Individuell uppgift Backend1

Frågor för betyget Godkänd (G)

1. Hur fungerar asynkron kommunikation i NodeJS, och varför är det viktigt?

Beskriv hur asynkron kod fungerar i NodeJS.
Ge ett exempel på en situation där asynkron hantering är används och varför man inte kan eller bör använda synkron hantering.

Svar

Asynkron kod betyder att programmet inte behöver vänta på att något ska bli klart, till exempel att läsa en fil eller hämta data.
Det kan fortsätta med annat under tiden.
Node.js använder asynkron kod för att vara snabb och effektiv, speciellt när man gör saker som tar tid (som att kommunicera med en databas).
Exempel:

```
console.log("Börjar läsa fil...");
fs.readFile("text.txt", "utf8", (err, data) => {
  console.log("Filen är klar! Innehåll:", data);
});
console.log("Under tiden gör vi något annat!");
```

Här börjar vi läsa filen, men istället för att vänta, körs nästa kod direkt.
När filen är klar, körs funktionen med innehållet (det är asynkront).

2. Varför används JSON inom webbutveckling, och vilka är dess fördelar och nackdelar?

Förklara varför JSON är ett standardformat för API:er.
Nämn fördelar och eventuella nackdelar med JSON jämfört med andra format tex csv och BSON.

Svar

JSON (JavaScript Object Notation) används för att skicka och ta emot data mellan webbplatser, appar och servrar.
Det är enkelt att läsa av.
Varför är JSON standardformat för API:er:
-JSON är lätt att förstå.
-Det funkar direkt i JavaScript (som är vanligt på webben).
-Det är snabbt att skriva och läsa.
-Stöds av nästan alla programmeringsspråk.

Fördelar med JSON	Nackdelar med JSON
Läsbart (ser ut som vanliga JavaScript-objekt)	Ingen stöd för binär data (till exempel bilder) då är BSON bättre
Enkelt att använda i både frontend och backend	Lite större filstorlek än till exempel CSV för enkla data
Bra för struktur (kan ha listor, objekt i objekt)	Kan inte ha kommentarer (som vissa andra format kan)
Stöd överallt (nästan alla API:er och verktyg stöder JSON)	

Format	Fördelar	Nackdelar
JSON	Lättläst, standard för API, flexibel	Inte bra för binär data, lite större
CSV	Enkel, liten filstorlek	Inga strukturer (bara tabeller)
BSON	Stöd för binär data, snabbare ibland	Svårare att läsa, större

3. Hur hanteras lösenord på ett säkert sätt i en NodeJS-applikation?

Förklara varför lösenord inte ska sparas i klartext.
Beskriv varför hashning och salting är viktigt.

Svar

Att lagra lösenord i klartext är en allvarlig säkerhetsrisk och om någon får tillgång till databasen kan de se alla användares lösenord direkt vilket medför följande risker:
-Många människor använder samma lösenord på flera sidor = stora säkerhetsrisker.
-Det är ett stort integritetsbrott för användarna.
Därför använder vi en process som kallas hashing och salting för att skydda lösenorden.
Hashing:
Omvandlar ett lösenord till en oigenkännlig sträng.
Det är en envägsprocess, vilket innebär att originallösenordet inte kan återskapas från den resulterande hash-strängen.
Exempel lösenordet **hemligt123** blir till **5f4dcc3b5aa765d61d8327deb882cf99** med hjälp av hashing.
Salting:
Lägger till en slumpmässig sträng (salt) innan lösenordet hashas.
Detta ser till att även identiska lösenord får unika hashade värden, vilket försvårar attacker ytterligare.

4. Hur fungerar enhetstester i NodeJS och varför är de viktiga?

Beskriv syftet med enhetstester och varför man använder Jest.
Ge ett exempel på ett Model test och vad som vi försöker testa.

Svar

Genom att testa din kod kan du verifiera att den fungerar som förväntat, undvika regressioner (det vill säga att nya buggar introduceras när man gör ändringar) och höja kvaliteten på din programvara.
Enhetstester (unit tests) är en typ av test som fokuserar på att testa små, isolerade delar av koden, till exempel enskilda funktioner eller metoder.
Varför enhetstester är viktiga:
Fångar upp buggar tidigt:
-Genom att testa din kod regelbundet kan du upptäcka fel tidigt i utvecklingsprocessen, vilket gör dem billigare och enklare att åtgärda.
Ökar tillförlitligheten:
-Tester ger dig en större säkerhet i att din kod fungerar som den ska, även när du gör ändringar i framtiden.
Underlättar underhåll:
-Tydliga tester gör det lättare att förstå vad koden gör och hur den ska bete sig, vilket underlättar underhåll och vidareutveckling.
Förbättrar design:
-Att skriva tester kan hjälpa dig att tänka igenom designen av din kod och skapa mer modulär och testbar kod.
Jest är ett populärt testverktyg i Node.js och används för att:
-Skriva tester enkelt.
-Få tydliga felmeddelanden.
-Köra tester snabbt.
-Testar både backend och frontend.
Exempel:
Du har en User-modell med en funktion som skapar en användare.
userModel.js

```
function createUser(name, age) {
 if (!name || age < 0) throw new Error("Ogiltiga värden");
 return { name, age };
}
module.exports = { createUser };
```

userModel.test.js

```
const { createUser } = require('./userModel');
test('Skapar en användare med namn och ålder', () => {
 const user = createUser('Anna', 25);
 expect(user).toEqual({ name: 'Anna', age: 25 });
});
test('Kastar fel om ålder är negativ', () => {
 expect(() => createUser('Anna', -5)).toThrow('Ogiltiga värden');
});
```

Vi testar följande med dessa koder:
-Att funktionen skapar rätt objekt med rätt data.
-Att den kastar fel om något är fel (till exempel negativ ålder minus 30år).

5. Vad är JWT och hur används det för autentisering?

Förklara vad en JSON Web Token (JWT) är.
Beskriv hur JWT kan användas för att skydda en API-route.
Beskriv vilka åtgärder som måste tas i frontend för att spara JWT.

Svar

JSON Web Token är en metod för att på ett säkert sätt autentisera och auktorisera användare i webbtjänster.
Den låter er skicka med signerad (och eventuell krypterad) information i en token, så att servern kan verifiera identiteten och rätten till åtkomst.
JWT skyddar en API-route genom följande sätt:
1, Användaren loggar in (backend skapar en JWT med till exempel användarens ID).
2, JWT skickas till frontend (frontend sparar token).
3, Vid varje förfrågan till ett skyddat API, skickas JWT med i headers = Authorization: Bearer <din-token>
4, Servern kontrollerar JWT (om den är giltig, släpps användaren in).
Exempel på skyddad route i Node.js:

```
const jwt = require('jsonwebtoken');
function verifyToken(req, res, next) {
 const token = req.headers.authorization?.split(' ')[1];
 if (!token) return res.status(401).json({ message: "Ingen token" });
 try {
  const decoded = jwt.verify(token, 'hemlig-nyckel');
  req.user = decoded;
  next();
 } catch {
  res.status(403).json({ message: "Ogiltig token" });
 }
}
// Skyddad route
app.get('/protected', verifyToken, (req, res) => {
 res.send("Du är inloggad!");
});
```

Frontend kan spara JWT på 2 vanliga metoder:
LocalStorage som är lätt att använda men mer sårbar för XSS-attacker (cross-site scripting).

HttpOnly Cookie är mer säker, skyddad från JavaScript (kan inte nås av skadlig kod) men kräver lite mer inställningar, men säkrare.

Punkt	Förklaring
Vad är JWT?	En säker token som visar vem användaren är.
Hur skyddar det API?	Användaren skickar token – servern kollar om den är giltig.
Hur spara JWT i frontend?	LocalStorage (enkelt) eller HttpOnly Cookie (säkrare).

Frågor för betyget Väl Godkänd (VG)

6. Scenario: Felhantering i en NodeJS-applikation

Du arbetar som backendutvecklare på ett företag som bygger en forumapplikation där användare kan skapa inlägg och kommentera på varandras inlägg.
Under utvecklingen har det visat sig att applikationen ibland kraschar eller returnerar oväntade fel.
Du har nu i uppgift att säkerställa att:
-All inkommande data valideras strikt innan den sparas eller uppdateras i databasen.
-Alla anrop till en specifik Post eller kommentar hanterar fall där resursen inte finns och svarar med ett tydligt 404meddelande.
-Ingen route får krascha applikationen, och vid fel ska API:t returnera tydliga och användbara felmeddelanden till frontend.
Nedan finns ett kodexempel.
Skriv det som best practice-kod där en erfaren utvecklare enkelt förstår flödet.
Använd gärna kommentarer, men bara om de tillför något.
Kodens fokus ska ligga på stabilitet, förutsägbarhet och användarvänlig felhantering.

```
const express = require('express');
const router = express.Router();
const Post = require('../models/Post');
router.post('/posts', async (req, res, next) => {
  const post = new Post({
    title: req.body.title,
    body: req.body.body
  });
  const savedPost = await post.save();
  res.status(201).json(savedPost);
});
router.delete('/posts/:id', async (req, res, next) => {
  const post = await Post.findByIdAndDelete(req.params.id);
  await Comment.deleteMany({ postId: post._id });
  res.json({ message: 'Inlägg och kommentarer borttagna' });
```

```
});
module.exports = router;
```

Svar

Här är en förbättrad version av koden enligt best practices för en erfaren Node.js-utvecklare, med fokus på:
-Validering av inkommande data.
-Kontroll om resurser finns (404 Not Found).
-Stabil felhantering (så att inget kraschar).
-Tydliga felmeddelanden till frontend.

```
/*
Importerar moduler och skapar routes
*/
const express = require('express');
const router = express.Router();
const Post = require('../models/Post');
const Comment = require('../models/Comment');
const mongoose = require('mongoose');

/*
Kolla om ett ID är ett giltigt MongoDB-id
*/
function isValidId(id) {
  return mongoose.Types.ObjectId.isValid(id);
}

/*
Skapa ett nytt inlägg
*/
router.post('/posts', async (req, res) => {
  try {
    const { title, body } = req.body;

/*
Kontrollera att användaren skickat med titel och text
*/
    if (!title || !body) {
      return res.status(400).json({ error: 'Du måste fylla i både titel och
innehåll' });
    }
    const post = new Post({ title, body });
    const savedPost = await post.save();

/*
Skickar tillbaka det sparade inlägget
*/
```

```
    res.status(201).json(savedPost);
  } catch (err) {

/*
Felhantering om något går fel
*/
    res.status(500).json({ error: 'Något gick fel när inlägget skulle sparas'
});
  }
});

/*
Ta bort ett inlägg och dess kommentarer
*/
router.delete('/posts/:id', async (req, res) => {
  try {
    const postId = req.params.id;

/*
Felhantering om ID inte är korrekt
*/
    if (!isValidId(postId)) {
      return res.status(400).json({ error: 'Felaktigt ID' });
    }

/*
Hitta och ta bort inlägget
*/
    const post = await Post.findByIdAndDelete(postId);

/*
Om inlägget inte finns
*/
    if (!post) {
      return res.status(404).json({ error: 'Inlägget finns inte' });
    }

/*
Ta bort alla kommentarer som hör till inlägget
*/
    await Comment.deleteMany({ postId: post._id });
    res.json({ message: 'Inlägget och dess kommentarer har tagits bort' });
  } catch (err) {
    res.status(500).json({ error: 'Något gick fel vid borttagning' });
  }
});
```

```
/*
Gör router tillgänglig för andra filer som vill använda den här modulen
*/
module.exports = router;
```

Tips för global felhantering kan läggas i app.js:

```
/*
Global felhanterare – fångar alla "next(err)"
*/
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    error: 'Något gick fel på servern',
    details: process.env.NODE_ENV === 'development' ? err.message : undefined
  });
});
```

Förbättringar som vi har gjort:
-Applikationen kraschar aldrig (alla fel fångas).
-Användaren får tydliga svar (till exempel "titel krävs", "ogiltigt ID").
-Loggar för felsökning.
-Skydd mot felaktiga ID och saknade resurser.

7. Scenario: Nytt krav från Hakim – Analytics-endpoint för affärsdata

Din projektägare, Hakim, har anlitat en dataanalytiker som behöver tillgång till viss affärsdata från företagets webbshop.
Analytikern ska dock inte arbeta i koden eller databasen direkt, utan vill istället få tillgång till informationen via ett nytt, dedikerat API-gränssnitt.
Hakim ber dig därför skapa en ny samling administrativa endpoints under /api/analytics som endast är tillgänglig för administratörer.

Krav på funktionalitet:

1. Månadsvis orderintäkt för senaste 12 månader - /api/analytics/revenue-per-month/
A, API:t ska returnera en lista med totala orderintäkter för varje månad bakåt i tiden, från nuvarande månad till exakt ett år tillbaka.
B, Exempel om anrop görs i april 2025:

{
"april-2025": 560000,
"mars-2025": 430000,
...
"april-2024": 38000
}

2. Topp 10 största kunder - /api/analytics/top-customers/
A, Returnerar en lista över de 5 kunder som har spenderat mest totalt i webbshopen.
3. Åtkomstskydd för administratörer
A, Endast användare med admin-behörighet ska kunna anropa dessa endpoints.
B, Om en icke-admin försöker få åtkomst ska API:t svara med en 404 Not Found, så att resursen inte avslöjas.

Kod Data

orders.js

```
const users = require("./users");
const products = require("./products");
const { faker } = require("@faker-js/faker");
const generateDates = (completed = false, cancelled = false) => {
  const today = new Date();
  const lastYear = new Date(today.getFullYear() - 1, today.getMonth(),
today.getDate());
  const purchasedAt = faker.date.between({
    from: lastYear,
    to: today,
  });
  let twoWeeksAfterPurchased = new Date(purchasedAt.getTime() + 14 * 24 * 60 *
60 * 1000);
  if (twoWeeksAfterPurchased > today) {
    twoWeeksAfterPurchased = today;
  }
  const completedAt = completed ? faker.date.between({
    from: purchasedAt,
    to: twoWeeksAfterPurchased,
  }) : null;
  const cancelledAt = cancelled ? faker.date.between({
    from: purchasedAt,
    to: twoWeeksAfterPurchased,
  }) : null;
  return { purchasedAt, completedAt, cancelledAt };
};
const generateOrders = () => {
  const orders = [
    {
      user: users[0]._id,
      products: [
        { product: products[0]._id, quantity: 2 },
        { product: products[3]._id, quantity: 1 },
      ],
      status: "completed",
```

```
    paymentMethod: "credit_card",
    paymentStatus: "completed",
    notes: "Order #1 - Tomatoes and Greek Yogurt",
    totalPrice: (2 * products[0].price) + (1 * products[3].price),
    ...generateDates(true),
  },
  {
    user: users[1]._id,
    products: [
      { product: products[1]._id, quantity: 1 },
    ],
    status: "pending",
    paymentMethod: "paypal",
    paymentStatus: "pending",
    notes: "Order #2 - Bananas",
    totalPrice: 1 * products[1].price,
    ...generateDates(),
  },
  {
    user: users[2]._id,
    products: [
      { product: products[2]._id, quantity: 3 },
    ],
    status: "completed",
    paymentMethod: "bank_transfer",
    paymentStatus: "completed",
    notes: "Order #3 - Green Apples",
    totalPrice: 3 * products[2].price,
    ...generateDates(true),
  },
  {
    user: users[3]._id,
    products: [
      { product: products[4]._id, quantity: 1 },
      { product: products[0]._id, quantity: 1 },
    ],
    status: "completed",
    paymentMethod: "credit_card",
    paymentStatus: "completed",
    notes: "Order #4 - Ground Beef and Tomatoes",
    totalPrice: (1 * products[4].price) + (1 * products[0].price),
    ...generateDates(true),
  },
  {
    user: users[4]._id,
    products: [
      { product: products[3]._id, quantity: 2 },
```

```
      ],
      status: "cancelled",
      paymentMethod: "credit_card",
      paymentStatus: "failed",
      notes: "Order #5 - Greek Yogurt (cancelled)",
      totalPrice: 2 * products[3].price,
      ...generateDates(false, true),
    },
    {
      user: users[5]._id,
      products: [
        { product: products[2]._id, quantity: 1 },
        { product: products[1]._id, quantity: 2 },
      ],
      status: "completed",
      paymentMethod: "debit_card",
      paymentStatus: "completed",
      notes: "Order #6 - Green Apples and Bananas",
      totalPrice: (1 * products[2].price) + (2 * products[1].price),
      ...generateDates(true),
    },
  ];
  orders.forEach(order => {
    order.totalPrice = Number(order.totalPrice.toFixed(2));
  });
  return orders;
};
module.exports = generateOrders();
```

products.js

```
const products = [
    {
      _id: "67ed981187368c7bb51b1b00",
      title: "Tomatoes",
      description: "Fresh and organic tomatoes.",
      category: "Vegetables",
      price: 24.99,
      stock: 120,
      unit: "kg",
      image: "/images/tomatoes.jpg",
    },
    {
      _id: "67ed981187368c7bb51b1b01",
      title: "Bananas",
      description: "Sweet bananas from Ecuador.",
      category: "Fruits",
      price: 19.99,
      stock: 200,
```

```
    unit: "kg",
    image: "/images/bananas.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b02",
    title: "Green Apples",
    description: "Crisp and sour green apples.",
    category: "Fruits",
    price: 25.99,
    stock: 150,
    unit: "kg",
    image: "/images/green-apples.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b03",
    title: "Greek Yogurt",
    description: "Thick and creamy Greek yogurt.",
    category: "Dairy",
    price: 39.99,
    stock: 90,
    unit: "pcs",
    image: "/images/greek-yogurt.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b04",
    title: "Ground Beef",
    description: "Lean ground beef 5%.",
    category: "Meat",
    price: 89.99,
    stock: 60,
    unit: "kg",
    image: "/images/ground-beef.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b05",
    title: "Milk",
    description: "Full-fat milk 1L.",
    category: "Dairy",
    price: 14.99,
    stock: 100,
    unit: "liters",
    image: "/images/milk.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b06",
    title: "Basmati Rice",
    description: "Basmati rice 5kg bag.",
```

```
    category: "Bakery",
    price: 59.99,
    stock: 40,
    unit: "pcs",
    image: "/images/rice.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b07",
    title: "Chicken Breast",
    description: "Skinless chicken breasts.",
    category: "Meat",
    price: 79.99,
    stock: 70,
    unit: "kg",
    image: "/images/chicken.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b08",
    title: "Butter",
    description: "Real creamy butter.",
    category: "Dairy",
    price: 29.99,
    stock: 90,
    unit: "pcs",
    image: "/images/butter.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b09",
    title: "Broccoli",
    description: "Fresh green broccoli.",
    category: "Vegetables",
    price: 15.99,
    stock: 130,
    unit: "kg",
    image: "/images/broccoli.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b0a",
    title: "Cheddar Cheese",
    description: "Aged cheddar cheese block.",
    category: "Dairy",
    price: 22.99,
    stock: 100,
    unit: "pcs",
    image: "/images/cheddar.jpg",
  },
  {
```

```
    _id: "67ed981187368c7bb51b1b0b",
    title: "Orange Juice",
    description: "Fresh squeezed orange juice.",
    category: "Beverages",
    price: 49.99,
    stock: 60,
    unit: "liters",
    image: "/images/orange-juice.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b0c",
    title: "Bread Loaf",
    description: "Whole grain sandwich bread.",
    category: "Bakery",
    price: 18.99,
    stock: 80,
    unit: "pcs",
    image: "/images/bread.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b0d",
    title: "Eggs",
    description: "Organic brown eggs (12-pack).",
    category: "Dairy",
    price: 34.99,
    stock: 75,
    unit: "pcs",
    image: "/images/eggs.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b0e",
    title: "Lettuce",
    description: "Fresh iceberg lettuce.",
    category: "Vegetables",
    price: 12.99,
    stock: 50,
    unit: "kg",
    image: "/images/lettuce.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b0f",
    title: "Olive Oil",
    description: "Extra virgin olive oil.",
    category: "Pantry",
    price: 69.99,
    stock: 45,
    unit: "liters",
```

```
    image: "/images/olive-oil.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b10",
    title: "Tomato Sauce",
    description: "Rich tomato sauce.",
    category: "Pantry",
    price: 24.99,
    stock: 100,
    unit: "pcs",
    image: "/images/tomato-sauce.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b11",
    title: "Strawberries",
    description: "Fresh picked strawberries.",
    category: "Fruits",
    price: 35.99,
    stock: 60,
    unit: "kg",
    image: "/images/strawberries.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b12",
    title: "Oats",
    description: "Rolled oats for breakfast.",
    category: "Pantry",
    price: 27.99,
    stock: 120,
    unit: "pcs",
    image: "/images/oats.jpg",
  },
  {
    _id: "67ed981187368c7bb51b1b13",
    title: "Salmon Fillet",
    description: "Fresh Norwegian salmon fillet.",
    category: "Meat",
    price: 129.99,
    stock: 50,
    unit: "kg",
    image: "/images/salmon.jpg",
  },
];
module.exports = products;
```

users.js

```
const users = [
```

```
{
  _id: "67ed981187368c7bb51b1aa1",
  firstName: "Ali",
  lastName: "Hakim",
  email: "ali.hakim@example.com",
  password: "admin123",
  role: "admin",
},
{
  _id: "67ed981187368c7bb51b1aa2",
  firstName: "Sara",
  lastName: "Lind",
  email: "sara.lind@example.com",
  password: "user123",
  role: "user",
},
{
  _id: "67ed981187368c7bb51b1aa3",
  firstName: "Mikael",
  lastName: "Svensson",
  email: "mikael.svensson@example.com",
  password: "user123",
  role: "user",
},
{
  _id: "67ed981187368c7bb51b1aa4",
  firstName: "Fatima",
  lastName: "Yilmaz",
  email: "fatima.yilmaz@example.com",
  password: "user123",
  role: "user",
},
{
  _id: "67ed981187368c7bb51b1aa5",
  firstName: "Omar",
  lastName: "Ali",
  email: "omar.ali@example.com",
  password: "user123",
  role: "user",
},
{
  _id: "67ed981187368c7bb51b1aa6",
  firstName: "Emma",
  lastName: "Karlsson",
  email: "emma.karlsson@example.com",
  password: "user123",
  role: "user",
```

```
  },
  {
    _id: "67ed981187368c7bb51b1aa7",
    firstName: "David",
    lastName: "Nguyen",
    email: "david.nguyen@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aa8",
    firstName: "Isabella",
    lastName: "Persson",
    email: "isabella.persson@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aa9",
    firstName: "Lucas",
    lastName: "Johansson",
    email: "lucas.johansson@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aaa",
    firstName: "Nora",
    lastName: "Andersson",
    email: "nora.andersson@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aab",
    firstName: "Leo",
    lastName: "Lindberg",
    email: "leo.lindberg@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aac",
    firstName: "Julia",
    lastName: "Eriksson",
    email: "julia.eriksson@example.com",
    password: "user123",
```

```
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aad",
    firstName: "Max",
    lastName: "Nilsson",
    email: "max.nilsson@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aae",
    firstName: "Hanna",
    lastName: "Fransson",
    email: "hanna.fransson@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1aaf",
    firstName: "William",
    lastName: "Sjöberg",
    email: "william.sjoberg@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1ab0",
    firstName: "Alma",
    lastName: "Berg",
    email: "alma.berg@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1ab1",
    firstName: "Noah",
    lastName: "Pettersson",
    email: "noah.pettersson@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1ab2",
    firstName: "Freja",
    lastName: "Holm",
    email: "freja.holm@example.com",
```

```
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1ab3",
    firstName: "Melvin",
    lastName: "Axelsson",
    email: "melvin.axelsson@example.com",
    password: "user123",
    role: "user",
  },
  {
    _id: "67ed981187368c7bb51b1ab4",
    firstName: "Tilde",
    lastName: "Wallin",
    email: "tilde.wallin@example.com",
    password: "user123",
    role: "user",
  },
];
module.exports = users;
```

Kod Middleware

admin.js

```
module.exports = function (req, res, next) {
  if (!req.user || req.user.role !== "admin") {
    return res.status(404).json({ message: "Not Found" });
  }
  next();
};
```

auth.js

```
const jwt = require("jsonwebtoken");
module.exports = (req, res, next) => {
  try {
    const token = req.header("Authorization").replace("Bearer ", "");
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    res.status(404).json({ message: "Not Found" });
  }
};
```

Kod Models

Order.js

```
const mongoose = require("mongoose");
const orderSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
  products: [
    {
      product: { type: mongoose.Schema.Types.ObjectId, ref: "Product" },
      quantity: { type: Number, default: 1 }
    }
  ],
  totalPrice: { type: Number, required: true },
  status: { type: String, enum: ["pending", "completed", "cancelled"],
default: "pending" },
  paymentMethod: { type: String, required: true },
  paymentStatus: { type: String, enum: ["pending", "completed", "failed"],
default: "pending" },

  purchasedAt: { type: Date, default: Date.now },
  completedAt: { type: Date },
  cancelledAt: { type: Date },
  notes: { type: String, trim: true }
});
orderSchema.pre("save", function (next) {
  if (this.isModified("status")) {
    if (this.status === "completed") this.completedAt = Date.now();
    else if (this.status === "cancelled") this.cancelledAt = Date.now();
  }
  next();
});
module.exports = mongoose.model("Order", orderSchema);
```

Product.js

```
const mongoose = require("mongoose");
const productSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  description: { type: String, required: true },
  category: { type: String, required: true },
  price: { type: Number, required: true, min: 0 },
  stock: { type: Number, required: true },
  tags: [{ type: String }],
  rating: { type: Number, min: 0, max: 5, default: 0 }
}, { timestamps: true });
module.exports = mongoose.model("Product", productSchema);
```

User.js

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

```
const bcrypt = require("bcryptjs");
const userSchema = new mongoose.Schema({
  firstName: { type: String, required: true, trim: true },
  lastName: { type: String, required: true, trim: true },
  email: { type: String, required: true, trim: true, lowercase: true },
  password: { type: String, required: true },
  role: {
    type: String,
    enum: ["user", "admin"],
    default: "user",
  },
  lastLogin: { type: Date }
}, { timestamps: true });
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);
```

Kod Routes

analyctics.js

```
const express = require("express");
const router = express.Router();
const Order = require("../models/Order");
const auth = require("../middleware/auth");
const adminAuth = require("../middleware/admin");
router.get("/revenue-per-month", auth, adminAuth, async (req, res) => {
  try {
    const oneYearAgo = new Date();
    oneYearAgo.setFullYear(oneYearAgo.getFullYear() - 1);
    const result = await Order.aggregate([
      {
        $match: {
          purchasedAt: { $gte: oneYearAgo },
          paymentStatus: "completed",
        },
      },
      {
        $group: {
          _id: {
            year: { $year: "$purchasedAt" },
            month: { $month: "$purchasedAt" },
          },
          totalRevenue: { $sum: "$totalPrice" },
        },
```

```
      },
      { $sort: { "_id.year": -1, "_id.month": -1 } },
    ]);
    const months = [
      "januari", "februari", "mars", "april", "maj", "juni",
      "juli", "augusti", "september", "oktober", "november", "december"
    ];
    const formatted = {};
    result.forEach((item) => {
      const label = `${months[item._id.month - 1]}-${item._id.year}`;
      formatted[label] = Number(item.totalRevenue.toFixed(2));
    });
    res.json(formatted);
  } catch (err) {
    res.status(500).json({ error: "Kunde inte hämta intäktsdata" });
  }
});
router.get("/top-customers", auth, adminAuth, async (req, res) => {
  try {
    const result = await Order.aggregate([
      { $match: { paymentStatus: "completed" } },
      {
        $group: {
          _id: "$user",
          totalSpent: { $sum: "$totalPrice" },
          orders: { $sum: 1 },
        },
      },
      { $sort: { totalSpent: -1 } },
      { $limit: 5 },
    ]);
    res.json(result);
  } catch (err) {
    res.status(500).json({ error: "Kunde inte hämta kunddata" });
  }
});
module.exports = router;
```

auth.js

```
const express = require("express");
const router = express.Router();
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const User = require("../models/User");
router.post("/register", async (req, res) => {
  try {
    const { firstName, lastName, email, password, role, school, subjects } =
req.body;
```

```
    if (!["user", "admin"].includes(role)) {
      return res.status(400).json({ message: "Invalid role" });
    }
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: "User already exists" });
    }
    const user = new User({
      firstName,
      lastName,
      email,
      password,
      role,
      school,
      subjects,
    });
    await user.save();
    const userData = {
      id: user._id,
      firstName: user.firstName,
      lastName: user.lastName,
      email: user.email,
      role: user.role,
      school: user.school,
      subjects: user.subjects,
    };
    if (user.role === "admin") {
      const token = jwt.sign(
        { userId: user._id, role: user.role },
        process.env.JWT_SECRET,
        { expiresIn: "24h" }
      );

      return res.status(201).json({
        message: "User registered successfully",
        token,
        user: userData,
      });
    }
    res.status(201).json({
      message: "User registered successfully",
      user: userData,
    });
  } catch (error) {
    res.status(500).json({ message: "Error registering user", error:
error.message });
  }
```

```
});
router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: "Invalid credentials"
});
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: "Invalid credentials"
});

    const userData = {
      id: user._id,
      firstName: user.firstName,
      lastName: user.lastName,
      email: user.email,
      role: user.role,
      school: user.school,
      subjects: user.subjects,
    };
    if (user.role === "admin") {
      const token = jwt.sign(
        { userId: user._id, role: user.role },
        process.env.JWT_SECRET,
        { expiresIn: "24h" }
      );
      return res.json({
        message: "Login successful",
        token,
        user: userData,
      });
    }
    res.json({
      message: "Login successful (no token for regular users)",
      user: userData,
    });
  } catch (error) {
    res.status(500).json({ message: "Error logging in", error: error.message
});
  }
});
module.exports = router;
```

orders.js

```
const express = require("express");
const router = express.Router();
const Order = require("../models/Order");
const Product = require("../models/Product");
```

```
const auth = require("../middleware/auth");
router.get("/", auth, async (req, res) => {
  try {
    const orders = await Order.find().populate("products.product", "name
price");
    res.json(orders);
  } catch (error) {
    res.status(500).json({ message: "Error fetching orders", error:
error.message });
  }
});
router.get("/my-orders", auth, async (req, res) => {
  try {
    const orders = await Order.find({ user: req.user.userId
}).populate("products.product", "name price");
    res.json(orders);
  } catch (error) {
    res.status(500).json({ message: "Error fetching user orders", error:
error.message });
  }
});
router.post("/", auth, async (req, res) => {
  try {
    const { products, paymentMethod } = req.body;
    const productDocs = await Promise.all(products.map(async ({ product,
quantity }) => {
      const found = await Product.findById(product);
      return found ? found.price * quantity : 0;
    }));
    const totalPrice = productDocs.reduce((sum, price) => sum + price, 0);
    const order = new Order({
      user: req.user.userId,
      products,
      totalPrice,
      status: "pending",
      paymentMethod,
      paymentStatus: "pending",
      purchasedAt: new Date(),
    });
    await order.save();
    res.status(201).json(order);
  } catch (error) {
    res.status(500).json({ message: "Error creating order", error:
error.message });
  }
});
module.exports = router;
```

products.js

```
const express = require("express");
const router = express.Router();
const Product = require("../models/Product");
const auth = require("../middleware/auth");
router.get("/", async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (error) {
    res.status(500).json({ message: "Error fetching products", error:
error.message });
  }
});
router.get("/:id", async (req, res) => {
  try {
    const product = await Product.findById(req.params.id);
    if (!product) return res.status(404).json({ message: "Product not found"
});
    res.json(product);
  } catch (error) {
    res.status(500).json({ message: "Error fetching product", error:
error.message });
  }
});
router.post("/", auth, async (req, res) => {
  try {
    const product = new Product(req.body);
    await product.save();
    res.status(201).json(product);
  } catch (error) {
    res.status(500).json({ message: "Error creating product", error:
error.message });
  }
});
router.put("/:id", auth, async (req, res) => {
  try {
    const updated = await Product.findByIdAndUpdate(req.params.id, req.body, {
new: true });
    if (!updated) return res.status(404).json({ message: "Product not found"
});
    res.json(updated);
  } catch (error) {
    res.status(500).json({ message: "Error updating product", error:
error.message });
  }
});
```

```
router.delete("/:id", auth, async (req, res) => {
  try {
    const deleted = await Product.findByIdAndDelete(req.params.id);
    if (!deleted) return res.status(404).json({ message: "Product not found"
});
    res.json({ message: "Product deleted successfully" });
  } catch (error) {
    res.status(500).json({ message: "Error deleting product", error:
error.message });
  }
});
module.exports = router;
```

Kod

seed.js

```
const Course = require("../models/Product");
const User = require("../models/User");
const Order = require("../models/Order");
const courses = require("../data/products");
const users = require("../data/users");
const orders = require("../data/orders");
const bcrypt = require("bcryptjs");
const seedDatabase = async () => {
  try {
    const courseCount = await Course.countDocuments();
    if (courseCount === 0) {
      console.log("No courses found in database. Seeding initial courses...");
      await Course.insertMany(courses);
      console.log("Successfully seeded courses");
    } else {
      console.log(`Database already contains ${courseCount} courses`);
    }
    const userCount = await User.countDocuments();
    if (userCount === 0) {
      console.log("No users found in database. Seeding initial users...");
      const hashedUsers = await Promise.all(
        users.map(async (user) => {
          const hashedPassword = await bcrypt.hash(user.password, 10);
          return { ...user, password: hashedPassword };
        })
      );
      await User.insertMany(hashedUsers);
      console.log("Successfully seeded users");
    } else {
      console.log(`Database already contains ${userCount} users`);
    }
```

```
    const orderCount = await Order.countDocuments();
    if (orderCount === 0) {
      console.log("No orders found in database. Seeding initial orders...");
      await Order.insertMany(orders);
      console.log("Successfully seeded orders");
    } else {
      console.log(`Database already contains ${orderCount} orders`);
    }
    return {
      coursesAdded: courseCount === 0 ? courses.length : 0,
      usersAdded: userCount === 0 ? users.length : 0,
      ordersAdded: orderCount === 0 ? orders.length : 0,
    };
  } catch (error) {
    console.error("Error seeding database:", error);
    throw error;
  }
};
const wipeAndReseed = async () => {
  try {
    await Course.deleteMany({});
    await User.deleteMany({});
    await Order.deleteMany({});
    const response = await seedDatabase();
    return response;
  } catch (error) {
    console.error("Error wiping database:", error);
    throw error;
  }
};
module.exports = {
  seedDatabase,
  wipeAndReseed,
};
```

.env

```
PORT=8081
MONGODB_URI=mongodb://localhost:27017/coreacademy
JWT_SECRET=JWT secret key
```

server.js

```
require("dotenv").config();
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const { seedDatabase, wipeAndReseed } = require("./utils/seed");
const authRoutes = require("./routes/auth");
const productRoutes = require("./routes/products");
```

```
const orderRoutes = require("./routes/orders");
const analyticsRoutes = require("./routes/analytics");
const app = express();
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use("/api/auth", authRoutes);
app.use("/api/products", productRoutes);
app.use("/api/orders", orderRoutes);
app.use("/api/analytics", analyticsRoutes);
app.get("/api/", (req, res) => {
  res.json({ message: "Welcome to Hakim Livs" });
});
app.post("/api/wipe", async (req, res) => {
  try {
    const result = await wipeAndReseed();
    res.json({
      message: "Database wiped and reseeded successfully",
      ...result,
    });
  } catch (error) {
    res.status(500).json({ error: "Failed to wipe and reseed database" });
  }
});
mongoose
  .connect(process.env.MONGODB_URI || "mongodb://localhost:27017/core-
academy")
  .then(async () => {
    console.log("Connected to MongoDB");
    await seedDatabase();
  })
  .catch((err) => console.error("MongoDB connection error:", err));
const PORT = process.env.PORT || 8081;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```