Login/Signup API

```
// Import necessary modules

import express, { Request, Response, NextFunction } from 'express';

import mongoose from 'mongoose';

import bodyParser from 'body-parser';

import jwt from 'jsonwebtoken';


// Create Express app

const app = express();

const port = 3000;


// MongoDB connection

mongoose.connect('mongodb://localhost:27017/your-database', { useNewUrlParser: true, useUnifiedTopology: true });


// Create MongoDB Schema and Model for Seller

interface ISeller {

    username: string;

    password: string;

    // Add other fields as needed

}
```

```
const SellerSchema = new mongoose.Schema<ISeller>({
    username: { type: String, required: true },
    password: { type: String, required: true },
});

const SellerModel = mongoose.model<ISeller>('Seller', SellerSchema);

// Middleware to authenticate the seller
const authenticateSeller = async (req: Request, res: Response, next: NextFunction) => {
    try {
        // Get token from headers
        const token = req.header('Authorization')?.replace('Bearer ', '');

        if (!token) {
            throw new Error('Authentication failed');
        }

        // Verify the token
        const decoded: any = jwt.verify(token, 'your-secret-key');

        // Attach the seller information to the request
        const seller = await SellerModel.findById(decoded._id);

        if (!seller) {
            throw new Error('Seller not found');
```

```
        }

        req.body.seller = seller;

        next();
    } catch (error) {
        res.status(401).send({ error: 'Authentication failed' });
    }
};

// Express middleware to parse JSON
app.use(bodyParser.json());

// Login endpoint
app.post('/auth/login', async (req, res) => {
    try {
        const { username, password } = req.body;

        // Check if the seller exists
        const seller = await SellerModel.findOne({ username });

        if (!seller || seller.password !== password) {
            throw new Error('Invalid username or password');
        }
```

```
        // Create a JWT token
        const token = jwt.sign({ _id: seller._id }, 'your-secret-key');

        res.send({ token });
    } catch (error) {
        res.status(400).send({ error: error.message });
    }
});

// Logout endpoint
app.post('/auth/logout', authenticateSeller, (req, res) => {
    // Your logout logic here, such as destroying the session
    res.send({ message: 'Logout successful' });
});

// Session check endpoint
app.get('/auth/session', authenticateSeller, (req, res) => {
    // Access seller information from req.body.seller
    res.send({ seller: req.body.seller });
});

// Start the server
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Tracking API

Order Tracking:

```
// Import necessary modules

import express, { Request, Response } from 'express';

import mongoose from 'mongoose';

import bodyParser from 'body-parser';


// Create Express app

const app = express();

const port = 3000;


// MongoDB connection

mongoose.connect('mongodb://localhost:27017/your-database', { useNewUrlParser: true, useUnifiedTopology: true });


// Create MongoDB Schema and Model for Order Tracking

interface ITracking {

    trackingId: string;

    shippingBy: string;

    shippingFrom: string;

    deliveringTo: string;

    // Add other fields as needed

}


const TrackingSchema = new mongoose.Schema<ITracking>({
```

```
    trackingId: { type: String, required: true },
    shippingBy: { type: String, required: true },
    shippingFrom: { type: String, required: true },
    deliveringTo: { type: String, required: true },
});

const TrackingModel = mongoose.model<ITracking>('Tracking', TrackingSchema);

// Express middleware to parse JSON
app.use(bodyParser.json());

// Order Tracking endpoints
app.get('/tracking', async (req, res) => {
    try {
        // Implement logic to retrieve list of tracking information with pagination, sorting, and filtering
        const trackingList = await TrackingModel.find({});

        res.send(trackingList);
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.get('/tracking/search', async (req, res) => {
    try {
```

```
        const searchTerm = req.query.searchTerm as string;

        // Implement logic to search tracking information based on the search term
        const searchResults = await TrackingModel.find({ $text: { $search: searchTerm } });

        res.send(searchResults);
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.get('/tracking/filter', async (req, res) => {
    try {
        // Implement logic to filter tracking information based on query parameters
        const filterResults = await TrackingModel.find(req.query);

        res.send(filterResults);
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.put('/tracking/:trackingId', async (req, res) => {
    try {
        const { trackingId } = req.params;
```

```
        // Implement logic to update tracking information for a specific order
        const updatedTracking = await TrackingModel.findOneAndUpdate({ trackingId }, req.body, { new: true });

        res.send(updatedTracking);
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.delete('/tracking/:trackingId', async (req, res) => {
    try {
        const { trackingId } = req.params;

        // Implement logic to delete tracking information for a specific order
        await TrackingModel.deleteOne({ trackingId });

        res.send({ message: 'Tracking information deleted successfully' });
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.get('/tracking/:trackingId', async (req, res) => {
    try {
```

```
        const { trackingId } = req.params;

        // Implement logic to retrieve detailed tracking information for a specific order
        const detailedTracking = await TrackingModel.findOne({ trackingId });

        res.send(detailedTracking);
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.patch('/tracking/bulk-update', async (req, res) => {
    try {
        // Implement logic to perform bulk updates on multiple tracking entries
        // req.body should contain the updates and criteria for bulk updates
  res.send({ message: 'Bulk updates applied successfully' });
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

// Start the server
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Analy cs and Repor ng:

```
// Import necessary modules

import express, { Request, Response } from 'express';

import mongoose from 'mongoose';

import bodyParser from 'body-parser';


// Create Express app

const app = express();

const port = 3000;


// MongoDB connection

mongoose.connect('mongodb://localhost:27017/your-database', { useNewUrlParser: true, useUnifiedTopology: true });


// Create MongoDB Schema and Model for Orders

interface IOrder {

    orderId: string;

    productId: string;

    quantity: number;

    // Add other fields as needed

}


const OrderSchema = new mongoose.Schema<IOrder>({

    orderId: { type: String, required: true },
```

```
    productId: { type: String, required: true },
    quantity: { type: Number, required: true },
});

const OrderModel = mongoose.model<IOrder>('Order', OrderSchema);

// Create MongoDB Schema and Model for Customers
interface ICustomer {
    customerId: string;
    name: string;
    email: string;
    // Add other fields as needed
}

const CustomerSchema = new mongoose.Schema<ICustomer>({
    customerId: { type: String, required: true },
    name: { type: String, required: true },
    email: { type: String, required: true },
});

const CustomerModel = mongoose.model<ICustomer>('Customer', CustomerSchema);

// Express middleware to parse JSON
app.use(bodyParser.json());
```

```
// Analytics and Reporting endpoints
app.get('/reports/sales', async (req, res) => {
    try {
        // Implement logic to generate sales reports
        const salesReport = await OrderModel.aggregate([
            {
                $group: {
                    _id: '$productId',
                    totalSales: { $sum: '$quantity' },
                },
            },
        ]);

        res.send(salesReport);
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.get('/reports/orders', async (req, res) => {
    try {
        // Implement logic to generate order reports
        const orderReport = await OrderModel.find({});

        res.send(orderReport);
```

```
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

app.get('/reports/customers', async (req, res) => {
    try {
        // Implement logic to generate customer reports
        const customerReport = await CustomerModel.find({});

        res.send(customerReport);
    } catch (error) {
        res.status(500).send({ error: 'Internal Server Error' });
    }
});

// Start the server
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```