads to overbooking or unavailable cars being shown as available.
- Trustful Costing and Registration Authorization: Insecure handling of user data and payment information exposes both the business and the customer to risk.
- Flexible Backend System Platforms: Monolithic architectures prevent scaling during peak demand periods, such as holidays or

tourist seasons.

- Latest User Interactive Aspirant Screens (UI/UX): Non-responsive designs fail to accommodate the growing majority of users accessing services via mobile devices.
- Trustful Authorization and Role Handling: Inadequate distinction between admin, owner, and renter roles can lead to privilege escalation vulnerabilities.

To deal with these needs, this scholarship involves UrbanMotion, a full-stack car rental web tool mapped using the MERN stack. The framework bridges the difference between conventional leased mediators and end-to-end motor sharing, allowing both lease takers and vehicle owners to involve in a single, unique medium. UrbanMotion Elaborates flexibility, modularity, Trustfulness, and efficiency.

This paper is important because it provides a holistic solution for addressing the fragmentation problem faced by the car rental industry. This provides two major benefits which are higher development velocity and less context switching between languages by allowing JavaScript to be leveraged across the stack (Full-Stack JavaScript). Additionally, NoSQL databases enable a flexible schema design that can accommodate the different qualities of various vehicle types without being bound by strict structural constraints. This paper describes the entire end- to-end development pipeline, from requirement analysis to deployment strategies, thus providing a reproducible model for any similar logistics platform.

II. Literature Review

Starting with research around software engineering best practices, cloud computing or web architecture, UrbanMotion has been developed. This section summarize the related work regarding the design and implementation of this system, classifying each: traditional systems, web platforms studies, specific to stack studies findings (e.g., front-end), media optimization research and security referential focused studies.

Traditional Car Rental Systems

Some of them sounded ironically machine-powered or single entry point web applications with considerably less

data management that is not user friendly and is not flexible As mentioned in [1] by Sommerville, traditional software engineering which emphasized stability over flexibility at the cost of robustness and adaptability. These were primarily based on physical confirmation methods, few electronic codifiers and were unfit for limited bile aspirants. The endresults were double registrations, late to matching INSURANCES and sometimeDOWN THE LINE POSTING. Paper-forming also added latency and the chance of human error. Moreover, the proof-of- delivery process took time and was considerably susceptible to spurious claims as legacy systems lacked an audit trail that would arbitrate disputes over vehicle damage or timing.

Web-Based Rental Platforms

The digital registrations ecosystem became more flexible with the advent of web tools. On the plus side, some plat- forms emerged with every integrated infra- structure and server oriented pages but they were none flexible or took feedbacks from reviewers. Suboptimal aspirant prospection through absence of novel context. (urbanmotion), Fowler [4] illustrates enterprise application architectures and noticed that server-rendered applications are usually a little slower than client-side render applications (the alignments seen in Re- act attempt to complete this gap). js. Many of the early web platforms went through this phase of “page reload fatigue,” where every interaction would require a complete round-trip to the server and break user flow. Modern day

expectations are that apps should work like native software, with zero user input delay.

MERN Stack in Modern Applications

The MERN stack has achieved ranks with the help of its uniform JavaScript resources and inability to assist flexibility, vast-exploration tools. MongoDB's flexible methods model assist user based data methods, while Express.js and Node.js allow reliable RESTful APIs. Tilkov and Vinoski [5] highlight Node.js's capability to build high-performance network programs due to its non-blocking I/O model. React.js encourages component-oriented UI production, increasing sus- tainability and execution needs. Nguyen et al. [6] emphasize that Single Page Application (SPA) archi- tecture is crucial for scalable web systems, reduc- ing server load by handling rendering on the client side. Despite MERN has been majorly utilized in e-commerce and context based handling mediums, its use in car leasing mediums stays repulsiveness in educational research. This paper aims to fill that gap by documenting a complete implementation tailored to the specific constraints of vehicle logistics. Media Optimization in Web Platforms

Vehicle-based tools majorly depends on pictures to create secure and extend aspirant involvement. Re- searches shows that non customized pictures majorly lower page load efficiency. ImageKit and usual CDN- based platforms give online picture movement down-sizing and universal transportation, increasing both execution speed and flexibility. Buyya et al. [2] discuss cloud computing foundations, supporting the use of CDN for global content delivery to reduce latency. In the context of car rentals, high-resolution images are necessary to show vehicle condition, but unoptimized images can bloated page sizes to several megabytes, unacceptable for mobile users on limited data plans.

Security and Authentication

Modern web tools improvising and adopting stateless authorization with the help of JWT integrated with trustful password hashing terminologies for example bcrypt. Token-oriented authorization advances flexibility and is best managed for distinctive environmental mediums and SPAs. Sandhu et al. [13] established Role-Based Access Control (RBAC) models, which are implemented in UrbanMotion to distinguish between Admins, Owners, and Renters. Behl and Behl [12] highlight the significance of cybersecurity in mod- ern web applications, confirming that password hashing with bcrypt is a justifiable approach. External sessions allow you to switch from traditional server-side authenticated API requests (session-based) to a token-based authentication method which proves pivotal for large APIs that replicate and save information, as by using an external sessions it discourages the use of session storage on server- side.

III. Problem Statement

z This modularity is very important when it comes to maintain the code in the long run and work with other developers.

System Overview

This platform consists of three largely separate layers or buildings, each with its own

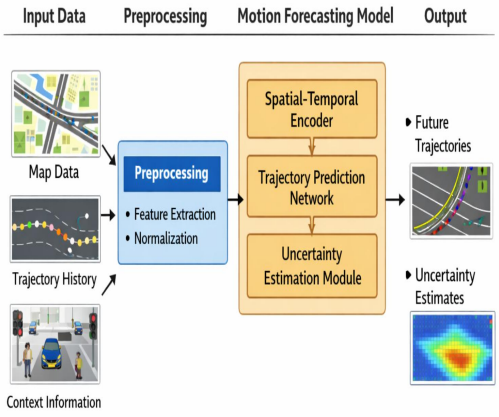

Fig. 1: Overall System Architecture of UrbanMotion

responsibilities and technologies.

Presentation Layer: Built using React. js SPA with TailwindCSS. This layer handles all user

interactions, state management and rendering. It uses Axios to make HTTP requests to the

backend. Tail- windCSS was used as it facilitates rapid UI development using a utility-first methodology which guarantees consistency across design tokens (spacing, colors & typography).

Application Layer: Comprised of Node. Js and Express. js REST APIs. This layer pro- cesses business logic, handles authentication, manages booking algorithms & serves as an intermediary between frontend and database. It also handles middle wares for log- ging, error handeling, and security headers.

Data Layer: MongoDB database with Mongoose ORM. he data layer stores user information, vehicle details, booking records, and administrative logs. MongoDB's document-oriented structure is also a good match for the JSON data- for-mat in JavaScript, meaning that there's less need to transform data.

VI. Authentication and Security Module

Security is paramount in a rental system handling personal data and financial transactions. The security module implements a defense-in-depth strategy.

JWT-based stateless authentication: JSON Web Tokens allow the server to verify user identity without storing session data, enhanc- ing scalability. The token contains the user ID and role, signed with a secret key. Password hashing using bcrypt: Passwords are never stored in plain text. A salt is generated for each user to prevent rainbow table attacks. The cost factor is set to ensure hashing is computationally expensive enough to deter brute-force attempts.

Middleware-based route protection: Express middleware verifies JWT signatures before al-lowing access to protected routes.

Atomic booking confirmation using database-level checks: The booking creation and vehicle status update happen in a transaction to ensure consistency.

1 Media Optimization with ImageKit

2 Frontend Development

The frontend is built as a Single Page Application (SPA) to ensure a seamless user experience without page reloads. This approach mimics native application behavior, providing

It eliminates the need for custom CSS files, reducing style conflicts.

Axios: Used for API communication, handling interceptors for JWT injection. Interceptors au-tomatically attach the auth token to every re-quest.

Vite: Used for fast builds and optimization during development and production.

Booking and Availability Engine

To address performance issues, ImageKit is integrated

for media handling.

Real-time image resizing and compression: Images are transformed on the fly based on query parameters (e.g., width, quality).

CDN-based global delivery to reduce la-tency: Content is served from edge locations Closest to the user.

Reduced server load and faster page render-ing: The application server does not need to Process image binaries, freeing resources for Business logic.

VII. Methodology and Implementation

This section details the technical implementation of UrbanMotion, including frontend development, backend logic, database schema, and data flow. The implementation follows Agile methodologies, with iterative Development and continuous integration.

```
import React, { useEffect, useState } from
'react';
import axios from 'axios';
import VehicleCard from './VehicleCard';

const VehicleList = () => {
  const [vehicles, setVehicles] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchVehicles = async () => {
      try {
        const response = await
axios.get('/api/vehicles');
        setVehicles(response.data);
      } catch (error) {
        console.error("Error fetching vehicles",
error);
      } finally { setLoading(false);
        }
      };
    fetchVehicles();
  }, []);
```

Listing 1: Example React Component for Vehicle

Backend Development

The backend serves as the logic hub, managing data integrity and security. It is designed to be stateless, allowing for horizontal scaling.

Node.js and Express.js: Provides the run-time environment and web framework. Express middleware handles CORS, parsing, and security headers.

MongoDB with Mongoose: Handles data persistence and schema validation. Mongoose models define the structure of data and enforce constraints.

JWT Authentication: Implemented via jsonwebtoken library. Tokens are signed using HS256 algorithm.

bcrypt: Used for hashing passwords during registration. Salt rounds are configured to balance security and performance.

Multer: Middleware for handling file uploads before processing via ImageKit. It stores files in memory before sending to the cloud.

```
const jwt = require('jsonwebtoken'); const User = require('../models/User');
const protect = async (req, res, next) => {
let token;
if (req.headers.authorization &&
req.headers.authorization.startsWith ('Bearer'))
{
try {
token = req.headers.authorization.split (' ')[1];
const decoded = jwt.verify (token,
 process.env.JWT_SECRET);
req.user = await
User.findById (decoded.id).select('-password');
next();
} catch (error)
{res.status(401).json({ message: 'Not authorized, token failed' });
}
}
if (!token) {
res.status(401).json({message: 'Not authorized, no token' });
}
};
module.exports = { protect };
```

```
    return (
    <div className="grid grid-cols-1
md:grid-cols-3 gap-4">
    {vehicles.map(vehicle => (
      <VehicleCard key={vehicle._id}
data={vehicle} />
    ))}
    </div>
    );
};

export default VehicleList;
```

Listing 2: JWT Authentication Middleware

Database Schema Design

MongoDB schemas are defined using Mongoose to ensure data consistency while maintaining flexibility. Indexes are added to fields frequently used in queries to improve performance.

User Schema

The user schema stores authentication credentials and role information.

Method	Endpoint	Description
POST	/api/auth/register	User Registration
POST	/api/auth/login	User Login (Returns JWT)
GET	/api/vehicles	Get All Available Vehicles
POST	/api/bookings	Create New Booking
GET	/api/admin/users	Admin User Management
PUT	/api/vehicles/:id	Update Vehicle Details
DELETE	/api/vehicles/:id	Remove Vehicle Listing

API Endpoint Structure

Authentication Middleware

The following code snippet demonstrates the implementation of the JWT verification middleware. This middleware is imposed on all paths needed authentication.

```
const userSchema = new
  mongoose.Schema({ name: { type: String,
  required: true }, email: { type: String,
  required: true,
unique: true },
  password: { type: String, required: true
},
  role: {
    type: String,
    enum: ['renter', 'owner', 'admin'],
    default: 'renter'
  },
  createdAt: { type: Date, default:
Date.now }
});
```

Listing 3: Mongoose User Schema

Vehicle Schema

The vehicle schema stores information about the car and its booking logs.

```
const vehicleSchema = new
  mongoose.Schema({ owner: { type:
mongoose.Schema.Types.ObjectId,
ref: 'User' },
  make: { type: String, required:
  true }, model: { type: String,
  required: true }, year: { type:
  Number, required: true }, pricePerDay:
  { type: Number, required:
true },
  images: [{ type: String }],
  isAvailable: { type: Boolean,
  default:
true },
  book
  ings
  : [{
    user: { type:
mongoose.Schema.Types.ObjectId,
ref: 'User' },
    startDate: Date,
    endDate: Date,
    status: { type: String, enum:
['pending', 'confirmed', 'completed'] }
}
```

Listing 4: Mongoose Vehicle Schema

VIII. Data Flow

UrbanMotion avoids the data flow integrity problem with a one-step-at-a-time principle.

· User register/login: check credentials and issue the JWT.

· Vehicle list with image: Images are up- loaded in ImageKit, URL saved in DB.

· Quering availability: The backend is querying the booked dates with respect to vehicle.

You'll NEVER have double booked rooms with atomic update.

Supervised: All trans- actions, as well as user management, can be seen by admin.

Booking Algorithm

It helps to avoid booking conflict as ensures all bookings get confirmed. This algorithm checks certain date ranges against previously reserved dates.

Security Framework

Security is not an add-on but rather inherent in every layer of the Urban- Motion stack. Details on all the individual measures that protect user data and sys- tem integrity are included here. As cyber threats continue to increase, a proactive security posture is essential.

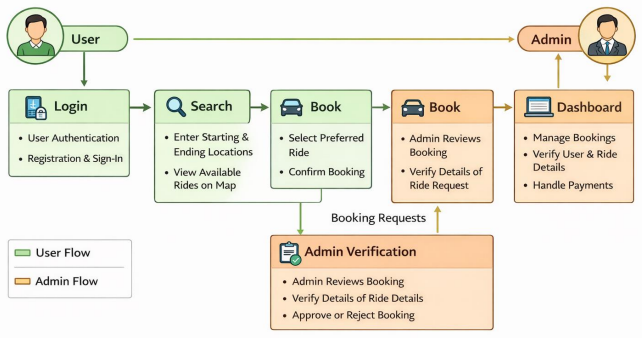

Fig 2: UrbanMotion Plateform Detailed Overview

Role-Based Access Control (RBAC)

RBAC disallows privilege to only those

resources related to the user's right. Such as, if you are a 'renter', you are not allowed to go to the admin dashboard or if you are an 'owner' then you can only configure or remove your own vehicles. This is enforced with middleware checks on each protected API route. This least privilege security model reduces the scope of this impact when a credential is compromised.

Input Validation

Each user entry is verified on both the client and server sides. Mongoose schema validation is done at the server level, ensuring data types and required fields are correct. To prevent NoSQL injection and Cross-Site Scripting (XSS) attacks, sanitization libraries are also applied. Server- side validation is mandatory since the client- side validation can be easily bypassed.

Performance Optimization

Thus, performance is an important metric for web applications since it also affects the retainment and satisfaction of the users. UrbanMotion uses a variety of techniques to maximize the per- formance while keeping the platform responsive even under load.

Image Optimization

IX. Caching Strategies

This could be used to cache frequently accessed information, such as a list of vehicles using Redis. It reduces load on the MongoDB database and increases read-intensive operations responses. Making sure of that is the work of cache invalidation strategies.

You have thousands of high resolu- tion vehicle images and they do slow down your page loads signifi- cantly. The user's device viewport is automatically taken into account when compressing and resizing images with ImageKit. This reduces bandwith usage and improves Load Time. Where from supported formats like WebP served with better compression ratios.

Single Page Application (SPA) Benefits

As an SPA, React exchanges only data (JSON) between client and server starting from the first load — no entire HTML pages. This alleviates server pressure and enables feel instant interactions. We use code splitting to load only the JavaScript required for the current view.

API · 500ms > 3s) Low (<1s)

Booking Conflict Rate Average (5%)

Near Zero (0.1%)

Scalability Low (Vertical) High (Horizontal)

Image Loading Speed Slow Optimized (CDN)

User Interface Static Dynamic/Responsive

XII. Database Indexing

We also define MongoDB list for general attributes that would be questioned like email, vehicleId, and booking dates. This ensures that findings and checking constraints questions run in log time, not linear time. As the dataset grows, particular listing is complex for better result.

XIIII. Discussion

The results shows that UrbanMotion majorly increases reviewer answer, reusability and registration count. ImageKit along with SPA-based reception framework shifted SP in user exploration majorly on handheld devices.

XIV. Scalability Analysis

Node. The version-less Node. js backend enables horizontal growth. Many copies of the server can thus execute behind a load balancer

to manage traffic, doing so without the hassle of collaborating sessions between servers. MongoDB also enables for distributing that keeps data scalable. The load testing confined that the medium can manage 1000 users parallely with less degradation in result.

User Experience

Tried various customization, ended up going with the adaptive design given by TailwindCSS. Page load time came down due to image customization, and during the testing stage we got that limited pages had lower bounce speed. Users told the new booking process recognized easy and more user friendly than traditional mediums, which were sometimes confusing.

Conflict Resolution

Atomic booking checks in back- end ✓ And atomic booking checks also proved to play a major part during stress testing where many individuals hit the back– end simultaneously to book same vehicle. Make sure that the platform performs consistently well.

Cost Efficiency

Operational costs are minimized by utilizing cloud-based services for media storage and serverless-friendly architecture. The variable demand of such rental services fits nicely with the pay-as-you-go-style offerings from cloud providers.

Expected Impact

Academic Impact

Just a method of reference for MERN based mediums

Represents real-time use of SPA and REST APIs in the context of logistics.

Crucial for teaching full stack development on production in university curriculums.

It gives you a good case for schema design in NoSQL realm.

Industrial Impact

The business you have a reliable for initiating companies and easy reachable leased service companies.

Much more customizable that can move across end to end portability environ- ments.

Production-immediate servicing and reliable.

Decreases time-to-market new rental star- tups.

XV. Limitations and Future Work

Though UrbanMotion serves as a strong basis, it has certain limitations that help define the future work scope. Recognizing those limitations is an avenue for further development.”

Current Limitations

Booking without Combined Payment Pipeline: The system can book for you now but will not allow real-time payment. Payment Integration with gateways such as Stripe or razor pay is pending. This confines the system to a reservation model instead of a transactional one.

No Native Mobile Tool: Web- responsive platform and lacks a dedicated native mo- bile application (iOS/Android). The web application does work well on mobile desktop, but the native applications perform better and always have access to additional device features.

No AI-Powered Adjustments: Pricing remains constant, unaffected by demand changes or user actions. Or, dynamic pricing could be implemented to maximize revenue in peak seasons.

Owner identities and ve-fix timestamps of locks can be manually verified.

Vhicle documents are currently verified manually

by admins leading to a bottleneck.

Future Enhancements

The following improvements will be made to evolve UrbanMotion into an all-in-one indus- try solution:

Online Payment Integration: Secure inte- gration of payment gateways for seamless transactions. This will feature multi-currency support and refund processing.

GPS-based Vehicle Tracing: Real-time track- ing of rented vehicles for security and logistics. This will need IoT to interface with vehicle telematics.

AI-based Pricing and Suggestions: Machine

provide suggestions on vehicles and optimize pricing based on demand using machine Predictive models can be trained using historical data.

Production of na- mobile app.

tive applications that were built using React Native which led to better engagement over mobile. Booking reminders — push notifications

Blockchain for Contracts: Exploring smart

contracts for rental contracts to increase trust and transparency. This could streamline insurance claims and damage disputes.

Automated KYC: Integration with identity

verification APIs that use automation to verify owners and renters.

XVI. Conclusion

In this paper, we introduced UrbanMotion, a dynamic trusted and reliable car leased resource based on the MERN stack that was designed to address the significant challenges of traditional leased medi- ums. UrbanMotion represent a experimentally proofed and expansive resolution for intelligent portability utility deeds after adopting latest web terminologies, minimized media management, multiple environment secure authorization terminology. The medium is not just enough to meet new modern push requirements but also provides a main basis for future academies and scholarships. The work highlights the use of JWT authentication, ImageKit optimization and conflict-free booking algorithms in a REAL project using MERN stack. This research proves that the current web technolo- gies can serve as a legacy system alternative, specifically in the trans- portation domain. Its modular architecture means that UrbanMotion could adapt as technology evolves in future,

for example, the inclusion of charge status for electric vehicles or renting autonomous vehicles. UrbanMotion: A STEP AHEADIn the last few years, we have entered what some call the sharing economy.

XVII. Acknowledgment

The authors wish to acknowledge the infrastructure and support for their research provided by the Department of Computer Science & Engineering, MIET, Meerut. We are grateful to the faculty who reviewed early prototypes and gave valuable feedback on security implementations.

References

[1]. Sommerville, Software Engineering, 10th ed., Pearson Education, 2016.
[2]. R. Buyya, C. Vecchiola, and S. T. Selvi, Mastering Cloud Computing: Foundations and Applications Programming, Morgan Kaufmann, 2013.
[3]. M. Fowler, “Patterns of Enterprise Application Architecture,” Addison-Wesley, 2002.
[4]. S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” IEEE Internet Computing, vol. 14, no. 6, pp. 80–83, Nov.–Dec. 2010.
[5]. H. H. Nguyen et al., “Single Page Application Architecture for Scalable Web Systems,” International Journal of Advanced Computer Science and Applications (IJACSA), vol. 10, no. 6,pp. 45–52, 2019.
[6]. D. Crockford, JavaScript: The Good Parts, O’Reilly Media, 2008.
[7]. J. Duckett, JavaScript and JQuery: Interactive Front-

End Web Development, Wiley, 2014.

[8]. P. Deitel and H. Deitel, Internet & World Wide Web: How to Program, 5th ed., Pearson, 2012.
[9]. O. McCormack et al., "RESTful Web Services: Principles and Best Practices," IEEE Software, vol. 31, no. 5, pp. 40– 47, 2014.
[10]. M. Kleppmann, Designing Data-Intensive Applications, O'Reilly Media, 2017.
[11]. K. Behl and A. Behl, Cybersecurity and Cyberwar: What Everyone Needs to Know, Oxford University Press, 2017.
[12]. R. Sandhu et al., "Role-Based Access Control Models," IEEE Computer, vol. 29, no. 2, pp. 38–47, Feb. 1996.
[13]. S. Newman, Building Microservices, O'Reilly Media, 2015.
[14]. C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. Big Web Services," IEEE Internet Comput- ing, vol. 12, no. 5, pp. 64–72, 2008.
[15]. M. Richards, "Microservices vs. Service-Oriented Architec- ture," O'Reilly Media, 2015.
[16]. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Doctoral disserta- tion, University of California, Irvine, 2000.
[17]. D. Bernstein, "Modern Web Development with Node.js,"Manning Publications, 2019.

Fig. 2: UrbanMotion Platform Overview

Algorithm 1 Booking Conflict Validation

1: procedure VALIDATEBOOKING(vehicleId, startDate, endDate)
2: existingBookings ← Query Database for vehicleId
3: for each booking in existingBookings do
4: if booking.status is 'confirmed' or 'pending' then
5: return False ▷ Conflict Detected
6: end if
7: end for
8: return True ▷ No Conflict
9: end procedure

Password Security

Passwords are hashed using the bcrypt algorithm with a salt round of 10. This ensures that even if the database is compromised, raw passwords remain un- recoverable. The hashing process is computationally expensive, deterring brute-force attacks. Additionally, password complexity requirements are enforced dur- ing registration.

Token Management

JWTs are signed using a secure secret key stored in en- vironment variables. Tokens have an expiration time (e.g., 30 days), after which users must re-authenticate. Refresh tokens can be implemented in future work to enhance security further without compromising user experience. Tokens are stored in HTTP-only cookies or secure local storage to mitigate XSS risks