Backend Workshop 7

1) Express.js'te Middleware'in açıklamasını yapın.

```
const express = require('express');
const app = express();

// Örnek bir middleware fonksiyonu
const myMiddleware = (req, res, next) => {
  console.log('Bu middleware fonksiyonu çalıştı!');
  // Middleware işlemleri burada yapılır
  next(); // Bir sonraki middleware'e geçiş
};

// Uygulamaya middleware'i ekleme
app.use(myMiddleware);

// Route tanımı
app.get('/', (req, res) => {
  res.send('Merhaba Express!');
});

// Server'ı dinleme
const port = 3000;
app.listen(port, () => {
  console.log(`Uygulama ${port} portunda çalışıyor.`);
});
```

Express.js'te Middleware fonksiyonları, istek ve cevap nesnelerine erişime sahip olan ve bunları değiştirebilen fonksiyonlardır. Bunlar, günlükleme, kimlik doğrulama, istek verilerini ayrıştırma ve daha fazlası gibi görevleri yerine getirmek için kullanılır. Middleware fonksiyonları, uygulamada tanımlandıkları sırayla çalıştırılır.

2) Express.js'te routing nedir ve nasıl uygulanır?

```
const express = require('express');
const app = express();

// Ana sayfa route'u
app.get('/', (req, res) => {
  res.send('Ana sayfa');
});

// Hakkında sayfası route'u
app.get('/hakkinda', (req, res) => {
  res.send('Hakkında sayfası');
});

// İletişim sayfası route'u
app.get('/iletisim', (req, res) => {
  res.send('İletişim sayfası');
});

// Dinamik parametre kullanımı
app.get('/kullanici/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`Kullanıcı ID: ${userId}`);
});

// HTTP POST isteğini işleme
app.post('/api/kullanici', (req, res) => {
  // POST verilerini işleme
  res.send('Kullanıcı oluşturuldu');
});

// Server'ı dinleme
const port = 3000;
app.listen(port, () => {
  console.log(`Uygulama ${port} portunda çalışıyor.`);
});
```

Express.js'te routing, bir uygulamanın belirli bir uç noktaya (URL) gelen bir isteğe nasıl yanıt vereceğini belirleme sürecini ifade eder. Express'te, uygulamanın hangi HTTP yöntemine ve URL desenine nasıl yanıt vereceğini belirtmek için app.get(), app.post(), app.put() ve app.delete() yöntemlerini kullanarak rotaları tanımlayabilirsiniz.

3) Express.js'te hata işleme nasıl çalışır?

```
const express = require('express');
const app = express();

// Örnek bir route, hata fırlatma
app.get('/hataOrnegi', (req, res, next) => {
  // Örnek bir hata fırlatma
  const err = new Error('Bu bir örnek hatadır');
  err.status = 500; // Hata durumu belirleme
  next(err); // Hata nesnesini bir sonraki middleware'e iletimi
});

// Hata işleme middleware'i
app.use((err, req, res, next) => {
  // Hatanın durum koduna göre cevap verme
  res.status(err.status || 500);

  // Hata mesajını ve durumu gönderme
  res.json({
    error: {
      message: err.message,
      status: err.status || 500,
    },
  });
});

// Server'ı dinleme
const port = 3000;
app.listen(port, () => {
  console.log(`Uygulama ${port} portunda çalışıyor.`);
});
```

Express.js, middleware kullanarak yerleşik bir hata işleme mekanizması sağlar. Hata işleme middleware fonksiyonlarını tanımlayabilir ve bu fonksiyonlar dört parametre alır (err, req, res, next). Uygulamanızda bir hata oluştuğunda, next(err) çağrısı ile hatayı hata işleme middleware'ine iletebilirsiniz; burada hatayı uygun şekilde kaydedebilir, biçimlendirebilir veya yanıtlayabilirsiniz.

4) Express.js app.use() yönteminin amacı nedir?

```
const express = require('express');
const app = express();

// Middleware fonksiyonu
const myMiddleware = (req, res, next) => {
  console.log('Bu middleware fonksiyonu her istekte çalışacak!');
  next();
};

// Middleware'i uygulamaya ekleme
app.use(myMiddleware);

// Route tanımı
app.get('/', (req, res) => {
  res.send('Merhaba Express!');
});

// Server'ı dinleme
const port = 3000;
app.listen(port, () => {
  console.log(`Uygulama ${port} portunda çalışıyor.`);
});
```

app.use() yöntemi, Express.js'te middleware fonksiyonlarını monte etmek için kullanılır. Herhangi bir isteğe, HTTP yöntemine veya URL desenine bakılmaksızın her istek için çalıştırılması gereken middleware'ı belirtmek için kullanılabilir. Genellikle CORS başlıklarını ayarlamak, statik dosyaları sunmak veya oturumları yönetmek gibi görevler için kullanılır.

5) Express.js response nesnesinin rolünü açıklayın.

```
const express = require('express');
const app = express();

// Temel route tanımı
app.get('/', (req, res) => {
  // Metin cevabı gönderme
  res.send('Merhaba Express!');
});

// JSON cevabı gönderme
app.get('/api/data', (req, res) => {
  const data = { message: 'Express.js JSON Response' };
  res.json(data);
});

// Başlık ekleme
app.get('/header-example', (req, res) => {
  res.setHeader('Content-Type', 'text/plain');
  res.send('Bu bir örnek metindir.');
});

// Durum kodu ve özel başlık kullanımı
app.get('/status-example', (req, res) => {
  res.status(404).send('Sayfa bulunamadı');
});

// Server'ı dinleme
const port = 3000;
app.listen(port, () => {
  console.log(`Uygulama ${port} portunda çalışıyor.`);
});
```

Express.js'teki cevap nesnesi (res), HTTP yanıtlarını istemciye göndermek için kullanılır. Cevap başlıklarını ayarlamak, veriyi istemciye göndermek ve HTTP durum kodlarıyla ilgilenmek için yöntemler sağlar. Örneğin, res.send(), res.json() veya res.status() gibi yöntemleri kullanarak çeşitli türlerde yanıtları istemciye gönderebilirsiniz.

6) Sequelize nedir ve ana amacı nedir?

```
const { Sequelize, DataTypes } = require('sequelize');

// Sequelize ile bir SQLite veritabanı bağlantısı oluşturma
const sequelize = new Sequelize('sqlite::memory:');

// Kullanıcı modeli tanımlama
const User = sequelize.define('User', {
  firstName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  lastName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
});

// Veritabanını senkronize etme
sequelize.sync()
  .then(() => {
    // Kullanıcı oluşturma
    return User.create({
      firstName: 'John',
      lastName: 'Doe',
    });
  })
  .then((john) => {
    console.log(john.toJSON());
  });
```

Sequelize, Node.js için bir Nesne-İlişkisel Eşleme (ORM) kütüphanesidir ve ilişkisel veritabanlarıyla etkileşimleri basitleştirir. JavaScript nesneleri kullanarak veritabanı işlemlerini gerçekleştirmenizi sağlar ve veritabanlarıyla çalışmanın yapılandırılmış bir yolunu sunar.

7) Sayfalama nedir ve Sequelize veya Mongoose ile veritabanı sorgusunda nasıl uygulanır?

Sayfalama, sorgu sonuçlarını yönetilebilir parçalara bölmek anlamına gelir. Sequelize'de sayfalama için limit ve offset seçeneklerini kullanabilirsiniz. Mongoose'da sayfalama için skip() ve limit() gibi yöntemleri kullanabilirsiniz.

```sequelize
// Kullanıcı modeli tanımlama
const User = sequelize.define('User', {
  firstName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  lastName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
});

// Veritabanını senkronize etme
sequelize.sync()
  .then(() => {
    // Kullanıcıları oluşturma
    return User.bulkCreate([
      { firstName: 'John', lastName: 'Doe' },
      { firstName: 'Jane', lastName: 'Doe' },
      { firstName: 'Bob', lastName: 'Smith' },
      // ... daha fazla kullanıcı
    ]);
  })
  .then(() => {
    // Sayfalama için kullanıcıları sorgulama
    const page = 1;
    const perPage = 2;
    const offset = (page - 1) * perPage;

    return User.findAll({
      limit: perPage,
      offset: offset,
    });
  })
  .then((users) => {
    console.log(users);
  });
```

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

// Mongoose ile bir MongoDB veritabanı bağlantısı oluşturma
mongoose.connect('mongodb://localhost:27017/mydatabase', { useNewUrlParser: true, use

// Kullanıcı şeması tanımlama
const userSchema = new mongoose.Schema({
  firstName: String,
  lastName: String,
});

// Kullanıcı modeli oluşturma
const User = mongoose.model('User', userSchema);

// Kullanıcıları oluşturma
User.create([
  { firstName: 'John', lastName: 'Doe' },
  { firstName: 'Jane', lastName: 'Doe' },
  { firstName: 'Bob', lastName: 'Smith' },
  // ... daha fazla kullanıcı
])
  .then(() => {
    // Sayfalama için kullanıcıları sorgulama
    const page = 1;
    const perPage = 2;

    return User.find()
      .skip((page - 1) * perPage)
      .limit(perPage);
  })
  .then((users) => {
    console.log(users);
  });
```

8) Sequelize'de 'where' ifadesini nasıl kullanarak arama sorguları oluştururursunuz?

```
const User = sequelize.define('User', {
  firstName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  lastName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  age: {
    type: DataTypes.INTEGER,
  },
});

// Veritabanını senkronize etme
sequelize.sync()
  .then(() => {
    // Kullanıcıları oluşturma
    return User.bulkCreate([
      { firstName: 'John', lastName: 'Doe', age: 30 },
      { firstName: 'Jane', lastName: 'Doe', age: 25 },
      { firstName: 'Bob', lastName: 'Smith', age: 35 },
      // ... daha fazla kullanıcı
    ]);
  })
  .then(() => {
    // Basit bir where sorgusu kullanma
    return User.findAll({
      where: {
        firstName: 'John',
      },
    });
  })
  .then((users) => {
    console.log(users);
  })
  .catch((error) => {
```

Sequelize'de 'where' ifadesini kullanarak sorgularda kayıtları filtrelemek mümkündür. Örneğin, belirli bir isme sahip kullanıcıları bulmak için User.findAll({ where: { name: 'John' } }) kullanabilirsiniz.

9) JSON Web Token'lar (JWT) nedir ve web uygulamalarındaki rolü nedir?

JWT, bilgileri JSON nesnesi olarak taraflar arasında temsil etmenin kompakt ve kendi başına geçerli bir yoludur. Web uygulamalarında JWT'ler genellikle kimlik doğrulama ve yetkilendirme için kullanılır. Sunucu tarafından verilir ve kullanıcının kimliğini doğrulamak, erişim kontrolü ve daha fazlasını doğrulamak için kullanılabilir.

```
const jwt = require('jsonwebtoken');

// Örnek kullanıcı bilgileri
const user = {
  id: 123,
  username: 'example_user',
  role: 'admin',
};

// JWT oluşturma
const secretKey = 'your-secret-key';
const token = jwt.sign(user, secretKey, { expiresIn: '1h' });
console.log('Oluşturulan JWT:', token);

// JWT doğrulama
try {
  const decodedUser = jwt.verify(token, secretKey);
  console.log('Doğrulanan Kullanıcı:', decodedUser);
} catch (error) {
  console.error('JWT Doğrulama Hatası:', error.message);
}
```

JWT, bilgileri JSON nesnesi olarak taraflar arasında temsil etmenin kompakt ve kendi başına geçerli bir yoludur. Web uygulamalarında JWT'ler genellikle kimlik doğrulama ve yetkilendirme için kullanılır. Sunucu tarafından verilir ve kullanıcının kimliğini doğrulamak, erişim kontrolü ve daha fazlasını doğrulamak için kullanılabilir.

10) JWT'yi kimlik doğrulama için kullanırken dikkate almanız gereken bazı yaygın güvenlik hususlar nelerdir?

```
// Türkçe kod örnekleriyle devam edelim:

// JWT oluşturma örneği
const jwt = require('jsonwebtoken');
const secretKey = 'your-secret-key';

// Kullanıcı bilgileri
const user = {
  id: 123,
  username: 'example_user',
};

// JWT oluşturma
const token = jwt.sign(user, secretKey, { expiresIn: '1h' });
console.log('Oluşturulan JWT:', token);

// JWT doğrulama örneği
try {
  const decodedUser = jwt.verify(token, secretKey);
  console.log('Doğrulanan Kullanıcı:', decodedUser);
} catch (error) {
  console.error('JWT Doğrulama Hatası:', error.message);
}
```

JWT'yi kimlik doğrulama için kullanırken dikkate almanız gereken bazı yaygın güvenlik hususları, belirtilenin güvenli bir şekilde saklandığından emin olmak, HTTPS kullanmak, makul token sona erme süreleri belirlemek ve token'ın imzasını doğrulayarak manipülasyonu önlemektir. Ayrıca, duyarlı bilgileri token payload'ından uzak tutmak önemlidir.

Backend Teamwork 7

1) Express.js rotaları hangi HTTP yöntemlerini işleyebilir?

a) GET
b) POST
c) PUT
d) Hepsi

Cevap: d) Hepsi

Açıklama: Express.js, GET, POST ve PUT dahil olmak üzere çeşitli HTTP yöntemlerini işleyebilir.

2) Express.js'te GET isteğini işlemek için temel bir rota nasıl oluşturulur?

a) app.get('/', callbackFonksiyon)
b) app.use('/', callbackFonksiyon)
c) app.route('/', callbackFonksiyon)
d) app.post('/', callbackFonksiyon)

Cevap: a) app.get('/', callbackFonksiyon

Açıklama: app.get() yöntemi, GET isteklerini işlemek için bir rota tanımlamak için kullanılır.

3) Express.js handle error middleware'ini nasıl işler?

a) next() fonksiyonunu çağırarak
b) Otomatik olarak herhangi bir yapılandırma olmadan
c) Ayrı bir handle error rotası kullanarak
d) Hataları işlemez.

Cevap: c) Ayrı bir handle error rotası kullanarak

Açıklama: Express.js, ayrı handle error middleware'i tanımlayarak hataları işler.

4) Express.js'de request body JSON verilerini nasıl çözebilirsiniz?

a) req.body kullanarak
b) req.json kullanarak
c) req.parseJSON() kullanarak
d) req.data kullanarak

Cevap: a) req.body kullanarak

Açıklama: JSON verilerine req.body kullanarak erişilebilir.

5) Express.js'de app.set('view engine', 'ejs') yapılandırması ne amaçla kullanılır?

a) Sunucuyu başlatmak için
b) Bir görünüm motoru kurmak için
c) Rotaları yapılandırmak için
d) Rota yollarını tanımlamak için

Cevap: b) Bir görünüm motoru kurmak için

Açıklama: app.set('view engine', 'ejs') yapılandırması, Express.js'de EJS görünüm motorunu kurar.

6) Express.js'in res.redirect() yöntemi ne için kullanılır?

a) HTML şablonları renderlamak için
b) JSON yanıtları göndermek için
c) Başka bir URL'ye yönlendirmek için
d) Hataları işlemek için

Cevap: c) Başka bir URL'ye yönlendirmek için

Açıklama: res.redirect() yöntemi, başka bir URL'ye yönlendirmek için kullanılır.

7) Express.js'in app.locals nesnesinin amacı nedir?

a) Oturum verilerini depolamak için
b) Şablonlar için global değişkenleri tanımlamak için
c) Rota middleware'ini yönetmek için
d) Sunucunun güvenlik ayarlarını yapılandırmak için

Cevap: b) Şablonlar için genel değişkenleri tanımlamak için

Açıklama: Express.js'deki app.locals, tüm şablonlarda kullanılabilen genel değişkenleri tanımlamak için kullanılır.

8) Express.js ile güvenli kimlik doğrulama ve yetkilendirme için genellikle hangi araç kullanılır?

a) Passport.js
b) Express-Authenticator
c) SecureAuth.js
d) AuthGuard

Cevap: a) Passport.js

Açıklama: Express.js ile güvenli kimlik doğrulama ve yetkilendirme için genellikle Passport.js kullanılır.

9) Express.js'in static middleware'i ne için kullanılır?

a) JSON verilerini ayrıştırmak için
b) Hataları işlemek için
c) CSS ve JavaScript gibi statik dosyaları sunmak için
d) Kullanıcıları kimlik doğrulamak için

Cevap: c) CSS ve JavaScript gibi statik dosyaları sunmak için

Açıklama: Express.js'deki static middleware, stil sayfaları ve istemci tarafındaki JavaScript dosyaları gibi statik dosyaları sunmak için kullanılır.

10) Web uygulamasında sayfalama uygulamanın temel amacı nedir?

a) Arama motoru optimizasyonunu (SEO) iyileştirmek için
b) Kullanıcı arayüzü tasarımını geliştirmek için
c) İçeriği yönetilebilir sayfalara düzenlemek için
d) Güvenli kimlik doğrulamayı uygulamak için

Cevap: c) İçeriği yönetilebilir sayfalara düzenlemek için

Açıklama: Sayfalama, içeriği kullanıcılar için daha yönetilebilir hale getirmek amacıyla sayfalara böler.

11) Sequelize'de, varsa yeni bir veritabanı kaydı oluşturmak veya varsa güncellemek için hangi metod kullanılır?

a) findOrCreate()
b) updateOrCreate()
c) createOrUpdate()
d) insertOrUpdate()

Cevap: a) findOrCreate()

Açıklama: findOrCreate() yöntemi, yeni bir kayıt oluşturmak veya varsa güncellemek için kullanılır.

12) Skip tabanlı sayfalama nasıl çalışır?

a) Daha iyi performans için tüm verileri bir kerede yükler.
b) Belirli bir sayfa verisini belirli bir sayıda kaydı atlayarak alır.
c) Sabit boyutlu bloklara veriyi böler.
d) Veriyi rastgele herhangi bir sırayla alır.

Cevap: b) Belirli bir sayfa verisini belirli bir sayıda kaydı atlayarak alır.

Açıklama: Skip tabanlı sayfalama, belirli bir sayfa verisini almak için belirli bir sayıda kaydı atlar.

13) JWT Sorusu: JSON Web Token (JWT) içinde "payload"ın amacı nedir?

a) Token hakkında meta veri saklamak için
b) Token'ın geçerlilik süresini tanımlamak için
c) Kullanıcı veya talepler hakkında bilgi taşımak için
d) Token'ı şifrelemek için

Cevap: c) Kullanıcı veya iddialar hakkında bilgi taşımak için

Açıklama: JWT'nin "payload" kısmı, kullanıcı veya iddialar gibi bilgileri taşımak için kullanılır.

14) Bir Express.js uygulamasında etkili ve ölçeklenebilir bir kimlik doğrulama ve yetkilendirme mekanizması kurmak istiyorsunuz. Bu hedeflere ulaşmak için güvenlik, performans ve esneklik gibi faktörleri göz önünde bulundurarak hangi süreci ve bileşenleri kullanırdınız?

a) Hem kimlik doğrulamayı hem de yetkilendirmeyi aynı anda ele alan tek bir karmaşık middleware uygular.
b) Express.js oturumlarını kullanır ve oturum verilerini bir sunucu tarafı veritabanında saklar, her istek için kullanıcı kimlik bilgilerini ve izinleri kontrol eder.
c) JWT tabanlı kimlik doğrulamayı uygular, kullanıcı rollerini ve izinlerini tokenlarda saklar ve her korunan rota için bunları doğrular.
d) Verimlilik için Basic Authentication kullanır ve kullanıcı kimlik bilgilerini doğrudan uygulama kodunda saklar.

c) JWT tabanlı kimlik doğrulamayı uygular, kullanıcı rollerini ve izinlerini tokenlarda saklar ve her korunan rota için bunları doğrular.

Açıklama: JWT tabanlı kimlik doğrulama, roller ve izinlerin tokenlarda saklanması, her bir korunan rota için bu bilgilerin doğrulanması gibi avantajları nedeniyle genellikle tercih edilen bir yöntemdir.

15) Yüksek trafikli bir Express.js uygulaması oluşturuyorsunuz ve performansını optimize etmek istiyorsunuz. İstek işleme hızını artırmak ve yanıt sürelerini minimize etmek için hangi middleware'yi kullanmayı düşünmelisiniz?

a) express-compression
b) express-sessions
c) express-validator
d) express-static-gzip

Cevap: d) express-static-gzip

Açıklama: express-static-gzip middleware'i, statik dosyaları sıkıştırarak sunar ve bu da istek işleme hızını önemli ölçüde artırabilir ve yanıt sürelerini azaltabilir, bu nedenle yüksek trafikli uygulamalar için optimal bir tercihtir.