SPORT BICYCLE STORE

Mukhamedali Kuanysh
Ulan Bakytbekov
SE-2302
Advanced Databases (NoSQL)
Shynar Akhmetzhanova

TABLE OF CONTENT

PROJECT GOAL AND OBJECTIVES

GOAL

Develop a fully functional web application for managing a bicycle store with secure user authentication, inventory management, and order processing.

OBJECTIVES

- Implement a MongoDB-based database structure optimized for CRUD operations.
- Develop a secure authentication system using JWT.
- Provide users with an intuitive interface to browse and order bicycles.
- Enable administrators to manage inventory and track orders.
- Ensure data consistency, query optimization, and API security.

PROJECT RELEVANCE

01

The global bicycle market is growing due to increasing interest in sustainable transportation

02

Online stores often lack efficient inventory and order management systems.

03

Security concerns exist in many existing e-commerce platforms.

RESEARCH & COMPETITOR ANALYSIS

TREK BIKES

01

- High-quality UI/UX with a seamless shopping experience
- Advanced filtering and search features
- Integrated dealer locator for offline purchases

02

- No rental or subscription-based models
- No built-in admin dashboard for store owners

HTTPS://WWW.TREKBIKES.COM/INTERNATIONAL/EN_IN_TL/STORE-FINDER/

SPECIALIZED BIKES

01

- Well-organized inventory and product recommendations
- Supports product customization options
- Secure checkout with multiple payment gateway

02

- No direct order tracking within the system (handled by third-party carriers)
- Limited admin control over product management

HTTPS://WWW.SPECIALIZED.COM/KZ/EN/SHOP/BIKES

SYSTEM ARCHITECTURE

Backend: Node.js, Express.js

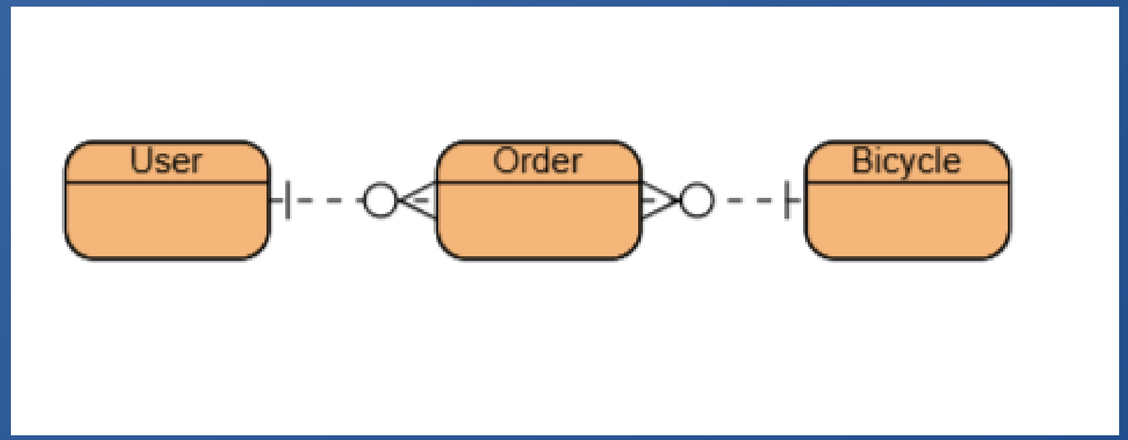
Database: MongoDB, Mongoose

Authentication: JWT, bcrypt.js

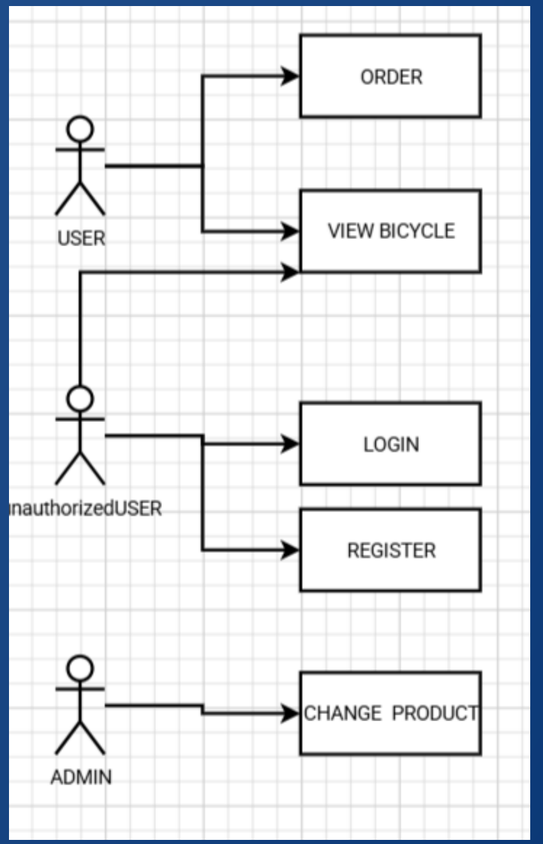
Security: Role-based access control, password hashing

DATABASE DESIGN

ERD/CRD



UML



APPLICATION FEATURES

01 Secure Authentication & Authorization

```
const registerUser = async (req, res) => {  Show usages
    try {
        const { name, email, password, isAdmin } = req.body;

        const existingUser = await User.findOne( filter: { email });
        if (existingUser) {
            return res.status(400).json({ message: "User already exists" });
        }

        const hashedPassword = await bcrypt.hash(password, salt: 10);
        const user = new User( doc: { name, email, password: hashedPassword, isAdmin: isAdmin || false });

        await user.save();

        const token = jwt.sign( payload: { id: user._id, isAdmin: user.isAdmin }, process.env.JWT_SECRET,

        res.status(201).json({ message: "User registered successfully", token });
    } catch (error) {
        res.status(500).json({ message: "Registration failed", error: error.message });
    }
};
```

02 CRUD Operations

```
router.get( path: "/", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBo
    const bicycles : Query<...> = await Bicycle.find();
    res.json(bicycles);
});

router.get( path: "/:id", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, Loc
    const bicycle : Query<...> & ObtainSchemaGeneric<module:mon... = await Bicycle.findById(req.params.id);
    if (!bicycle) return res.status( code: 404).json( body: { message: "Bicycle not found" });
    res.json(bicycle);
});

router.post( path: "/", auth, admin, async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<any, Reco
    const bicycle : HydratedDocument<InferSchemaType<...> = new Bicycle(req.body);
    await bicycle.save();
    res.status( code: 201).json(bicycle);
});

router.put( path: "/:id", auth, admin, async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<any, Re
    const bicycle : Query<...> = await Bicycle.findByIdAndUpdate(req.params.id, req.body, options: { new: true }
    if (!bicycle) return res.status( code: 404).json( body: { message: "Bicycle not found" });
    res.json(bicycle);
});
```

03 Order Management

```
const router : Router = express.Router();

router.post( path: "/", auth, createOrder);

router.get( path: "/", auth, admin, getOrders);

router.get( path: "/my-orders", auth, getUserOrders);

router.put( path: "/:id", auth, admin, updateOrderStatus);

module.exports = router;
```

04 Security Measures

```
const userSchema : Schema<any, Model<...>, {...}, {...}, {...}, DefaultSchemaOptions, {...}> =
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    isAdmin: { type: Boolean, default: false },
});

module.exports = mongoose.model( name: "User", userSchema);
```

CHALLENGES AND SOLUTIONS

ENSURING SECURE AUTHENTICATION

Implemented JWT-based authentication with bcrypt password hashing.

OPTIMIZING DATABASE QUERIES

Used indexing and aggregation in MongoDB for efficient queries.

MANAGING STOCK LEVELS AUTOMATICALLY

Updated stock dynamically when orders are placed.

CONCLUSION AND FUTURE IMPROVEMENTS

Future Enhancements:

- Develop a frontend using React/Vue for a better user experience.
- Integrate a payment gateway for online transactions.
- Implement AI-based bicycle recommendations.

- This project lays a strong foundation for a full-scale e-commerce bicycle store. By integrating a frontend, enhancing security, and optimizing performance, it has the potential to become a fully functional online marketplace.

THANK YOU

123-456-7890

hello@reallygreatsite.com