🏛️ Mimari Özeti: Öğrenci Kulübü Projesi

**1.** 🚀 Seçilen Teknoloji Yığını (Technology Stack)

Projemiz, modern ve esnek bir Node.js ekosistemi üzerine kurulmuştur. Kullanılan ana teknolojiler `package.json` dosyasında detaylandırılmıştır.

🖥️ Backend & Sunucu

**Node.js:** Ana çalışma zamanı (runtime) ortamı.

**Express.js:** Sunucu oluşturma, yönlendirme (routing) ve API katmanı yönetimi için ana framework.

**EJS (Embedded JavaScript):** Sunucu tarafında dinamik HTML sayfaları oluşturmak için kullanılan şablon motoru.

💾 Veritabanı & Veri Yönetimi

**MySQL:** Ana ilişkisel veritabanı sunucusu.

**Sequelize:** Veritabanı işlemleri için kullanılan ORM (Object-Relational Mapper). Karmaşık SQL sorguları yerine JavaScript objeleriyle çalışmamızı sağlar.

**mysql2:** Sequelize'nin MySQL veritabanı ile konuşabilmesi için gereken sürücü.

🔐 Oturum Yönetimi (Session)

**express-session:** Kullanıcıların giriş durumunu yönetmek için standart oturum middleware'i.

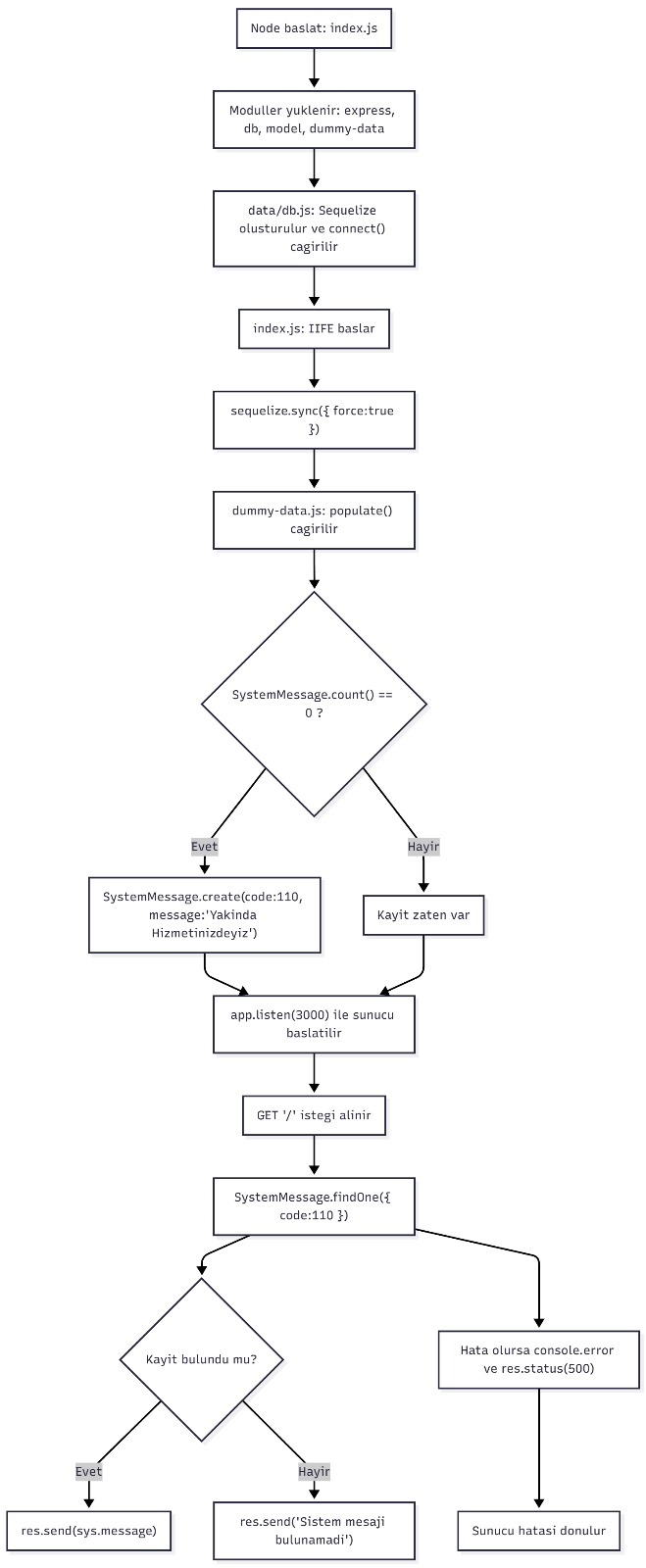
**connect-session-sequelize:** Oturum bilgilerini sunucu hafızası yerine kalıcı olarak MySQL veritabanında saklamak için kullanılır.

🛠️ Geliştirme Araçları

**Nodemon:** Geliştirme sırasında dosya değişikliklerini algılayıp sunucuyu otomatik olarak yeniden başlatan araç.

**2**. 🌊 Veri ve Uygulama Akış Diyagramı

Uygulamanın başlangıç (`init`) ve bir `GET /` isteğini yanıtlama süreci aşağıdaki diyagramda görselleştirilmiştir.



Akışın Adım Adım Açıklaması:

1- Node çalıştırılarak uygulama başlatılır (npm start).

2- index.js dosyası gerekli modülleri yükler: Express, Sequelize bağlantısı (data/db.js), model (models/systemMessage.js) ve başlangıç verisi fonksiyonu (data/dummy-data.js).

3- data/db.js içinde tanımlı connect() fonksiyonu çalışır ve sequelize.authenticate() ile MySQL’e bağlantı denenir; bağlantı başarılıysa konsola mesaj yazılır, başarısızsa hata loglanır.

4- index.js içindeki IIFE (anında çalıştırılan async fonksiyon) başlar.

5- IIFE içinde sequelize.sync({ force: true }) çağrılır; bu çağrı mevcut tabloları siler ve model tanımına göre yenilerini oluşturur. (Not: force: true veri temizler.)

6- data/dummy-data.js içindeki populate() (ya da dummyData()) çağrılır; SystemMessage.count() ile tablo boş mu diye kontrol edilir.

7- Eğer SystemMessage tablosunda kayıt yoksa (count == 0), SystemMessage.create({ code:110, message:"Yakında Hizmetinizdeyiz" }) ile başlangıç kaydı eklenir.

8- Başlangıç verisi eklenmesi veya atlanması sonrası Express sunucusu app.listen(3000) ile dinlemeye alınır.

9- Kullanıcı tarayıcıdan veya istemciden GET "/" isteği gönderir.

10- index.js içindeki route handler çalışır; SystemMessage.findOne({ where: { code: 110 } }) sorgusu ile veritabanında code:110 olan kayıt aranır.

11- Eğer sorgu başarıyla bir kayıt dönerse, bulunduğu takdirde res.send(sys.message || "") ile message içeriği istemciye gönderilir.

12- Eğer kayıt bulunamazsa, res.send("Sistem mesajı bulunamadı.") ile uygun bilgi istemciye döndürülür.

13- Eğer sorgu sırasında veya herhangi bir DB işleminde hata oluşursa, hata console.error("DB hata:", err) ile loglanır ve res.status(500).send("Sunucu hatası.") ile 500 hata kodu döndürülür.