4. Payment Processing (Cash and Basic E-Wallet Integration) Description: Enable payment processing for cash and e-wallets.
• Include options for: o Cash Payments: Record payments and mark the order as "Paid." o E-Wallets: Placeholder buttons for integration with APIs like GCash or PayPal.
• Update the database to reflect payment status.

BACKEND CODE

```
npm init -y
npm install express mongoose body-parser dotenv cors
```

models/order.js (Order Model)

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  customerName: { type: String, required: true },
  amount: { type: Number, required: true },
  paymentMethod: { type: String, enum: ['cash', 'e-wallet'], required: true },
  eWalletType: { type: String, enum: ['paypal', 'gcash'], default: null },
  paymentStatus: { type: String, enum: ['pending', 'paid'], default: 'pending' },
  transactionId: { type: String, default: null },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order', orderSchema);
```

controllers/paymentController.js (Payment Controller)

```
const Order = require('../models/order');

// Process Cash Payment
exports.processCashPayment = async (req, res) => {
  try {
    const { orderId } = req.body;
    const order = await Order.findById(orderId);

    if (!order) return res.status(404).json({ message: 'Order not found' });
    if (order.paymentStatus === 'paid') return res.status(400).json({ message: 'Order already paid' });

    order.paymentStatus = 'paid';
    await order.save();

    res.status(200).json({ message: 'Payment recorded successfully', order });
  } catch (error) {
    res.status(500).json({ message: 'Server error', error });
  }
};

// Placeholder for E-Wallet Payment (PayPal / GCash)
exports.processEwalletPayment = async (req, res) => {
  try {
    const { orderId, eWalletType } = req.body;
    const order = await Order.findById(orderId);

    if (!order) return res.status(404).json({ message: 'Order not found' });
    if (order.paymentStatus === 'paid') return res.status(400).json({ message: 'Order already paid' });

    // Simulate E-Wallet Payment API Call (Here, it is just a placeholder)
    const fakeTransactionId = `${eWalletType.toUpperCase()}_${Date.now()}`;

    order.paymentStatus = 'paid';
    order.eWalletType = eWalletType;
    order.transactionId = fakeTransactionId;
    await order.save();

    res.status(200).json({ message: `${eWalletType} Payment successful`, order });
  } catch (error) {
    res.status(500).json({ message: 'Server error', error });
  }
};

// Create a New Order
exports.createOrder = async (req, res) => {
  try {
    const { customerName, amount,
paymentMethod, eWalletType } = req.body;

    const newOrder = new Order({
      customerName,
      amount,
      paymentMethod,
      eWalletType: paymentMethod === 'e-
wallet' ? eWalletType : null
    });

    await newOrder.save();

    res.status(201).json({ message: 'Order created
successfully', order: newOrder });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};
```

routes/paymentRoutes.js

```
const express = require('express');
const router = express.Router();
const paymentController =
require('../controllers/paymentController');

// Create a new order
router.post('/create-order',
paymentController.createOrder);

// Cash payment processing
router.post('/cash-payment',
paymentController.processCashPayment);

// E-wallet payment processing (Placeholder for
PayPal/GCash)
router.post('/ewallet-payment',
paymentController.processEwalletPayment);

module.exports = router;
```

app.js (Main Application Entry Point)

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const dotenv = require('dotenv');
const paymentRoutes =
require('./routes/paymentRoutes');

dotenv.config();
const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true
}));

// MongoDB Connection
mongoose.connect(process.env.MONGO_URI, {
useNewUrlParser: true, useUnifiedTopology: true
})
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));

// Routes
app.use('/api/payments', paymentRoutes);

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

5. Kiosk System Description: Develop a touchscreen-friendly interface for in-store customer self-service.
Features:
o Large, intuitive buttons for item selection and order customization.
o Show customers a summary of their order with an option to modify it before finalizing.
o Sync kiosk orders with the POS system for tracking and fulfillment

Code

```
npm init -y
npm install express mongoose body-parser dotenv
cors
```

models/item.js (Item Model)

```
const mongoose = require('mongoose');
const itemSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, required: true },
  description: String,
  category: String,
  imageUrl: String,
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Item',
itemSchema);
```

models/order.js (Order Model)

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  customerName: { type: String },
  items: [
    {
      itemId: { type:
mongoose.Schema.Types.ObjectId, ref: 'Item',
required: true },
      quantity: { type: Number, default: 1 }
    }
  ],
  totalAmount: { type: Number, required: true },
  status: { type: String, enum: ['pending',
'completed'], default: 'pending' },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order',
orderSchema);
```

controllers/kioskController.js (Kiosk Controller)

```
const Item = require('../models/item');
const Order = require('../models/order');

// Get all items
exports.getItems = async (req, res) => {
  try {
    const items = await Item.find();
    res.status(200).json(items);
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};

// Create a new order
exports.createOrder = async (req, res) => {
  try {
    const { customerName, items } = req.body;

    // Calculate total amount
    let totalAmount = 0;
    const itemDetails = await Promise.all(
      items.map(async (orderItem) => {
        const item = await
Item.findById(orderItem.itemId);
        totalAmount += item.price *
orderItem.quantity;
        return { itemId: item._id, quantity:
orderItem.quantity };
      })
    );

    const newOrder = new Order({
      customerName,
      items: itemDetails,
      totalAmount
    });

    await newOrder.save();
    res.status(201).json({ message: 'Order created
successfully', order: newOrder });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};

// Get order summary
exports.getOrderSummary = async (req, res) => {
  try {
    const { orderId } = req.params;
    const order = await
Order.findById(orderId).populate('items.itemId');

    if (!order) return res.status(404).json({
message: 'Order not found' });

    const orderSummary = {
      items: order.items.map((item) => ({
          name: item.itemId.name,
          price: item.itemId.price,
          quantity: item.quantity,
          total: item.itemId.price * item.quantity
        })),
        totalAmount: order.totalAmount
      };

      res.status(200).json(orderSummary);
    } catch (error) {
      res.status(500).json({ message: 'Server error',
error });
    }
};

// Modify order (add or remove items)
exports.modifyOrder = async (req, res) => {
    try {
      const { orderId } = req.params;
      const { items } = req.body;

      const order = await Order.findById(orderId);
      if (!order) return res.status(404).json({
message: 'Order not found' });

      let totalAmount = 0;
      const itemDetails = await Promise.all(
        items.map(async (orderItem) => {
          const item = await
Item.findById(orderItem.itemId);
          totalAmount += item.price *
orderItem.quantity;
          return { itemId: item._id, quantity:
orderItem.quantity };
        })
      );

      order.items = itemDetails;
      order.totalAmount = totalAmount;
      await order.save();

      res.status(200).json({ message: 'Order
modified successfully', order });
    } catch (error) {
      res.status(500).json({ message: 'Server error',
error });
    }
};

// Sync order with POS system
exports.syncOrderWithPOS = async (req, res) =>
{
    try {
      const { orderId } = req.params;
      const order = await Order.findById(orderId);
      if (!order) return res.status(404).json({
message: 'Order not found' });

      // Placeholder for POS system integration
      const posSyncResponse = {
        status: 'success',
        message: 'Order synced with POS system'
      };

      res.status(200).json(posSyncResponse);
    } catch (error) {
      res.status(500).json({ message: 'Server error',
error });
    }
};
```

routes/kioskRoutes.js (Routes)

```
const express = require('express');
const router = express.Router();
const kioskController =
require('../controllers/kioskController');

// Get all items
router.get('/items', kioskController.getItems);

// Create a new order
router.post('/create-order',
kioskController.createOrder);

// Get order summary
router.get('/order-summary/:orderId',
kioskController.getOrderSummary);

// Modify an order
router.put('/modify-order/:orderId',
kioskController.modifyOrder);

// Sync order with POS
router.get('/sync-order/:orderId',
kioskController.syncOrderWithPOS);
```

```
module.exports = router;
```

app.js (Main Application Entry Point)

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const dotenv = require('dotenv');
const kioskRoutes =
require('./routes/kioskRoutes');

dotenv.config();
const app = express();
const PORT = process.env.PORT || 3000;

app.use(cors());
app.use(bodyParser.json());

// MongoDB Connection
mongoose.connect(process.env.MONGO_URI, {
useNewUrlParser: true, useUnifiedTopology: true
})
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));

// Routes
app.use('/api/kiosk', kioskRoutes);

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

6. Online Ordering System Description: Provide a platform for customers to place orders online. • Features:
o A responsive web/mobile interface for browsing, selecting, and ordering items.
o Customers can log in to view past orders and save preferences.
o Sync online orders with the in store POS, marking them as “Online Orders.”
• Admin Actions: o Manage the online menu and monitor online order activity.

CODE

```
npm init -y
npm install express mongoose body-parser dotenv
cors bcrypt jsonwebtoken
```

models/item.js (Item Model)

```
const mongoose = require('mongoose');

const itemSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, required: true },
  description: String,
  category: String,
  imageUrl: String,
  available: { type: Boolean, default: true },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Item',
itemSchema);
```

models/order.js (Order Model)

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  userId: { type:
mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  items: [
    {
      itemId: { type:
mongoose.Schema.Types.ObjectId, ref: 'Item',
required: true },
      quantity: { type: Number, default: 1 }
    }
  ],
  totalAmount: { type: Number, required: true },
  status: { type: String, enum: ['pending',
'completed'], default: 'pending' },
  orderType: { type: String, enum: ['online', 'in-
store'], default: 'online' },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order',
orderSchema);
```

models/user.js (User Model)

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, unique: true, required: true
},
  password: { type: String, required: true },
  preferences: Array,
  createdAt: { type: Date, default: Date.now }
});

// Hash password before saving
userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  this.password = await
bcrypt.hash(this.password, 10);
  next();
});

module.exports = mongoose.model('User',
userSchema);
```

controllers/userController.js (User Controller)

```
const User = require('../models/user');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const dotenv = require('dotenv');
dotenv.config();

exports.register = async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const user = new User({ name, email,
password });
    await user.save();
    res.status(201).json({ message: 'User
registered successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};

exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user || !(await bcrypt.compare(password,
user.password))) {
      return res.status(401).json({ message:
'Invalid credentials' });
    }
    const token = jwt.sign({ userId: user._id },
process.env.JWT_SECRET);
    res.status(200).json({ token });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};
```

controllers/orderController.js (Order Controller)

```
const Order = require('../models/order');
const Item = require('../models/item');

exports.placeOrder = async (req, res) => {
  try {
    const { userId, items } = req.body;
    let totalAmount = 0;

    const itemDetails = await Promise.all(
      items.map(async (item) => {
        const itemData = await
Item.findById(item.itemId);
        totalAmount += itemData.price *
item.quantity;
        return { itemId: itemData._id, quantity:
item.quantity };
      })
    );

    const order = new Order({
      userId,
      items: itemDetails,
      totalAmount,
      orderType: 'online'
    });
    await order.save();

    res.status(201).json({ message: 'Order placed
successfully', order });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};
```

controllers/adminController.js (Admin Controller)

```
const Item = require('../models/item');
const Order = require('../models/order');

exports.addItem = async (req, res) => {
   try {
       const { name, price, description, category } = req.body;
       const item = new Item({ name, price, description, category });
       await item.save();
       res.status(201).json({ message: 'Item added successfully', item });
   } catch (error) {
       res.status(500).json({ message: 'Server error', error });
   }
};

exports.viewOrders = async (req, res) => {
   try {
       const orders = await Order.find().populate('items.itemId').populate('userId');
       res.status(200).json(orders);
   } catch (error) {
       res.status(500).json({ message: 'Server error', error });
   }
};
```

routes/userRoutes.js

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.post('/register', userController.register);
router.post('/login', userController.login);

module.exports = router;
```

routes/orderRoutes.js

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.post('/register', userController.register);
router.post('/login', userController.login);

module.exports = router;
```

routes/adminRoutes.js

```
const express = require('express');
const router = express.Router();
const adminController = require('../controllers/adminController');

router.post('/add-item', adminController.addItem);
router.get('/view-orders', adminController.viewOrders);

module.exports = router;
```

app.js

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');
const userRoutes = require('./routes/userRoutes');
const orderRoutes = require('./routes/orderRoutes');
const adminRoutes = require('./routes/adminRoutes');

dotenv.config();
const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true })
   .then(() => console.log('MongoDB connected'))
   .catch(err => console.log(err));

app.use('/api/users', userRoutes);
app.use('/api/orders', orderRoutes);
app.use('/api/admin', adminRoutes);

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

7. Sales Reporting Description: Generate and display sales insights for admins. • Features: o Aggregate and display sales data for selected

periods (daily, weekly, monthly). o Identify top-selling items and generate reports showing itemized and total sales. o Export reports in CSV or PDF formats for analysis.

CODE

```
npm init -y
npm install express mongoose body-parser cors
dotenv json2csv pdfkit moment
```

models/item.js (Item Model)

```
const mongoose = require('mongoose');

const itemSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, required: true },
  description: String,
  category: String,
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Item',
itemSchema);
```

models/order.js (Order Model)

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  items: [
    {
      itemId: { type:
mongoose.Schema.Types.ObjectId, ref: 'Item',
required: true },
      quantity: { type: Number, default: 1 }
    }
  ],
  totalAmount: { type: Number, required: true },
  status: { type: String, enum: ['completed',
'pending'], default: 'completed' },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order',
orderSchema);
```

controllers/salesController.js (Sales Controller)

```
const Order = require('../models/order');
const Item = require('../models/item');
const { Parser } = require('json2csv');
const PDFDocument = require('pdfkit');
const moment = require('moment');

// Get aggregated sales data
exports.getSalesData = async (req, res) => {
  try {
    const { period } = req.query; // 'daily',
'weekly', 'monthly'
    const matchStage = {};
    const groupStage = {
      _id: null,
      totalSales: { $sum: "$totalAmount" },
      totalOrders: { $sum: 1 },
    };

    // Define date range based on period
    let startDate;
    if (period === 'daily') {
      startDate = moment().startOf('day');
    } else if (period === 'weekly') {
      startDate = moment().startOf('week');
    } else if (period === 'monthly') {
      startDate = moment().startOf('month');
    }

    if (startDate) {
      matchStage.createdAt = { $gte:
startDate.toDate() };
    }

    const salesData = await Order.aggregate([
      { $match: matchStage },
      { $group: groupStage }
    ]);

    res.status(200).json(salesData[0] || {
totalSales: 0, totalOrders: 0 });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};

// Get top-selling items
exports.getTopSellingItems = async (req, res) => {
  try {
    const topItems = await Order.aggregate([
        { $unwind: "$items" },
        {
          $group: {
            _id: "$items.itemId",
            totalSold: { $sum: "$items.quantity" }
          }
        },
        { $sort: { totalSold: -1 } },
        { $limit: 10 },
        {
          $lookup: {
            from: 'items',
            localField: '_id',
            foreignField: '_id',
            as: 'itemDetails'
          }
        },
        { $unwind: "$itemDetails" },
        {
          $project: {
            _id: 0,
            itemName: "$itemDetails.name",
            totalSold: 1
          }
        }
      ]);

      res.status(200).json(topItems);
  } catch (error) {
      res.status(500).json({ message: 'Server error',
error });
  }
};

// Export sales data as CSV
exports.exportSalesCSV = async (req, res) => {
  try {
      const orders = await
Order.find().populate('items.itemId');
      const csvData = orders.map(order => ({
        orderId: order._id,
        totalAmount: order.totalAmount,
        createdAt: order.createdAt.toISOString(),
      }));

      const parser = new Parser();
      const csv = parser.parse(csvData);
      res.header('Content-Type', 'text/csv');
      res.attachment('sales_report.csv');
      res.send(csv);
  } catch (error) {
      res.status(500).json({ message: 'Server error',
error });
  }
};

// Export sales data as PDF
exports.exportSalesPDF = async (req, res) => {
  try {
      const orders = await
Order.find().populate('items.itemId');
      const doc = new PDFDocument();

      res.setHeader('Content-Type',
'application/pdf');
      res.setHeader('Content-Disposition',
'attachment; filename=sales_report.pdf');

      doc.text('Sales Report', { align: 'center' });
      doc.moveDown();

      orders.forEach(order => {
        doc.text(`Order ID: ${order._id}`);
        doc.text(`Total Amount:
$${order.totalAmount}`);
        doc.text(`Created At:
${order.createdAt.toISOString()}`);
        doc.moveDown();
      });

      doc.end();
      doc.pipe(res);
  } catch (error) {
      res.status(500).json({ message: 'Server error',
error });
  }
};
```

routes/salesRoutes.js (Routes)

```
const express = require('express');
const router = express.Router();
const salesController =
require('../controllers/salesController');

router.get('/data', salesController.getSalesData);
router.get('/top-items',
salesController.getTopSellingItems);
router.get('/export/csv',
salesController.exportSalesCSV);
router.get('/export/pdf',
salesController.exportSalesPDF);

module.exports = router;
```

app.js

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');
const salesRoutes =
require('./routes/salesRoutes');

dotenv.config();
const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGO_URI, {
useNewUrlParser: true, useUnifiedTopology: true
})
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));

app.use('/api/sales', salesRoutes);

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

8. Data Forecasting Using Time Series Forecasting Description: Leverage past sales data to predict future demand and aid inventory planning. • Features: o Collect historical sales data and structure it for forecasting. o Use time series techniques (moving average, exponential smoothing) for projections. o Display forecasts in graphs or tables, highlighting expected demand for specific periods

CODE

```
npm init -y
npm install express mongoose body-parser cors
dotenv
npm install mathjs
```

models/order.js (Order Model)

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  date: { type: Date, required: true },
  totalAmount: { type: Number, required: true },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order',
orderSchema);
```

controllers/forecastController.js (Forecast Controller)

```
const Order = require('../models/order');
const math = require('mathjs');

// Helper function: Moving Average
function movingAverage(data, windowSize) {
  const result = [];
  for (let i = 0; i <= data.length - windowSize;
i++) {
    const window = data.slice(i, i + windowSize);
    const avg = window.reduce((sum, value) =>
sum + value, 0) / windowSize;
    result.push(avg);
  }
  return result;
}

// Helper function: Exponential Smoothing
function exponentialSmoothing(data, alpha) {
  const result = [data[0]]; // Start with the first
data point
  for (let i = 1; i < data.length; i++) {
    const smoothValue = alpha * data[i] + (1 -
alpha) * result[i - 1];
    result.push(smoothValue);
  }
  return result;
}

// Collect and structure historical sales data
```

```
exports.getHistoricalSalesData = async (req, res)
=> {
  try {
    const orders = await Order.find().sort({ date:
1 });
    const salesData = orders.map(order => ({
      date: order.date,
      totalAmount: order.totalAmount
    }));
    res.status(200).json(salesData);
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};

// Generate sales forecast using Moving Average
exports.forecastMovingAverage = async (req, res)
=> {
  try {
    const { windowSize } = req.query;
    const orders = await Order.find().sort({ date:
1 });
    const sales = orders.map(order =>
order.totalAmount);

    const forecast = movingAverage(sales,
parseInt(windowSize));
    res.status(200).json({ method: 'Moving
Average', forecast });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};

// Generate sales forecast using Exponential
Smoothing
exports.forecastExponentialSmoothing = async
(req, res) => {
  try {
    const { alpha } = req.query;
    const orders = await Order.find().sort({ date:
1 });
    const sales = orders.map(order =>
order.totalAmount);
    const forecast = exponentialSmoothing(sales,
parseFloat(alpha));
    res.status(200).json({ method: 'Exponential
Smoothing', forecast });
  } catch (error) {
    res.status(500).json({ message: 'Server error',
error });
  }
};
```

routes/forecastRoutes.js (Routes)

```
const express = require('express');
const router = express.Router();
const forecastController =
require('../controllers/forecastController');

// Get historical sales data
router.get('/historical-data',
forecastController.getHistoricalSalesData);

// Forecast with Moving Average
router.get('/forecast/moving-average',
forecastController.forecastMovingAverage);

// Forecast with Exponential Smoothing
router.get('/forecast/exponential-smoothing',
forecastController.forecastExponentialSmoothing)
;

module.exports = router;
```

app.js

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');
const forecastRoutes =
require('./routes/forecastRoutes');

dotenv.config();
const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGO_URI, {
useNewUrlParser: true, useUnifiedTopology: true
})
  .then(() => console.log('MongoDB connected'))
```

```
    .catch(err => console.log(err));

app.use('/api/forecast', forecastRoutes);

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