```
const express = require("express");
const bcrypt = require('bcrypt')
const path = require("path");
const app = express();
const cors = require('cors')
const jwt = require('jsonwebtoken');
const port = process.env.PORT || 5100;
const mongoose = require('mongoose');
const { MONGO_URI } = require('./db/connect');
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

const models = require("./db/schema");

app.use(cors());

// admin middelware
function adminAuthenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) return res.status(401).send('Unauthorized');
    jwt.verify(token, 'ADMIN_SECRET_TOKEN', (err, user) => {
        if (err) return res.status(403).send('Forbidden');
        req.user = user;
        next();
    });
}

// user middleware
const userAuthenticateToken = async (req, res, next) => {
    try {
        const authHeader = req.headers['authorization'];
        const token = authHeader.split(" ")[1]
        if (!token) {
            res.status(401);
            return res.send('Invalid JWT Token');
        }
        const decoded = jwt.verify(token, 'USER_SECRET_TOKEN')
        req.user = decoded.user;
        next();

    } catch (err) {
        console.error(err);
        res.status(500);
        res.send('Server Error');
    }
};


// admin schema
app.post('/adminlogin', async (req, res) => {
    const { email, password } = req.body;
    const user = await models.Admins.findOne({ email });
    if (!user) {
        return res.status(401).json({ message: 'Invalid email or
password' });
    }
```

```
    const isAdmin = email == 'virat@gmail.com' && password ==
'virat@1234';
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
        return res.status(401).json({ message: 'Invalid email or
password' });
    }

    // Generate a JWT token
    if (!isAdmin) {
        const token = jwt.sign({ userId: user._id }, 'mysecretkey');
        res.json({ user, token });
    } else {
        const jwtToken = jwt.sign({ userId: user._id }, 'mysecretkey');
        res.json({ user, jwtToken });
    }
});


// user schema
app.post('/adminregister', async (req, res) => {
    try {
        const { firstname, lastname, username, email, password } =
req.body;

        if (!username) {
            return res.status(400).send('Username is required');
        }

        const userExists = await models.Admins.findOne({ username });

        if (userExists) {
            return res.status(400).send('Username already exists');
        }

        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);

        const newUser = new models.Admins({
            firstname,
            lastname,
            username,
            email,
            password: hashedPassword
        });

        const userCreated = await newUser.save();
        console.log(userCreated, 'user created');
        return res.status(201).json({ message: 'Successfully registered'
});
    } catch (error) {
        console.log(error);
        return res.status(500).json({ error: 'An error occurred during
registration' });

    }
});
```

```
// API endpoint to add a category
app.post('/add-category', async (req, res) => {
    try {
        const { category, description } = req.body;
        if (!category) {
            return res.status(400).send('Category and description are
required');
        }
        const existingCategory = await models.Category.findOne({ category
});
        if (existingCategory) {
            return res.status(400).send('Category already exists');
        }
        const newCategory = new models.Category({
            category,
            description
        });
        const savedCategory = await newCategory.save();
        console.log(savedCategory, 'category created');
        return res.status(200).send(savedCategory);
    } catch (error) {
        console.log(error);
        res.status(500).send('Server Error');
    }
});

app.get('/api/categories', async (req, res) => {
    try {
        const cotegoriesList = await models.Category.find();
        res.status(200).send(cotegoriesList);
    } catch (error) {
        res.status(500).send('Server error');
        console.log(error);
    }
})


// Server-side code (e.g., in your Node.js + Express.js backend)

// Define a route for handling the POST request to '/add-products'
app.post('/add-products', async (req, res) => {
    try {
        // Extract the product information from the request body
        const { productname, description, price, image, category,
countInStock, rating } = req.body;

        // Validate if all required fields are provided
        if (!productname || !description || !price || !image || !category
|| !countInStock || !rating) {
            return res.status(400).send({ message: 'Missing required
fields' });
        }

        // Assuming models.Product and models.Category are defined and
imported properly
        // Create a new product document
```

```
        const product = new models.Product({
            productname,
            description,
            price,
            image,
            category,
            countInStock,
            rating,
            dateCreated: new Date()
        });

        // Save the new product document to the database
        await product.save();

        // Send a success response with the newly created product
        res.status(201).send(product);
    } catch (error) {
        // Handle any errors that occur during the process
        console.error(error);
        res.status(500).send({ message: 'Internal server error' });
    }
});



// Endpoint for adding an item to the cart
app.post('/add-to-cart', async (req, res) => {
    const {userId, productId, productName, quantity = 1 } = req.body;
    const item = new models.AddToCart({userId, productId,productName,
quantity });
    try {
        await item.save();
        res.status(200).json({ message: `Added ${quantity} of product
${productId} to cart` });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal server error' });
    }
});


app.delete('/remove-from-cart/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const result = await models.AddToCart.deleteOne({ productId: id
});
        if (result.deletedCount === 0) {
            res.status(404).json({ message: `Product with id ${id} not
found in the cart` });
        } else {
            res.status(200).json({ message: `Removed product with id
${id} from cart` });
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal server error' });
    }
});
```

```
app.get('/cart/:id', async (req, res) => {
    try {
        const cartItems = await models.AddToCart.find({ userId:
req.params.id });
        const productIds = cartItems.map(item => item.productId);
        const products = await models.Product.find({ _id: { $in:
productIds } });
        res.send(products);
    } catch (error) {
        console.error(error);
        res.status(500).send('Internal server error');
    }
});

app.post('/orders', async (req, res) => {
    const { firstname, lastname, user, phone, productId, quantity,
paymentMethod, address } = req.body;
    const product = await models.Product.findById(productId);
    const amount = product.price * quantity;
    try {
        const order = new models.Order({
            firstname,
            lastname,
            user,
            price: amount,
            phone,
            productId,
            productName:product.productname,
            quantity,
            paymentMethod,
            address
        });
        const newOrder = await order.save();
        const payment = new models.Payment({
            user,
            name:firstname+ " " +lastname,
            order: newOrder._id, // Associate the order with the payment
            amount,
            deliveryStatus: newOrder.status,
            paymentMethod,
            status: 'Pending'
        });
        const savedPayment = await payment.save();
        res.status(201).json(newOrder);
    } catch (err) {
        res.status(400).json({ message: err.message });
    }
});

app.get('/payments', async (req, res) => {
    try {
        const payments = await models.Payment.find();
        res.status(200).json(payments);
    } catch (err) {
        console.error(err);
        res.status(500).send('Server Error');
    }
```

```
});



app.get('/orders', async (req, res) => {
    try {
        const order = await models.Order.find();
        if (!order) {
            return res.status(404).json({ message: 'Order not found' });
        }
        res.json(order);
    } catch (err) {
        res.status(500).json({ message: err.message });
    }
});

// Define a route for fetching orders by user ID
app.get('/my-orders/:id', async (req, res) => {
    const userId = req.params.id;
    try {
        const userOrders = await models.Order.find({ user: userId });
        if (userOrders.length === 0) {
            return res.status(404).json({ message: 'User orders not
found' });
        }
        res.json(userOrders);
    } catch (err) {
        res.status(500).json({ message: err.message });
    }
});

app.put('/orders/:id', async (req, res) => {
    try {
        const orderId = req.params.id;
        const { status } = req.body;
        const order = await models.Order.findById(orderId);
        if (!order) {
            return res.status(404).send('Order not found');
        }

        order.status = status; // Update the order status property
        order.createdAt = Date.now()
        const payment = await models.Payment.findOne({ order: orderId });
        if (!payment) {
            return res.status(404).send('Payment not found');
        }

        payment.deliveryStatus = status; // Update the payment status
property
        if(status === 'Delivered'){
            payment.status = 'Success'
        }else{
            payment.status = "Pending"
        }
        payment.createdAt = Date.now()
        await payment.save();
        const updatedOrder = await order.save();
        res.send(updatedOrder);
```

```
    } catch (error) {
        console.error(error);
        res.status(500).send('Server error');
    }
});

app.put('/cancel-order/:id', async (req, res) => {
    try {
        const orderId = req.params.id;
        const { status } = req.body;
        const order = await models.Order.findById(orderId);
        if (!order) {
            return res.status(404).send('Order not found');
        }

        order.status = status;
        const payment = await models.Payment.findOne({ order: orderId });
        if (!payment) {
            return res.status(404).send('Payment not found');
        }
        payment.deliveryStatus = status;
        payment.status = "Failed"
        payment.createdAt = Date.now()
        await payment.save();
        const updatedOrder = await order.save();
        res.send(updatedOrder);
    } catch (error) {
        console.error(error);
        res.status(500).send('Server error');
    }
});

app.get('/orders/:id', async (req, res) => {
    try {
        const order = await models.Order.findById(req.params.id);
        if (!order) {
            return res.status(404).json({ message: 'Order not found' });
        }
        res.json(order);
    } catch (err) {
        res.status(400).json({ message: err.message });
    }
});

// POST /payments
app.post('/payments', async (req, res) => {
    try {
        const payment = new models.Payment(req.body);
        const savedPayment = await payment.save();
        res.status(201).json(savedPayment);
    } catch (err) {
        console.error(err);
        res.status(500).send('Server Error');
    }
});

// Define the route for updating a payment
app.put('/payment/:id', async (req, res) => {
```

```
    try {
        const paymentId = req.params.id;

        const payment = await models.Payment.findById(paymentId);
        if (!payment) {
            return res.status(404).send('Payment not found');
        }
        const { amount, status } = req.body;
        if (!amount || !status) {
            return res.status(400).json({ message: 'Both amount and
status are required' });
        }
        const updatedPayment = await models.Payment.findByIdAndUpdate(
            paymentId,
            { amount, status },
            { new: true, runValidators: true }
        );
        res.status(200).json({
            message: 'Payment updated successfully',
            payment: updatedPayment,
        });
    } catch (error) {
        if (error.name === 'CastError') {
            return res.status(400).json({ message: 'Invalid payment ID'
});
        }
        if (error.name === 'ValidationError') {
            return res.status(400).json({ message: error.message });
        }
        console.error(error);
        res.status(500).send('Server error');
    }
});

// Create feedback from user
app.post('/feedback', async (req, res) => {
    try {
        const { user, message } = req.body;
        const feedback = new models.Feedback({ user, message });
        const savedFeedback = await feedback.save();
        res.status(201).json(savedFeedback);
    } catch (err) {
        res.status(400).json({ message: err.message });
    }
});

// Check feedback (admin only)
app.get('/feedback', async (req, res) => {
    try {
        const feedback = await models.Feedback.find();
        res.status(200).send(feedback);
    } catch (error) {
        res.status(500).send('Server error');
        console.log(error);
    }
});

// admin schema
```

```
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    const user = await models.Users.findOne({ email });
    if (!user) {
        return res.status(401).json({ message: 'Invalid email or
password' });
    }
    const isAdmin = email == 'virat@gmail.com' && password ==
'virat@1234';
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
        return res.status(401).json({ message: 'Invalid email or
password' });
    }

    // Generate a JWT token
    if (!isAdmin) {
        const token = jwt.sign({ userId: user._id }, 'mysecretkey');
        res.json({ user, token });
    } else {
        const jwtToken = jwt.sign({ userId: user._id }, 'mysecretkey');
        res.json({ user, jwtToken });
    }
});


// user schema
app.post('/register', async (req, res) => {
    try {
        const { firstname, lastname, username, email, password } =
req.body;

        if (!username) {
            return res.status(400).send('Username is required');
        }

        const userExists = await models.Users.findOne({ username });

        if (userExists) {
            return res.status(400).send('Username already exists');
        }

        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);

        const newUser = new models.Users({
            firstname,
            lastname,
            username,
            email,
            password: hashedPassword
        });

        const userCreated = await newUser.save();
        console.log(userCreated, 'user created');
        return res.status(201).json({ message: 'Successfully registered'
});
    } catch (error) {
```

```
        console.log(error);
        return res.status(500).json({ error: 'An error occurred during
registration' });

    }
});

// get users
app.get('/users', async (req, res) => {
    try {
        const users = await models.Users.find();
        res.send(users);
    } catch (error) {
        res.status(500).send('Server error');
        console.log(error);
    }
});

app.delete('/userdelete/:id',(req,res)=>{
    let id=req.params.id;
    models.Users.deleteOne({ _id: id })
       .then((user)=>{
        res.status(200).json(user)
         })
       .catch(() => {
        res.sendStatus(500)
       })
})

app.get('/getbookings/:userId', async (req, res) => {
    const userId = req.params.userId;
    try {
        const booking = await models.Order.find({ userId
}).sort('position');
        res.json(booking);
    } catch (err) {
        res.status(500).json({ error: 'Failed to fetch tasks' });
    }
});

app.delete('/userbookingdelete/:id',(req,res)=>{
    let id=req.params.id;
    models.Order.deleteOne({_id : id})
    .then((item)=>{
          res.status(200).json(item)
    })
    .catch(()=>{
        res.status(400).json({msg:"No item found"})
    })
})

// Get Products
const getAllProducts = async () => {
    try {
        const products = await models.Product.find();
        return products;
    } catch (error) {
        console.log(error);
```

```
        return error;
    }
};

// Define a route for the "get products" API endpoint
app.get('/products', async (req, res) => {
    const products = await getAllProducts();
    res.json(products);
});

// Get a single product
app.get('/products/:id', async (req, res) => {
    try {
        const product = await models.Product.findById(req.params.id);
        if (!product) {
            return res.status(404).json({ message: 'Product not found'
});
        }
        res.json(product);
    } catch (error) {
        console.error(`Error getting product with id ${req.params.id}`,
error);
        res.status(500).json({ message: `Error getting product with id
${req.params.id}` });
    }
});

app.delete('/products/:id', async (req, res) => {
    try {
        const deletedProduct = await
models.Product.findByIdAndDelete(req.params.id);
        if (!deletedProduct) {
            return res.status(404).json({ message: 'Product not found'
});
        }
        res.status(200).json({ message: 'Product deleted' });
    } catch (error) {
        console.error(`Error deleting product with id ${req.params.id}`,
error);
        res.status(500).json({ message: `Error deleting product with id
${req.params.id}` });
    }
});

app.put('/products/:id', async (req, res) => {
    try {
        const updatedProduct = await
models.Product.findByIdAndUpdate(req.params.id, req.body, { new: true });
        if (!updatedProduct) {
            return res.status(404).json({ message: 'Product not found'
});
        }
        res.status(200).json(updatedProduct);
    } catch (error) {
        console.error(`Error updating product with id ${req.params.id}`,
error);
        res.status(500).json({ message: `Error updating product with id
${req.params.id}` });
```

```
    }
});

app.listen(port, () => {
    console.log(`Server running at http://localhost:${port}`);
});


module.exports = app;
```