**3. Güncellenmiş Anlatım Metni:**

"Sunumuma, projemin çıkış noktası olan temel problemle başlamak istiyorum. Java kodlarını güçlendirmek için ProGuard gibi endüstri standardı, çok güçlü araçlar mevcut. Ancak bu güç, beraberinde ciddi bir karmaşıklık getiriyor ve kendi içinde bir uzmanlık alanı gerektiriyor."

*(Sol taraftaki ProGuard kuralları resmini işaret ederek)* "Bir yandan, doğru konfigürasyonu yapmak için bu şekilde yüzlerce kural ve parametreyi bilmeniz, -keep direktiflerini, istisnaları hatasız yazmanız gerekiyor. Buradaki tek bir virgül hatası veya unutulan bir kural, uygulamanızın çalışma zamanında beklenmedik bir şekilde çökmesine neden olabilir."

*(Sağ taraftaki ProGuard GUI resmini işaret ederek)* "Diğer yandan, bu araçların sunduğu grafik arayüzler bile, onlarca sekme, anlaşılması zor seçenekler ve teknik jargonla dolu. Son kullanıcı veya konuya hakim olmayan bir geliştirici için bu arayüz, bir çözümden çok, bir labirentten farksız."

"İşte bizim çözmek istediğimiz problem tam olarak buydu: Bu **güç**, ortalama bir geliştirici için **erişilebilir değildi.**"

"Ve bu durum, sadece bir kullanım zorluğu değil, aynı zamanda doğrudan **iş riski** anlamına geliyordu:"

*(Bundan sonra, daha önce planladığımız gibi 3 ana riski vurgulayarak devam edebilirsin)*

"**1. Güvenlik Riski:** Yanlış konfigürasyon, sahte bir güvenlik hissiyatı yaratır." "**2. Verimlilik Riski:** Geliştiricinin değerli zamanı, bu karmaşık araçları öğrenmekle geçer." "**3. Tutarlılık Riski:** Her geliştiricinin farklı ayarlar kullanması, öngörülemez sonuçlar doğurur."

"**Staj Görevim:** Bu uzmanlık gerektiren gücü, herkesin güvenle ve kolayca kullanabileceği, tutarlı sonuçlar üreten bir platforma dönüştürmekti."

**4. Güncellenmiş Anlatım Metni:**

"Slayt 3'te bahsettiğim problemi çözmek için işe, projenin ana fikrini test edecek en basit versiyonu, yani bir MVP geliştirerek başladım."

*(Soldaki basit mimari şemasını işaret ederek)* "Mimari olarak sistem, gördüğünüz gibi son derece basitti. Sadece bir arayüz, gelen isteği doğrudan komut satırına yönlendiren bir backend ve bu komutları çalıştıran bir mantıktan oluşuyordu."

*(Sağdaki ilkel arayüz görüntüsünü işaret ederek)* "**Ve bu mimarinin kullanıcıya yansıması da tam olarak buydu.** Gördüğünüz bu ilkel arayüz, benim ilk prototipimdi. Sadece bir dosya seçme butonu ve 'Çalıştır' düğmesinden ibaretti. Ne bir ayar, ne bir kullanıcı girişi, ne de bir raporlama vardı."

"Bu MVP'nin tek ve net bir amacı vardı: **Temel konseptin çalışabilirliğini kanıtlamak.** Yani, bu basit arayüz üzerinden karmaşık bir komut satırı aracını güvenli bir şekilde tetikleyip, sonucunu alabiliyor muyuz? Bu sorunun cevabını 'evet' olarak aldığımda, projenin bir sonraki evrim aşamasına geçmeye hazırdım."

**5. Anlatacağın Metin (Sunum Notların):**

"Minimum Uygulanabilir Ürün, temel konseptin çalıştığını kanıtladı. Bu başarı, stajımın en heyecanlı kısmının kapısını araladı. Artık 'bu çalışıyor mu?' sorusunu değil, **'bu ne kadar daha iyi ve kapsamlı olabilir?'** sorusunu sormaya başlamıştım."

*(Ortadaki MVP şemasını işaret ederek)* "Elimdeki bu basit çekirdek, aslında çok daha fazlası için bir temel olabilirdi."

*(Etraftaki baloncukları genel olarak göstererek)* "Stajımın geri kalanında, bu temelin üzerine yeni yetenekler ekleyerek onu tek amaçlı bir araçtan, tam teşekküllü bir platforma dönüştürmeye başladım."

"Kullanıcılara sadece bir 'çalıştır' butonu değil, tüm süreci yönetebilecekleri **uzman ayarları** vermeyi hedefledim. Yaptıkları işlemin başarısını görebilmeleri için **somut metrikler ve raporlama** ekledim. Platformu bir ekip tarafından kullanılabilir hale getirmek için **kullanıcı yönetimi** ve **admin paneli** gibi kurumsal yetenekler kazandırdım. Ve son olarak, projenin vizyonunu sadece kod *gizlemekten* öteye taşıyarak, **JavaScript desteği** ve **SonarQube ile kod kalite analizi** gibi özelliklerle kapsamını genişlettim."

"Şimdi size, bu özelliklerin her birini platforma nasıl entegre ettiğimi ve projenin nasıl adım adım bugünkü haline evrildiğini göstereceğim."

**6. Anlatacağın Metin (Sunum Notların):**

"MVP'nin temel konsepti kanıtladıktan sonra, platformu gerçek dünya ihtiyaçlarına cevap verecek şekilde geliştirmeye başladım. İlk adımda üç ana alana odaklandım:"

*(Soldaki "Uzman Kontrolü" görüntüsünü işaret ederek)* "**Birincisi, KONTROL.** 'Tek tuşla hallet' yaklaşımı herkese hitap etmiyordu. Uzman kullanıcıların, süreci en ince ayrıntısına kadar yönetmek isteyeceğini biliyordum. Bu yüzden, Control Flow Flattening gibi gelişmiş koruma tekniklerinden, Shrinking gibi optimizasyon ayarlarına kadar, kullandığımız araçların tüm profesyonel yeteneklerini arayüze taşıdım."

*(Ortadaki "Somut Metrikler" görüntüsünü işaret ederek)* "**İkincisi, ÖLÇÜLEBİLİRLİK.** Yaptığımız işlemin ne kadar etkili olduğunu somut verilerle göstermek kritikti. Bu nedenle, işlem sonunda dosya boyutu ve sembol sayısındaki değişim gibi metrikleri karşılaştıran ve sonucu indirilebilir kılan bu raporlama ekranını geliştirdim. Artık kullanıcılar, yaptıkları korumanın etkisini net rakamlarla görebiliyorlardı."

*(Sağdaki "Kurumsal Yetenekler" görüntüsünü işaret ederek)* "**Üçüncüsü, GÜVENLİK ve YÖNETİM.** Platformu kişisel bir araç olmaktan çıkarıp, bir ekip tarafından güvenle kullanılabilir hale getirmek için JWT tabanlı bir kullanıcı kayıt ve giriş sistemi ekledim. Buna ek olarak, yöneticilerin tüm sistemi denetleyebilmesi için bir admin paneli tasarladım."

"Bu üç adımla, iHTAR artık sadece bir 'çalıştırıcı' değil; kullanıcısına tam kontrol veren, başarısını ölçen ve kurumsal kullanıma hazır bir platform haline gelmişti."

 **Sol Bölüm (Kontrol):**

* Arayüzünüzdeki **detaylı parametre seçeneklerinin** (C++ için Control Flow Flattening, Java için Shrinking, JS için String Array vb.) en karmaşık ve etkileyici görünen kısmından bir ekran görüntüsü koy.
* **Başlık:** Uzman Kontrolü

 **Orta Bölüm (Raporlama):**

* İşlem bittikten sonra çıkan **sonuç raporu modal'ının** (ReportModal.tsx'in çıktısı) ekran görüntüsünü koy. Özellikle "önce/sonra" dosya boyutu veya sembol sayısı gibi metriklerin göründüğü bir anı yakala.
* **Başlık:** Somut Metrikler

 **Sağ Bölüm (Kurumsal Yetenekler):**

* **Giriş (Login) ekranınızın** ve **Admin Panelinizin** ekran görüntülerini birleştirerek koy.
* **Başlık:** Kurumsal Yetenekler

**7. Görsel Tasarım Konsepti:** Bu slaytı, projenin iki farklı yönde nasıl büyüdüğünü göstermek için ikiye böl.

* **Sol Taraf (Yeni Teknoloji Desteği):**
  + **JavaScript obfuscator** seçeneklerinin olduğu arayüz ekran görüntüsünü koy.
  + **Başlık:** Yatay Genişleme: JavaScript Desteği
* **Sağ Taraf (Yeni Yetenek Alanı):**
  + **SonarQube modülünün** arayüzdeki "SonarQube Arayüzünü Aç" butonunu gösteren ekran görüntüsünü koy. Arka plana veya yanına SonarQube'un kendi logosunu veya rapor ekranından bir kesit ekleyebilirsin.
  + **Başlık:** Dikey Derinleşme: Kod Kalite Analizi

**7. Anlatacağın Metin (Sunum Notların):**

"Platform artık C++ ve Java için olgunlaşmıştı. Ancak stajımın son aşamasında, iHTAR'ın potansiyelini daha da ileriye taşıyacak iki önemli adım daha attım:"

*(Soldaki "JavaScript Desteği" görüntüsünü işaret ederek)* "**İlk olarak, platformun kapsamını YATAYDA genişlettim.** Günümüz dünyasında web teknolojilerinin önemini göz ardı edemezdik. Bu yüzden, sadece C++ ve Java'yı değil, modern web geliştirmeyi de kapsayacak şekilde, JavaScript kodlarını güçlendiren tamamen yeni bir worker-js servisi geliştirdim ve sisteme entegre ettim. Artık iHTAR, daha geniş bir geliştirici kitlesine hitap ediyordu."

*(Sağdaki "Kod Kalite Analizi" görüntüsünü işaret ederek)* "**İkinci ve belki de en önemli adım olarak, projenin vizyonunu DİKEYDE derinleştirdim.** Bu noktaya kadar projenin odağı, yazılmış olan kodu dış tehditlere karşı **'gizlemek'** ve **'korumaktı'.** SonarQube entegrasyonu ile bu vizyonu dönüştürdüm."

"Artık iHTAR, sadece kodu gizlemekle kalmıyor, aynı zamanda kodun içindeki hataları, güvenlik zafiyetlerini ve 'code smell' olarak tabir edilen kötü pratikleri bularak kodun kalitesini artırmaya da yardımcı oluyordu. Platform artık sadece bir **'güvenlik'** aracı değil, aynı zamanda bir **'kalite güvence'** merkezi haline gelmişti."

"Bu iki adımla, iHTAR basit bir araçtan, yazılım geliştirme yaşam döngüsünün (SDLC) birden çok aşamasına dokunan kapsamlı bir platforma evrildi."

**Slayt 8: Bu Evrimi Mümkün Kılan Güç: Mikroservis Mimarisi**

**1. Slayt Başlığı:** Bu Evrimi Mümkün Kılan Güç: Mikroservis Mimarisi

**2. Görsel Tasarım Konsepti:** Bu slaytın tek ve en güçlü görseli, projenin **tam ve nihai mimari şeması** olmalı.

* docker-compose.yml dosyanı temel alan, tüm servislerin (Nginx, Frontend, Backend API, PostgreSQL, Redis ve tüm worker'ların: worker-cpp, worker-java, worker-js, worker-sonar) ve aralarındaki ana bağlantıların net bir şekilde görüldüğü diyagramı tam ekran kullan.

**3. Anlatacağın Metin (Sunum Notların):**

*(Bir önceki slaytı bitirirkenki son cümlen: "...platformu kapsamlı bir platforma evrildi.")*

"Peki, herkesin aklındaki soruya gelelim: Tüm bu birbirinden farklı özellikleri ve teknolojileri - C++, Java, JavaScript, Python tabanlı bir analiz aracı - projeye bu kadar kolay, mevcut sistemi hiç bozmadan nasıl dahil edebildim?"

*(Nihai mimari şemasını göstererek)*

"Cevap, projenin en başında attığım en sağlam temelde, bilinçli olarak seçtiğim bu **Mikroservis Mimarisi**'nde yatıyor."

"Bu mimari bana sadece teknik bir esneklik değil, aynı zamanda **İş Çevikliği (Business Agility)** de sağladı. Ne demek istiyorum? Şöyle anlatayım:"

"Yarın bir gün yönetimden veya bir müşteriden 'Artık Go dilinde yazılmış projeleri de güçlendirmemiz gerekiyor' diye bir talep geldiğini düşünelim. Eğer monolitik, yani tek parça bir yapıda olsaydım, bu yeni teknolojiyi mevcut sisteme eklemek aylar sürebilecek, riskli ve karmaşık bir yeniden yapılanma projesi anlamına gelirdi."

"Fakat bu mimari sayesinde, yapmam gereken tek şey, sisteme yeni bir worker-go servisi eklemek. Bu yeni servis, kendi Docker konteynerinde, kendi dünyasında çalışır, diğer servisleri hiç etkilemez ve sadece Redis üzerinden haberleşir. Bu sayede, böyle stratejik bir talebi haftalar gibi kısa bir sürede, güvenli bir şekilde karşılayabilirim."

"İşte projenin başından beri anlattığım bu evrim hikayesini mümkün kılan güç, bu esnek ve genişleyebilir temeldi."

"Şimdi, bu nihai mimariyi oluşturan katmanları ve her birinin görevini kod örnekleriyle birlikte daha yakından inceleyelim."

**3. Anlatacağın Metin ve Kodlar:**

"Nihai mimarimizin kalbinde, her biri tek bir görevde uzmanlaşmış, asıl ağır işi yapan 'işçi' servisleri yer alır. Şimdi bu işçilerin, kendi görevlerini nasıl yerine getirdiklerini kod seviyesinde inceleyelim."

**Sol Sütun: C++ Worker (main.cpp)**

* **Alt Başlık:** Harici Süreç Yönetimi: popen
* **Gösterilecek Kod Kesiti:**

C++

// Harici komutları çalıştırıp çıktısını yakalayan yardımcı fonksiyon

std::string exec(const char\* cmd) {

std::array<char, 128> buffer;

std::string result;

// popen ile komutu "read" modunda aç

std::unique\_ptr<FILE, decltype(&pclose)> pipe(popen(cmd, "r"), pclose);

if (!pipe) {

throw std::runtime\_error("popen() failed!");

}

// Komutun çıktısını satır satır oku

while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {

result += buffer.data();

}

return result;

}

// Ana döngüdeki kullanımı

std::string symbol\_count\_before = exec("nm -gC --defined-only original\_output | wc -l");

* **Anlatım Metni:** "C++ worker'ımızın kalbinde, harici shell komutlarını (clang++, nm gibi) çalıştırıp çıktısını bir string olarak yakalayan bu exec fonksiyonu yer alır. Burada standart C++ kütüphanesinden popen fonksiyonunu kullanıyoruz. Bu fonksiyon, belirtilen komutu bir alt süreç (sub-process) olarak başlatır ve bu sürecin standart çıktısını (stdout) okuyabileceğimiz bir boru hattı (pipe) açar. Örneğin, derleme sonrası nm komutunun çıktısını bu fonksiyonla alıp satır sayısını sayarak, obfuscation'ın sembol sayısını ne kadar azalttığını metrik olarak hesaplıyoruz."

**Orta Sütun: Java Worker (Worker.java)**

* **Alt Başlık:** Güvenli Süreç Yönetimi: ProcessBuilder
* **Gösterilecek Kod Kesiti:**

Java

// ProGuard'ı harici bir süreç olarak çalıştırma

ProcessBuilder pb = new ProcessBuilder(

"java",

"-jar",

"proguard.jar",

"@" + tempConfigFile.getAbsolutePath() // Dinamik oluşturulan config dosyası

);

pb.redirectErrorStream(true);

Process process = pb.start();

// Sürecin çıktısını anlık olarak yakalayıp Redis'e log olarak yayınla

new StreamGobbler(process.getInputStream(), line -> {

jedis.publish("job-logs", line);

}).run();

int exitCode = process.waitFor(); // Sürecin bitmesini bekle

* **Anlatım Metni:** "Java worker'ımızda ise harici süreçleri yönetmek için endüstri standardı ve en güvenli yöntem olan ProcessBuilder sınıfını kullanıyoruz. Bu sınıf, komutu ve argümanlarını ayrı ayrı alarak Command Injection gibi zafiyetleri engeller. Burada kritik olan nokta, kullanıcının arayüzden seçtiği ayarlara göre dinamik olarak bir ProGuard konfigürasyon dosyası (.pro) oluşturup, @ işaretiyle bu dosyayı ProGuard'a argüman olarak vermemizdir. Ayrıca StreamGobbler adını verdiğimiz yardımcı bir sınıf ile bu sürecin ürettiği çıktıyı canlı olarak okuyup, anında Redis üzerinden kullanıcıya log olarak yayınlıyoruz."

**Sağ Sütun: Sonar Worker (worker.py)**

* **Alt Başlık:** Basit ve Etkili Süreç Yönetimi: subprocess
* **Gösterilecek Kod Kesiti:**

Python

# sonar-scanner komutunu ve argümanlarını bir liste olarak oluştur

command = [

'sonar-scanner',

f'-Dsonar.projectKey={project\_key}',

f'-Dsonar.sources=.',

f'-Dsonar.host.url=http://sonarqube:9000',

f'-Dsonar.login={SONAR\_TOKEN}'

]

# subprocess.run ile komutu çalıştır ve çıktıyı yakala

result = subprocess.run(

command,

capture\_output=True,

text=True,

cwd=project\_path # Komutu projenin olduğu dizinde çalıştır

)

if result.returncode == 0:

# Başarılı ise SonarQube linkini oluştur ve yayınla

# ...

* **Anlatım Metni:** "Python worker'ımız ise Python'un gücünü, yani basitliğini ve etkinliğini kullanıyor. subprocess modülü ile sonar-scanner aracını gerekli tüm parametrelerle birlikte çalıştırıyoruz. -D ile başlayan parametreler, SonarQube'a analiz edilecek projenin anahtarını, kaynak kodun nerede olduğunu ve sunucu adresini bildirir. cwd parametresi ile komutu doğrudan analiz edilecek projenin klasörü içinde çalıştırarak path sorunlarını ortadan kaldırıyoruz. İşlem başarılı olursa, analizin sonuçlarının bulunduğu SonarQube linkini oluşturup Redis üzerinden Backend'e bildiriyoruz."

**Slayt 10: Platform Güvenliği ve Dayanıklılık**

**1. Slayt Başlığı:** Platform Güvenliği ve Dayanıklılık

**2. Görsel Tasarım Konsepti:** Bu slaytı üç ana bölüme ayır. Her bölümün bir başlığı, basit bir ikonu ve altında tek cümlelik açıklaması olsun.

* **Sol Bölüm:**
  + **İkon:** Üzerinde kilit olan bir dosya/klasör ikonu.
  + **Başlık:** Güvenli Dosya Yönetimi
* **Orta Bölüm:**
  + **İkon:** Docker'ın balina ikonu veya bir konteyner ikonu.
  + **Başlık:** Konteyner İzolasyonu
* **Sağ Bölüm:**
  + **İkon:** Veritabanı (silindir) ikonu.
  + **Başlık:** Veritabanı Güvenliği

**3. Anlatacağın Metin (Sunum Notların):**

"Bir güvenlik platformu geliştirirken, platformun kendisinin de güvenli ve dayanıklı olması gerekir. Bu konuda, projenin temelinden itibaren uyguladığım bazı önemli prensipler var:"

**1. Güvenli Dosya Yönetimi**

* **Problem:** "Kötü niyetli bir kullanıcı, dosya adı olarak ../../etc/passwd veya my\_code.zip; rm -rf / gibi bir ifade göndererek sistemde **Path Traversal** veya **Command Injection** zafiyetleri yaratmaya çalışabilir."
* **Çözüm ve Kod Anlatımı:** "Bunu engellemek için basit ama çok etkili bir kural uyguladım: **Kullanıcıdan gelen hiçbir girdiye güvenme.** Backend'de multer ile dosya alırken, kullanıcının gönderdiği orijinal dosya adını asla doğrudan kullanmıyoruz. Bunun yerine, uuid kütüphanesi ile her dosya için tamamen rastgele ve eşsiz bir dosya adı üretiyoruz."
  + **Gösterilecek Kod Kesiti (backend-api/index.js içindeki mantık):**

JavaScript

// Multer yapılandırması (örnek)

const storage = multer.diskStorage({

destination: './uploads',

filename: (req, file, cb) => {

// Orjinal dosya adı yerine rastgele bir uuid + orjinal uzantı kullan

const uniqueSuffix = uuidv4();

const extension = path.extname(file.originalname);

cb(null, uniqueSuffix + extension);

}

});

* + **Anlatım:** "Bu sayede, c1f2b1a3-....zip gibi zararsız bir dosya adı elde ediyoruz ve bu zafiyetleri temelden engellemiş oluyoruz."

**2. Konteyner İzolasyonu ve Güvenliği**

* **Problem:** "Docker konteynerleri varsayılan olarak root kullanıcısıyla çalışır. Eğer bir worker konteyneri ele geçirilirse, saldırgan root yetkileriyle tüm konteynere ve hatta potansiyel olarak ana makineye (host) zarar verebilir."
* **Çözüm ve Kod Anlatımı:** "Bunu önlemek için, tüm Dockerfile'larımda **'En Az Yetki Prensibi'ni (Principle of Least Privilege)** uyguladım. Konteyner içinde sınırlı yetkilere sahip özel bir appuser kullanıcısı oluşturup, uygulamayı bu kullanıcı ile çalıştırıyorum."
  + **Gösterilecek Kod Kesiti (Herhangi bir Dockerfile'dan):**

Dockerfile

# ...

# Yetkisiz bir kullanıcı oluştur

RUN addgroup -S appgroup && adduser -S appuser -G appgroup

# Dizinin sahipliğini bu kullanıcıya ver

RUN chown -R appuser:appgroup /usr/src/app

# Root'tan bu kullanıcıya geç

USER appuser

# Uygulamayı bu yetkisiz kullanıcı ile çalıştır

CMD ["node", "index.js"]

* + **Anlatım:** "Bu basit USER appuser komutu, bir zafiyet anında oluşabilecek hasarı büyük ölçüde sınırlar."

**3. Veritabanı Güvenliği**

* **Problem:** "Web uygulamalarının en bilinen zafiyetlerinden biri: **SQL Injection.**"
* **Çözüm ve Kod Anlatımı:** "Bunu engellemek için, veritabanına giden hiçbir sorguda kullanıcıdan gelen veriyi doğrudan string birleştirme ile kullanmadım. Bunun yerine, kullandığım node-postgres (pg) kütüphanesinin standart bir özelliği olan **'Parametreli Sorgular'ı (Parameterized Queries)** kullandım."
  + **Gösterilecek Kod Kesiti (backend-api/routes/users.js veya index.js'ten):**

JavaScript

// KÖTÜ ÖRNEK (Bunu yapmadık):

// const query = "SELECT \* FROM users WHERE email = '" + req.body.email + "'";

// İYİ ÖRNEK (Yaptığımız):

const { rows } = await db.query(

'SELECT \* FROM users WHERE email = $1',

[req.body.email]

);

* + **Anlatım:** "Burada, kullanıcıdan gelen email adresi sorguya doğrudan eklenmiyor. $1 ile bir yer tutucu olarak işaretleniyor ve kütüphaneye ayrı bir parametre olarak veriliyor. Kütüphane, bu veriyi güvenli bir şekilde işleyerek SQL enjeksiyonunu imkansız hale getiriyor."

Bu slaytla, projenin sadece çalışan değil, aynı zamanda güvenli ve sağlam temeller üzerine kurulu bir sistem olduğunu kanıtlamış oldun.

**Slayt 11: Nihai Sistemde Bir İşlemin Anatomisi**

**1. Slayt Başlığı:** Bir İşlemin Uçtan Uca Yolculuğu

**2. Görsel Tasarım Konsepti:** Bu slaytta, daha önce de kullandığın **nihai ve tam mimari şemanı** kullanacaksın. Ancak bu sefer statik bir resim olarak değil, **animasyonlu** olarak kullanman çok etkili olur. PowerPoint veya Google Slides'ta her adıma bir animasyon ekleyerek, sanki veri sistemin içinde akıyormuş gibi bir izlenim yaratabilirsin. Her adımda ilgili okun veya kutunun parlamasını sağlayabilirsin.

**3. Anlatacağın Metin ve Adımlar:**

"Tüm bu mimariyi, güvenlik önlemlerini ve özellikleri bir araya getirdiğimizde, bir kullanıcının başlattığı basit bir işlem arka planda nasıl bir yolculuğa çıkıyor? Gelin, bir C++ projesinin güçlendirilme sürecini uçtan uca takip edelim:"

**(Adım 1: İstek ve Yönlendirme)**

* **Animasyon:** Kullanıcı -> Nginx -> Frontend -> Backend API yolu parlasın.
* **Anlatım:** "**Adım 1: İstek.** Her şey kullanıcının, arayüz üzerinden dosyasını ve istediği ayarları seçip butona basmasıyla başlar. Bu istek, Nginx tarafından karşılanır ve /api/jobs endpoint'ine, yani Backend servisimize yönlendirilir."

**(Adım 2: Kabul ve Kuyruğa Alma)**

* **Animasyon:** Backend API kutusu parlasın. Backend API'den PostgreSQL'e ve Redis'teki cpp-jobs kuyruğuna giden oklar belirsin.
* **Anlatım:** "**Adım 2: Kabul ve Kuyruğa Alma.** Backend, isteği alır almaz üç şey yapar: Önce PostgreSQL veritabanına bu iş için 'processing' durumunda yeni bir kayıt açar. Ardından, worker'ın ihtiyaç duyacağı tüm bilgileri içeren bir iş paketi hazırlar. Son olarak, bu paketi Redis'teki cpp-jobs kuyruğuna atar. İşte burada, daha önce Slayt 8'de kodunu gördüğümüz lpush komutu devreye giriyor. Bu adımdan hemen sonra kullanıcıya 'İşleminiz başladı' mesajı döner ve kullanıcı beklemez."

**(Adım 3: İşleme ve Anlık Raporlama)**

* **Animasyon:** Redis'teki cpp-jobs kuyruğundan worker-cpp'ye bir ok gelsin. worker-cpp kutusu çalışıyormuş gibi parlasın. worker-cpp'den Redis'teki job-logs kanalına kesikli oklar akmaya başlasın.
* **Anlatım:** "**Adım 3: İşleme ve Anlık Raporlama.** Bu sırada worker-cpp servisi, kuyruğu dinlediği için yeni işi anında fark eder ve alır. Kod derleme, güçlendirme gibi ağır işlemleri yapmaya başlar. İşlem sırasında ürettiği her önemli bilgiyi, örneğin 'Derleme başladı...', 'Semboller sayılıyor...' gibi logları, anında Redis'in job-logs kanalına yayınlar. Slayt 9'da gördüğümüz ProcessBuilder veya popen'dan gelen çıktılar tam olarak burada yayınlanır."

**(Adım 4: Sonuç Bildirimi)**

* **Animasyon:** worker-cpp'den Redis'teki job-results kanalına kalın bir ok gelsin.
* **Anlatım:** "**Adım 4: Sonuç Bildirimi.** Worker, tüm görevini başarıyla tamamladığında, analiz metriklerini ve sonuç dosyasının adını içeren nihai bir sonuç paketini Redis'in job-results kanalına yayınlar."

**(Adım 5: Güncelleme ve Anlık İletim)**

* **Animasyon:** Redis'teki job-logs ve job-results kanallarından Backend API'ye oklar gelsin. Backend API'den PostgreSQL'e bir ok gitsin. Son olarak, Backend API'den Kullanıcı'ya doğru bir WebSocket oku (genellikle şimşek ikonuyla gösterilir) belirsin.
* **Anlatım:** "**Adım 5: Güncelleme ve Anlık İletim.** Backend API, hem anlık log kanallarına hem de sonuç kanalına abone olduğu için bu mesajları anında yakalar. Gelen sonuçlarla PostgreSQL'deki iş kaydını 'completed' olarak günceller. En önemlisi, hem anlık logları hem de nihai sonucu, açık olan WebSocket bağlantısı üzerinden doğrudan kullanıcının tarayıcısına iletir. Bu sayede kullanıcı, tüm süreci canlı olarak izler ve işlem bittiğinde sonucunu anında görür."

Bu slayt, projenin tüm parçalarının nasıl senkronize bir şekilde, asenkron bir yapıda çalıştığını mükemmel bir şekilde özetler.

**Slayt 12: Teknik Kararlar ve Alternatifler ("Neden?" Slaytı)**

**1. Slayt Başlığı:** Teknik Kararlar ve Değerlendirmeler

**2. Görsel Tasarım Konsepti:** Bu slaytı soru-cevap formatında tasarla. Her bir "Neden?" sorusu için bir bölüm olsun. Karmaşık görsellere gerek yok, metnin kendisi ve netliği ön planda olmalı. Soruları vurgulu, cevapları ise daha sade bir formatta yazabilirsin.

**3. Anlatacağın Metin (Sunum Notların):**

"Tüm bu mimariyi tasarlarken, her adımda belirli kararlar aldım ve bu kararların getiri ve götürülerini değerlendirdim. Şimdi, aklınıza gelebilecek bazı 'Neden?' sorularını ve bu kararların arkasındaki mantığı açıklamak istiyorum."

**Soru 1: Neden Monolitik (Tek Parça) Bir Yapı Değil de Mikroservis?**

* **Anlatım:** "Bu projenin en temel mimari kararıydı. Monolitik bir yapı, başlangıçta daha hızlı geliştirilebilir gibi görünse de, projenin evrim hikayesinde gördüğümüz gibi bu esnekliği sağlaması imkansızdı."
* **Cevap:**
  + **"Teknoloji Çeşitliliği (Heterojen Yapı):** Projemiz doğası gereği farklı teknolojileri bir arada kullanmak zorunda. C++ derlemek için C++, Java'yı işlemek için Java, SonarQube'u çalıştırmak için Python gerekiyor. Mikroservis mimarisi, bu farklı teknolojilerin kendi izole ortamlarında, birbirini etkilemeden çalışmasına olanak tanıdı."
  + **"İş Çevikliği ve Genişleyebilirlik:** Sunumun başında anlattığım gibi, sisteme yeni bir worker-js veya worker-sonar eklemek, mevcut yapıyı hiç riske atmadan, sadece yeni bir servis eklemek kadar kolay oldu. Bu, monolitik bir yapıda aylar sürebilecek bir refactoring gerektirirdi."
  + **"Bağımsız Ölçeklenebilirlik:** Eğer gelecekte C++ işlemleri çok yoğunlaşırsa, sadece worker-cpp servisinin sayısını artırarak sistemi kolayca ölçekleyebiliriz. Monolitik bir yapıda ise tüm uygulamayı, kullanılmayan kısımlarıyla birlikte ölçeklemek zorunda kalırdık."

**Soru 2: Backend için Neden Node.js? CPU-yoğun işler için ideal değil gibi?**

* **Anlatım:** "Bu, Node.js ile ilgili sıkça karşılaşılan çok yerinde bir soru. Ancak cevabı, mimarimizin sorumlulukları nasıl ayırdığında gizli."
* **Cevap:**
  + "Backend API servisimiz, yani Node.js'in çalıştığı yer, aslında hiç CPU-yoğun bir iş **yapmıyor.** Onun görevi tamamen **I/O-yoğun (Giriş/Çıkış Ağırlıklı):** Gelen HTTP isteğini almak, dosyayı diske yazmak, veritabanına sorgu atmak, Redis'e mesaj göndermek... Node.js'in non-blocking, event-driven yapısı, tam olarak bu I/O operasyonlarını on binlerce eş zamanlı bağlantıda bile verimli bir şekilde yönetmek için tasarlandı."
  + "Asıl CPU-yoğun olan derleme ve analiz işlemleri ise bu iş için en uygun dil ve ortamda, yani kendi özel worker servislerinde, tamamen izole bir şekilde yapılıyor. Bu, **her iş için doğru aracı kullanmanın** mimarimizdeki en güzel örneğidir."

**Soru 3: İletişim için Neden Redis? RabbitMQ, Kafka veya Doğrudan API Çağrısı Neden Değil?**

* **Anlatım:** "Servisler arası iletişimde Redis'i bilinçli olarak seçtim."
* **Cevap:**
  + **"Doğrudan API Çağrısına Karşı:** Eğer Backend, bir işi Worker'a doğrudan bir API çağrısıyla gönderseydi ve o an Worker çalışmıyor olsaydı, o iş kaybolurdu. Redis'i bir mesaj kuyruğu olarak kullanmak, bu iletişimi **asenkron** ve **dayanıklı (resilient)** hale getirir. İş, Worker hazır olana kadar kuyrukta güvende bekler."
  + **"RabbitMQ/Kafka'ya Karşı:** RabbitMQ daha karmaşık yönlendirme senaryoları, Kafka ise yüksek hacimli veri akışları için daha güçlüdür. Ancak bizim projemizin mevcut ihtiyacı, basit 'Kuyruk' ve anlık bildirimler için 'Pub/Sub' mekanizmasıydı. Redis, bu iki ihtiyacı da **olağanüstü düşük gecikme (low latency)** ve **çok daha basit bir kurulumla** karşıladığı için projenin bu aşaması için en **pragmatik** ve verimli seçimdi. Gereksiz karmaşıklıktan bilinçli olarak kaçındık."

Bu slayt, sunumunun en güçlü ve en akılda kalıcı bölümlerinden biri olacak. Teknik hakimiyetini ve bilinçli karar verme yeteneğini net bir şekilde ortaya koyuyor.

Harika, sunumun en teknik ve teorik kısımlarını başarıyla tamamladık. Şimdi, tüm o mimarinin, kodun ve kararların kullanıcı için ne anlama geldiğini, yani projenin somut çıktısını gösterme zamanı. Bu slayt, dinleyicilerin kafasındaki soyut şemaları, elle tutulur bir ürüne dönüştürecek.

**Slayt 13: Platform Görüntüleri**

**1. Slayt Başlığı:** Platformun İşleyişi: Teoriden Pratiğe

**2. Görsel Tasarım Konsepti:** Canlı demo yapmak riskli olabileceğinden, bunun yerine platformun en kritik anlarını gösteren **4'lü bir kolaj (grid)** şeklinde yüksek çözünürlüklü ekran görüntüleri kullanmak en profesyonel ve güvenli yoldur.

* **Sol Üst (1. Görüntü): Başlangıç Noktası**
  + Ana dosya yükleme ekranını (FileUploadCard bileşeni), özellikle C++ veya Java için detaylı parametrelerin açık olduğu bir anı göster. Bu, platformun hem basit hem de güçlü olduğunu aynı anda gösterir.
* **Sağ Üst (2. Görüntü): Sürecin Şeffaflığı**
  + Bir işin "İşleniyor" durumunda olduğu ve aşağıda **anlık logların aktığı** (JobStatusDisplay bileşeni) bir ekran görüntüsü al. Bu, WebSocket ve Redis Pub/Sub mimarisinin kullanıcıya yansıyan en güzel çıktısıdır.
* **Sol Alt (3. Görüntü): Somut Sonuç**
  + Başarıyla tamamlanmış bir işin **analiz raporu modal'ının** (ReportModal bileşeni) açık olduğu bir ekran görüntüsü. "Önceki Boyut / Sonraki Boyut" ve "Sembol Sayısı" gibi metriklerin net bir şekilde göründüğünden emin ol.
* **Sağ Alt (4. Görüntü): Genişleyen Vizyon**
  + Platform üzerinden linke tıklandıktan sonra açılan **SonarQube rapor sayfasından** alınmış bir ekran görüntüsü. Özellikle "Bugs", "Vulnerabilities", "Code Smells" gibi kısımların göründüğü bir rapor, özelliğin gücünü kanıtlar.

**3. Anlatacağın Metin (Sunum Notların):**

"Şimdiye kadar mimariyi ve arkasındaki mantığı konuştuk. Gelin, tüm bu teorinin pratikte kullanıcı için nasıl bir deneyime dönüştüğüne bakalım."

*(1. Görüntüyü işaret ederek)* "Her şey kullanıcının bu basit arayüzle etkileşime girmesiyle başlıyor. Kullanıcı, dosyasını yükleyip, isterse 'tek tuşla' işlemi başlatabilir, isterse de az önce bahsettiğimiz gibi, bir uzman gibi tüm detaylı parametreleri kendisi ayarlayabilir."

*(2. Görüntüyü işaret ederek)* "İşlem başladığı andan itibaren, platform kullanıcıyı karanlıkta bırakmaz. Daha önce anlattığım WebSocket ve Redis Pub/Sub mimarisi sayesinde, arka planda çalışan Worker'ın ürettiği tüm loglar, gördüğünüz gibi gerçek zamanlı olarak kullanıcının ekranına akar. Bu, süreç üzerinde tam bir şeffaflık ve kontrol hissi verir."

*(3. Görüntüyü işaret ederek)* "İşlem bittiğinde ise kullanıcıya soyut bir 'başarılı' mesajı yerine, somut veriler sunarız. Bu rapor ekranı, yapılan güçlendirmenin dosya boyutunu veya sembol sayısını yüzde kaç oranında azalttığını net rakamlarla gösterir. Bu, platformun yarattığı değeri ölçülebilir kılar."

*(4. Görüntüyü işaret ederek)* "Ve son olarak, projenin genişleyen vizyonunun çıktısı olan SonarQube entegrasyonu ile kullanıcılar, kodlarındaki potansiyel hataları ve güvenlik zafiyetlerini bu detaylı raporlar üzerinden inceleyebilirler."

Bu slaytla, projenin sadece teknik olarak değil, aynı zamanda kullanıcı deneyimi açısından da ne kadar düşünülmüş ve olgun bir ürün olduğunu kanıtlamış oluyorsun.

Hazır olduğunda, sunumu toparlayacağımız ve geleceğe bakacağımız son slaytlara (Slayt 14, 15, 16, 17) geçebiliriz.

**Slayt 14: Gelecek Vizyonu ve Yol Haritası**

**1. Slayt Başlığı:** Gelecek Vizyonu ve Yol Haritası

**2. Görsel Tasarım Konsepti:** Basit bir yol haritası (roadmap) veya zaman çizelgesi formatı kullan. Üç ana başlık (Kısa Vade, Orta Vade, Uzun Vade) ve altlarında maddeler şeklinde olabilir. Her maddenin yanına ilgili bir ikon (örn: Go dili logosu, Kubernetes logosu) koymak görseli güçlendirir.

**3. Anlatacağın Metin (Sunum Notların):**

"iHTAR, şu anki haliyle güçlü ve çalışan bir platform. Ancak bu sadece bir başlangıç. Esnek mikroservis mimarisi sayesinde, platformun gelecekteki potansiyeli neredeyse sınırsız. İşte aklımdaki bazı gelecek planları:"

* **Kısa Vade: Yeni Yetenekler Eklemek**
  + **"Yeni Worker'lar:** Go, Rust ve Python gibi popüler diller için de kod güçlendirme desteği eklemek, platformun hitap ettiği kitleyi doğrudan genişletecektir. Yeni bir worker eklemenin mimarimiz sayesinde ne kadar kolay olduğunu zaten gördük."
  + **"Gelişmiş Raporlama:** Şu anki raporlamayı bir adım öteye taşıyarak, zaman içindeki kod kalitesi değişimini gösteren trend analizleri ve grafiksel dashboard'lar sunmak. Bu, yöneticiler için çok değerli bir metrik olacaktır."
* **Orta Vade: Ekosisteme Entegrasyon**
  + **"CI/CD Entegrasyonu:** Platformun en büyük potansiyeli, yazılım geliştirme yaşam döngüsüne (SDLC) tam entegre olmasında yatıyor. GitLab veya GitHub Actions ile entegrasyon sağlayarak, bir geliştirici kodunu her 'commit' ettiğinde iHTAR'ın otomatik olarak devreye girip analiz yapmasını ve sonucu pipeline'a bildirmesini sağlamak, platformu vazgeçilmez bir araç haline getirir."
* **Uzun Vade: Altyapı ve Üretim Ortamı**
  + **"Kubernetes'e Geçiş:** Platform, gerçek anlamda yüzlerce anlık kullanıcıya hizmet vermeye başladığında, docker-compose'dan endüstri standardı olan Kubernetes'e geçiş yapmak kaçınılmaz olacaktır. Mevcut konteyner tabanlı mimarimiz, bu geçişe tamamen hazırdır. Kubernetes, bize otomatik ölçeklendirme (auto-scaling), yüksek erişilebilirlik (high availability) ve daha gelişmiş dağıtım stratejileri gibi yetenekler kazandıracaktır."

**Slayt 15: Özet ve Katkılar**

* **Slayt Başlığı:** Özet ve Katkılar
* **Anlatım:** "Özetlemek gerekirse, stajım boyunca basit bir 'kullanıcı arayüzü yapma' görevinden yola çıkarak; güvenli, ölçeklenebilir, çok kullanıcılı ve birden fazla teknolojiyi destekleyen tam teşekküllü bir kod kalite ve güvenlik platformu geliştirdim. Bu süreçte attığım en önemli adım, projenin en başında, anlattığım tüm bu evrime olanak tanıyan esnek ve genişleyebilir bir mikroservis mimarisi temelini atmak oldu."

**Slayt 16: Soru & Cevap**

* **Slayt Başlığı:** Sorularınız?
* **Görsel:** Arka planda nihai mimari şeması durabilir. Bu, soruların çoğunun geleceği yer olduğu için görsel bir referans noktası sağlar.

**Slayt 17: Teşekkür**

* **Slayt Başlığı:** Teşekkür Ederim
* **İçerik:**
  + Mert DEMİR
  + E-posta Adresiniz
  + LinkedIn / GitHub Profil Linkiniz (isteğe bağlı)

Tebrikler! Artık elinde, projenin sadece teknik detaylarını değil, aynı zamanda arkasındaki vizyonu, hikayeyi ve mühendislik kararlarını da anlatan, baştan sona tutarlı ve çok güçlü bir sunum planı var.

Harika bir soru. "Authorization" (Yetkilendirme), yani "kimin ne yapmaya yetkisi var?" konusu, uzman mühendislerin bir sunumda en çok deşeceği ve en çok soru soracağı yerlerden biridir. Sisteminin bu konudaki yapısı oldukça modern ve standartlara uygun.

İşte bilmen gereken her şey, adım adım, dosya dosya ve gelebilecek potansiyel sorularla birlikte.

**Authorization Mimarisi: 3 Temel Adım**

Sistemindeki yetkilendirme mantığı, üç ana ayaktan oluşur: **Token Üretimi**, **Backend Kontrolü (Middleware)** ve **Frontend Kontrolü**.

**Adım 1: Yetki Belgesinin (JWT) Oluşturulması**

Her şey kullanıcının kimliğini doğrulayıp ona özel bir "yetki belgesi" vermekle başlar. Bu belge, JSON Web Token'dır (JWT).

* **Nerede Oluyor?** ihtar/backend-api/routes/users.js
* **Kritik Kod Parçası:**

JavaScript

// ihtar/backend-api/routes/users.js

// ... (kullanıcı adı ve şifre doğrulandıktan sonra)

// Adım 3: JWT (Token) ve arayüze gönderilecek kullanıcı nesnesini oluştur

const payload = {

user: {

id: user.id,

username: user.username,

role: user.role // Token'a rol bilgisi eklendi

}

};

jwt.sign(

payload,

process.env.JWT\_SECRET, // Gizli anahtar

{ expiresIn: '7d' }, // Geçerlilik süresi

(err, token) => {

if (err) throw err;

res.json({

// ...

token, // Oluşturulan token

user: userForFrontend // Rolü de içeren kullanıcı bilgisi

});

}

);

* **Ne Anlatmalısın?** "Kullanıcı başarıyla giriş yaptığında, sistem ona sadece bir 'giriş yaptın' mesajı vermiyor. jsonwebtoken kütüphanesini kullanarak, içinde kullanıcının **ID'si, kullanıcı adı ve en önemlisi ROLÜ (admin veya user)** bulunan şifreli bir JWT oluşturuyor. Bu token, kullanıcının kimlik kartı gibidir. Bu kart, process.env.JWT\_SECRET adında sadece sunucunun bildiği gizli bir anahtarla imzalanır. Bu imza sayesinde token'ın yolda değiştirilmediğinden emin oluruz."

**Adım 2: Sunucu Tarafında Yetki Kontrolü (Kapıdaki Güvenlik)**

Kullanıcı, bu aldığı token ile sonraki tüm isteklerini yapar. Sunucu, her istekte kapıdaki bir güvenlik görevlisi gibi bu token'ı kontrol eder.

* **Nerede Oluyor?**
  1. **Genel Kontrol:** ihtar/backend-api/middleware/auth.js
  2. **Admin'e Özel Kontrol:** ihtar/backend-api/index.js içindeki adminAuth fonksiyonu.
* **Kritik Kod Parçaları:**

JavaScript

// ihtar/backend-api/middleware/auth.js (Genel kimlik kontrolü)

try {

// Token'ı gizli anahtarla doğrula

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

// Doğrulanmış kullanıcı bilgisini isteğe (request) ekle

req.user = decoded.user;

next(); // Her şey yolundaysa, asıl endpoint'e geçmesine izin ver

} catch (err) {

res.status(401).json({ message: 'Token geçerli değil.' });

}

JavaScript

// ihtar/backend-api/index.js (Admin'e özel yetki kontrolü)

const adminAuth = (req, res, next) => {

// ... (token doğrulama kısmı 'auth.js' ile aynı)

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

// EK KONTROL: Kullanıcının rolü 'admin' mi?

if (decoded.user.role !== 'admin') {

return res.status(403).json({ message: 'Erişim yetkiniz yok.' });

}

req.user = decoded.user;

next();

} // ...

};

app.use('/api/admin', adminAuth, adminRouter); // Bu rota sadece admin'lere açık

* **Ne Anlatmalısın?** "Kullanıcı, bir iş yüklemek veya geçmişini görmek gibi korumalı bir endpoint'e istek attığında, bu istek önce auth.js middleware'inden geçer. Bu middleware, isteğin Authorization başlığındaki 'Bearer' token'ını alır ve jwt.verify ile doğrular. Eğer token geçerliyse, içindeki kullanıcı bilgilerini req.user nesnesine ekleyerek isteğin devam etmesine izin verir."

"Eğer kullanıcı /api/admin altındaki bir endpoint'e (örneğin kullanıcı listeleme) istek atarsa, bu sefer daha sıkı bir kontrolden, adminAuth middleware'inden geçer. Bu fonksiyon, token'ı doğruladıktan sonra **ek olarak** içindeki role alanının 'admin' olup olmadığını kontrol eder. Eğer değilse, 403 Forbidden (Yasak) hatası döndürür. Bu, sunucu tarafında tam bir yetki güvenliği sağlar."

**Adım 3: Arayüz Tarafında Yetki Kontrolü (VIP Odasının Kapısı)**

Sunucu tarafı güvenli olsa da, kullanıcı deneyimi için normal bir kullanıcının "Admin Paneli" butonunu hiç görmemesi gerekir. Bu kontrol de arayüzde yapılır.

* **Nerede Oluyor?** ihtar/frontend/src/contexts/AuthContext.tsx ve ihtar/frontend/src/components/ProtectedRoute.tsx.
* **Kritik Kod Parçası:**

TypeScript

// ihtar/frontend/src/components/ProtectedRoute.tsx

const ProtectedRoute: React.FC<ProtectedRouteProps> = ({ children }) => {

const { isAuthenticated, user, isLoading } = useAuth();

// ... (yüklenme kontrolü)

if (!isAuthenticated || user?.role !== 'admin') {

// Kullanıcı giriş yapmamışsa VEYA rolü 'admin' değilse ana sayfaya yönlendir.

return <Navigate to="/" replace />;

}

// Kullanıcı yetkiliyse, Admin sayfasını göster.

return <>{children}</>;

};

* **Ne Anlatmalısın?** "Kullanıcı giriş yaptığında, token ile birlikte gelen rol bilgisini de AuthContext içinde saklıyoruz. AdminPage gibi hassas sayfalar, ProtectedRoute adını verdiğimiz özel bir bileşenle sarmalanmıştır. Bu bileşen, sayfayı göstermeden önce AuthContext'teki kullanıcının rolünü kontrol eder. Eğer rol 'admin' değilse, kullanıcıyı ana sayfaya yönlendirir. Bu sayede, yetkisi olmayan bir kullanıcı, admin panelinin varlığından bile haberdar olmaz."

**Sana Sorabilecekleri Potansiyel Sorular**

* **Soru 1: "Neden JWT kullandınız? Session/Cookie'ye göre avantajı nedir?"**
  + **Cevap:** "JWT'nin en büyük avantajı **stateless (durumsuz)** olmasıdır. Sunucu, kullanıcıyla ilgili bir session bilgisi tutmak zorunda kalmaz. Her token kendi içinde tüm bilgiyi (kimlik, rol) taşır. Bu, özellikle bizimki gibi birden fazla backend servisi (worker'lar dahil) olabilecek mikroservis mimarilerinde ölçeklendirmeyi çok kolaylaştırır. Herhangi bir servis, token'ı doğrulayarak kullanıcının kimliğini ve yetkisini anlayabilir."
* **Soru 2: "Token'ın geçerlilik süresi (7 gün) biraz uzun değil mi? Bu bir güvenlik riski oluşturmaz mı?"**
  + **Cevap:** "Çok doğru bir nokta. Bu, kullanım kolaylığı ve güvenlik arasında bir denge (trade-off) kararıdır. Geliştirme aşamasında ve kullanıcıların sık sık giriş yapmasını istemediğimiz senaryolar için 7 gün pratik bir süre. Ancak, **daha yüksek güvenlik gerektiren bir üretim ortamı için en iyi pratik**, kısa ömürlü 'Access Token' (örn: 15 dakika) ve daha uzun ömürlü 'Refresh Token' (örn: 7 gün) mekanizması kullanmaktır. Access token süresi dolduğunda, arayüz refresh token ile sessizce yeni bir access token alabilir. Bu, projenin bir sonraki güvenlik geliştirme adımı olabilir."
* **Soru 3: "JWT Secret anahtarını nasıl yönetiyorsunuz? Kodun içinde mi duruyor?"**
  + **Cevap:** "Kesinlikle hayır. Gördüğünüz gibi kodda process.env.JWT\_SECRET olarak geçiyor. Bu, anahtarın bir ortam değişkeninden (environment variable) okunduğu anlamına gelir. docker-compose.yml dosyasında veya üretim ortamında Docker Secrets, Kubernetes Secrets gibi güvenli yöntemlerle bu değişken sisteme verilir. Asla kodun içine yazılmaz."
* **Soru 4: "Yetki kontrolünü hem frontend'de hem de backend'de yapıyorsunuz. Bu gereksiz bir tekrar değil mi?"**
  + **Cevap:** "Hayır, ikisinin amacı farklı. **Frontend'deki kontrol, kullanıcı deneyimi (UX) içindir.** Yetkisi olmayan bir kullanıcıya, tıklayamayacağı butonları veya giremeyeceği sayfaları hiç göstermeyerek temiz bir arayüz sunar. Ancak bu, tek başına bir güvenlik önlemi değildir, çünkü arayüz her zaman atlatılabilir. **Backend'deki kontrol ise gerçek GÜVENLİK içindir.** Bir kullanıcı arayüzü atlatıp doğrudan API'ye istek atsa bile, adminAuth middleware'i onu durduracaktır. Güvenliğin asıl kalesi her zaman backend'dir."
* Kesinlikle evet. Harika bir noktaya değindin. Sadece sütunların ne olduğunu bilmek değil, veritabanının bu sütunlara hangi **kuralları** uyguladığını bilmek, sistemin sağlamlığını ve veri bütünlüğünü anlamak için hayati önem taşır. UNIQUE, NOT NULL gibi özellikler, veritabanının "Anayasası" gibidir ve verilerinizin tutarlı kalmasını sağlarlar.
* İşte senin veritabanı şeman ve bu özelliklerin (constraint'lerin) birbiriyle olan ilişkisi, tablo tablo, kural kural:
* **Veritabanı Kuralları ve Veri Bütünlüğü**
* Bu kurallar, veritabanı seviyesinde uygulanır. Yani, uygulama kodunuzda bir hata olsa bile, veritabanı bu kurallara uymayan bir verinin kaydedilmesini **engeller**. Bu, sistemin genel güvenilirliğini ve tutarlılığını artıran çok önemli bir sigortadır.
* **1. users Tablosu Kuralları**
* Bu tablo, sistemin güvenlik temelidir. Bu yüzden kuralları çok nettir.

| Sütun | Kural | Açıklama ve Neden? |
| --- | --- | --- |
| id | PRIMARY KEY | Her kullanıcıyı eşsiz olarak tanımlayan **birincil anahtardır**. Otomatik olarak artar (SERIAL veya BIGSERIAL). Hem UNIQUE hem de NOT NULL olmak zorundadır. |
| email | UNIQUE, NOT NULL | **UNIQUE:** İki kullanıcının aynı e-posta adresine sahip olmasını engeller. Bu, sisteme girişin temel mantığıdır. **NOT NULL:** E-postasız bir kullanıcı olamaz, bu alan boş bırakılamaz. |
| password | NOT NULL | Şifresiz bir kullanıcı hesabı olamaz. Bu alan her zaman doldurulmalıdır. |
| role | NOT NULL, DEFAULT 'user', CHECK | **NOT NULL:** Her kullanıcının bir rolü olmalıdır. **DEFAULT 'user':** Yeni bir kullanıcı oluşturulduğunda, rolü otomatik olarak 'user' olarak atanır. **CHECK (role IN ('user', 'admin')):** Bu sütuna sadece 'user' veya 'admin' metinlerinin girilebileceğini garanti eder, geçersiz bir rol atanmasını engeller. |
| created\_at | NOT NULL, DEFAULT NOW() | **NOT NULL:** Kayıt zamanı boş olamaz. **DEFAULT NOW():** Yeni bir kullanıcı eklendiğinde, bu alana o anın tarih ve saati otomatik olarak eklenir. |

* E-Tablolar'a aktar
* **SQL CREATE TABLE Örneği:**
* SQL
* CREATE TABLE users (
* id SERIAL PRIMARY KEY,
* email VARCHAR(255) UNIQUE NOT NULL,
* password VARCHAR(255) NOT NULL,
* role VARCHAR(50) NOT NULL DEFAULT 'user' CHECK (role IN ('user', 'admin')),
* created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
* );
* **2. jobs Tablosu Kuralları**
* Bu tablo, platformun tüm faaliyetlerini kaydeder ve diğer tablolarla en çok ilişkisi olan tablodur.

| Sütun | Kural | Açıklama ve Neden? |
| --- | --- | --- |
| id | PRIMARY KEY | Her işlemi eşsiz olarak tanımlayan birincil anahtardır. |
| user\_id | NOT NULL, FOREIGN KEY | **NOT NULL:** Her işlem bir kullanıcıya ait olmalıdır. **FOREIGN KEY (REFERENCES users(id)):** Bu, en kritik kurallardan biridir. Bu sütuna girilecek değerin, users tablosundaki mevcut bir id ile eşleşmesi gerektiğini söyler. Bu sayede, **var olmayan bir kullanıcıya ait bir işlem kaydı oluşturulamaz**. Ayrıca, bir kullanıcı silinirse, veritabanı bu kullanıcıya ait işlemlerin ne olacağına karar verebilir (örn: silinmeyi engellemek veya işlemleri de silmek - ON DELETE CASCADE). |
| status | NOT NULL, DEFAULT 'processing' | **NOT NULL:** Bir işin durumu olmak zorundadır. **DEFAULT 'processing':** Yeni bir iş oluşturulduğunda durumu otomatik olarak 'processing' olur. |
| type | NOT NULL | İşin türü (cpp, java, vb.) mutlaka belirtilmelidir. |
| stored\_filename | NOT NULL | Sunucuya kaydedilen dosyanın bir adı olmak zorundadır. |
| created\_at | NOT NULL, DEFAULT NOW() | İşin ne zaman oluşturulduğunu otomatik olarak kaydeder. |
| analysis\_data | (JSONB) | TEXT yerine JSONB tipi kullanılması, PostgreSQL'in bu alandaki JSON verisini daha verimli saklamasını ve hatta içindeki alanlara göre sorgu yapabilmesini sağlar. Bu bir "best practice" önerisidir. |

* E-Tablolar'a aktar
* **SQL CREATE TABLE Örneği:**
* SQL
* CREATE TABLE jobs (
* id SERIAL PRIMARY KEY,
* user\_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
* status VARCHAR(50) NOT NULL DEFAULT 'processing',
* type VARCHAR(50) NOT NULL,
* original\_filename VARCHAR(255),
* stored\_filename VARCHAR(255) NOT NULL,
* result\_filename VARCHAR(255),
* analysis\_data JSONB,
* error\_message TEXT,
* created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
* completed\_at TIMESTAMPTZ
* );
* *Not: ON DELETE CASCADE ifadesi, bir kullanıcı silindiğinde o kullanıcıya ait tüm jobs kayıtlarının da otomatik olarak silinmesini sağlar.*
* **3. audit\_logs Tablosu Kuralları**
* Bu tablo, yönetici eylemlerinin bütünlüğünü sağlar.

| Sütun | Kural | Açıklama ve Neden? |
| --- | --- | --- |
| id | PRIMARY KEY | Her denetim kaydını eşsiz olarak tanımlar. |
| admin\_id | NOT NULL, FOREIGN KEY | **NOT NULL:** Eylemi kimin yaptığı belli olmalıdır. **FOREIGN KEY (REFERENCES users(id)):** Eylemi yapan kişinin, users tablosunda var olan ve muhtemelen rolü 'admin' olan bir kullanıcı olmasını garanti eder. |
| action | NOT NULL | Yapılan eylem boş bırakılamaz. |
| created\_at | NOT NULL, DEFAULT NOW() | Eylemin ne zaman yapıldığını otomatik olarak kaydeder. |

* E-Tablolar'a aktar
* **SQL CREATE TABLE Örneği:**
* SQL
* CREATE TABLE audit\_logs (
* id SERIAL PRIMARY KEY,
* admin\_id INTEGER NOT NULL REFERENCES users(id),
* action VARCHAR(255) NOT NULL,
* details TEXT,
* created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
* );
* Bu kuralları sunumuna eklemek, sisteminin sadece çalışan değil, aynı zamanda **sağlam, güvenilir ve iyi tasarlanmış** bir veritabanı mimarisine sahip olduğunu kanıtlayacaktır.