app.js

```
require('dotenv').config();
const express = require('express');
const mongoose = require('mongoose');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

const app = express();
app.use(express.json());

/*Connecting to mongo
const mongoURL = "mongodb://localhost:27017/sg";
mongoose.connect(mongoURL, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useCreateIndex: true
})
.then(() => console.log('Connection Established'))
.catch(err => {
  console.error('Connection Failed:', err);
  process.exit(1);
});

// Schemas
const clientSchema = new mongoose.Schema({
  input: { type: String, required: true, trim: true },
  passwordHash: { type: String, required: true }
});

const Client = mongoose.model('Client', clientSchema);

const itemSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  price: { type: Number, required: true },
  information: { type: String, required: true, trim: true },
  totalCount: { type: Number, required: true, min: 0 }
});

const Item = mongoose.model('Item', itemSchema);

const checkoutSchema = new mongoose.Schema({
  managerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Client', required: true },
  items: [{
    itemRef: { type: mongoose.Schema.Types.ObjectId, ref: 'Item', required: true },
    cost: { type: Number, required: true }
  }]
});
```

```
const Checkout = mongoose.model('Checkout', checkoutSchema);

/*incoming requests*/
app.use((req, res, next) => {
  console.info(`Incoming Request => Method: [${req.method}] URL: ${req.originalUrl}`);
  next();
});

/*Authenticator*/
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  if (!authHeader) return res.status(401).json({ message: 'Authorization header missing' });

  const token = authHeader.split(' ')[1];
  if (!token) return res.status(401).json({ message: 'Token missing' });

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) return res.status(401).json({ message: 'Invalid token' });
    req.clientId = decoded.id;
    next();
  });
};

/*input new client*/
app.post('/auth/registerclient', async (req, res) => {
  try {
    const { input, password } = req.body;

    if (!(input && password)) {
      return res.status(400).json({ message: 'Input and password are required' });
    }

    const clientExists = await Client.findOne({ input });
    if (clientExists) {
      return res.status(409).json({ message: 'Input value already exists' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const newClient = new Client({ input, passwordHash: hashedPassword });
    await newClient.save();

    res.status(201).json({ message: 'Client registered successfully' });
  } catch (error) {
    res.status(500).json({ message: 'An error occurred during registration', error: error.message });
  }
});

/*login of a client*/
app.post('/auth/login', async (req, res) => {
  try {
    const { input, password } = req.body;
```

```
    if (!(input && password)) {
      return res.status(400).json({ message: 'Input and password are required' });
    }

    const client = await Client.findOne({ input });
    if (!client) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const isValidPassword = await bcrypt.compare(password, client.passwordHash);
    if (!isValidPassword) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const token = jwt.sign({ id: client._id }, process.env.JWT_SECRET, { expiresIn: '1h' });
    res.json({ token });
  } catch (error) {
    res.status(500).json({ message: 'An error occurred during login', error: error.message });
  }
});

app.get('/items', async (req, res) => {/*to get all item*/
  try {
    const items = await Item.find().exec();
    res.json(items);
  } catch (e) {
    res.status(500).json({ message: 'Error fetching items', e: e.message });
  }
});

app.get('/items/:pid', async (req, res) => {/*Get item by unique id*/
  try {
    const item = await Item.findById(req.params.pid).exec();
    if (!item) return res.status(404).json({ message: 'Item not found' });
    res.json(item);
  } catch (e) {
    res.status(400).json({ message: 'Invalid item ID', e: e.message });
  }
});

/*getting items for checkout*/
app.post('/cart', authenticateToken, async (req, res) => {
  try {
    const clientId = req.clientId;
    const { itemId, inputThings } = req.body;

    if (!(itemId && inputThings) || inputThings < 1) {
      return res.status(400).json({ message: 'Item ID and quantity are required' });
    }
```

```
    const itemExists = await Item.findById(itemId);
    if (!itemExists) return res.status(404).json({ message: 'Item not found' });

    let clientCheckout = await Checkout.findOne({ managerId: clientId });
    if (!clientCheckout) {
      clientCheckout = new Checkout({ managerId: clientId, items: [] });
    }

    const itemIndex = clientCheckout.items.findIndex(item => item.itemRef.toString() === itemId);
    if (itemIndex !== -1) {
      clientCheckout.items[itemIndex].cost += inputThings;
    } else {
      clientCheckout.items.push({ itemRef: itemId, cost: inputThings });
    }

    await clientCheckout.save();
    res.status(201).json(clientCheckout);
  } catch (error) {
    res.status(500).json({ message: 'Error adding item to checkout', error: error.message });
  }
});

app.put('/cart/:itemId', authenticateToken, async (req, res) => {/*to update items*/
  try {
    const clientId = req.clientId;
    const itemId = req.params.itemId;
    const { inputThings } = req.body;

    if (!inputThings || inputThings < 1) {
      return res.status(400).json({ message: 'Quantity is required' });
    }

    const clientCheckout = await Checkout.findOne({ managerId: clientId });
    if (!clientCheckout) return res.status(404).json({ message: 'Checkout not found' });

    const itemIndex = clientCheckout.items.findIndex(item => item.itemRef.toString() === itemId);
    if (itemIndex === -1) return res.status(404).json({ message: 'Item not found in checkout' });

    clientCheckout.items[itemIndex].cost = inputThings;
    await clientCheckout.save();
    res.json(clientCheckout);
  } catch (error) {
    res.status(500).json({ message: 'Error updating item quantity', error: error.message });
  }
});


app.delete('/checkout/:itemId', authenticateToken, async (req, res) => {/*To delete items*/
  try {
    const clientId = req.clientId;
    const itemId = req.params.itemId;
```

```
    const clientCheckout = await Checkout.findOne({ managerId: clientId });
    if (!clientCheckout) return res.status(404).json({ message: 'Checkout not found' });

    const updatedItems = clientCheckout.items.filter(item => item.itemRef.toString() !== itemId);
    if (updatedItems.length === clientCheckout.items.length) {
      return res.status(404).json({ message: 'Item not found in checkout' });
    }

    clientCheckout.items = updatedItems;
    await clientCheckout.save();
    res.json(clientCheckout);
  } catch (e) {
    res.status(500).json({ message: 'Error removing item from checkout', e: e.message });
  }
});


const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`hoisted at ${PORT}`);
});
```

Github Repo Link:

[Hitman-97/backendproject: In this file you will get all the required things to run the project precisely just checkout the js file]