

Web Programlama I

Ders 02 - Bash, Regex, Build Araçları ve Test

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

Eğitmen: Ömür ŞAHİN

1 Genel Bakış

2 Bash

3 Düzenli İfadeler (Regular Expression)

4 Build Araçları

5 Test

1-Genel Bakış

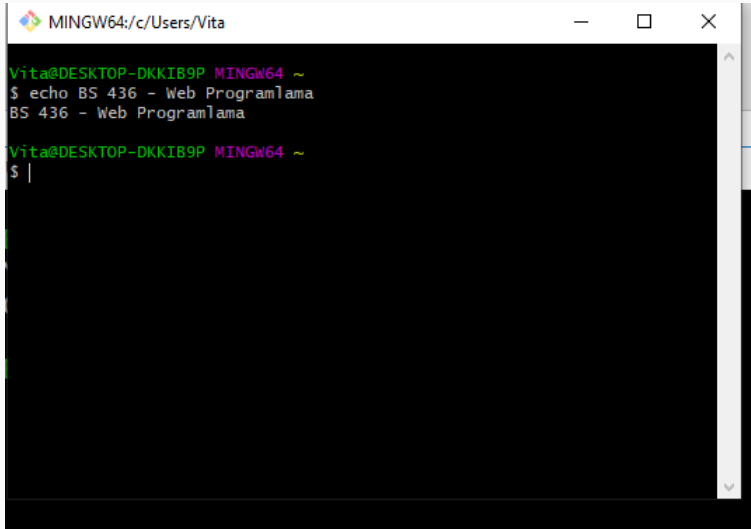
- Bash terminaline giriş
- Düzenli İfadelere (regular expression) giriş
- Build araçları: YARN, Npm, Pnpm, WebPack ve Babel
- Test senaryosu nasıl yazılır?

2-Bash

- Bash, Linux/Mac/Unix sistemlerde kabuktur (shell) ve komut dilidir.
- Pek çok farklı shell bulunmaktadır.
 - PowerShell Windows'ta kullanılmaktadır.
- Shell ayrıca terminal, konsol (console), komut satırı (command-line) gibi isimlendirmelere sahiptir.
- Komut yazıp çalıştırmaya olanak tanır.

- Bash, Linux/Mac/Unix sistemlerde kabuktur (shell) ve komut dilidir.
- Pek çok farklı shell bulunmaktadır.
 - PowerShell Windows'ta kullanılmaktadır.
- Shell ayrıca terminal, konsol (console), komut satırı (command-line) gibi isimlendirmelere sahiptir.
- Komut yazıp çalıştırmaya olanak tanır.

Bash



```
MINGW64:/c/Users/Vita

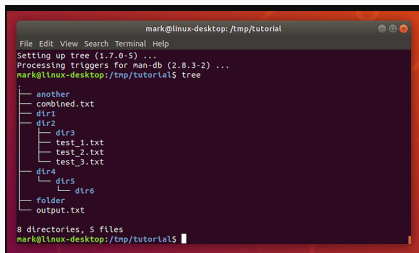
Vita@DESKTOP-DKKIB9P MINGW64 ~
$ echo BS 436 - Web Programlama
BS 436 - Web Programlama

Vita@DESKTOP-DKKIB9P MINGW64 ~
$ |
```


Neden?

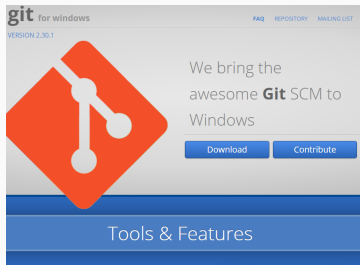
- Program yazıyorsanız oldukça kritik öneme sahiptir.
- Pek çok görevi otomatize etmeye fayda sağlar.
- Pek çok sunucu bir GUI sağlamaz.
 - Terminal kullanarak SSH üzerinden erişim sağlanır.
 - Gömülü ve IoT cihazlarına erişim için kullanılır.
- Temel komutları bilmeniz gerekir.
- Build araçlarını yürütmek için Bash komutlarına ihtiyaç duyulur.

- Linux/Mac kullanıyorsanız, kurulu olarak gelmektedir.
- Eğer Windows kullanıyorsanız GitBash kullanılması önerilmektedir.
 - <http://gitforwindows.org>



```
mark@linux-desktop: /tmp/tutorial
File Edit View Search Terminal Help
Setting up tree (1.7.0-5) ...
Processing triggers for man-db (2.8.3-2) ...
mark@linux-desktop: /tmp/tutorial$ tree
.
├── another
├── combined.txt
├── dir1
├── dir2
│   ├── dir3
│   │   ├── test_1.txt
│   │   ├── test_2.txt
│   │   └── test_3.txt
├── dir4
│   └── dir5
│       └── dir6
├── folder
└── output.txt

8 directories, 5 files
mark@linux-desktop: /tmp/tutorial$
```



- **.**: Mevcut klasör
- **..**: Üst klasör
- **~**: Home klasörü
- **pwd**: Mevcut klasörü yazdır (**P**ring **W**orking **D**irectory)
- **cd**: Klasör değiştir (**C**hange **D**irectory)
- **mkdir**: Klasör oluştur
- **ls**: Klasör içeriğini listele
- **cp**: Dosya kopyala
- **mv**: Dosya taşı
- **rm**: Sil ("-r" rekürsif olarak sil)
- **man**: Komutun kullanım kılavuzu

- **echo**: Girdi değerini yazdır
- **cat**: Dosya içeriğini yazdır
- **less**: Dosya içeriğini kaydırabilir (scroll) olarak yazdır.
- **>**: Yönlendirme
- **>>**: Ekleme
- **|**: Pipe komutu
- **which**: Programın konumu
- **\$**: Değişken yazdırırken kullanılır (örnek \$VAR)
- **wc**: Kelime saydır
- **find**: Arama
- **grep**: Regular Expressiona göre ayıkla
- **touch**: Erişim zamanını güncelle eğer dosya yoksa oluştur.

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ pwd

/d/Ders/Ornekler

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ ls

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ echo "Keyifler nasıl gençler" > test.txt

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ ls

test.txt

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ cat test.txt

Keyifler nasıl gençler

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ |



```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler
$ mkdir testKlasoru
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler
$ ls
test.txt  testKlasoru/
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler
$ cd testKlasoru/
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ pwd
/d/Ders/Ornekler/testKlasoru
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls ..
test.txt  testKlasoru/
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ |
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ cp ../test
test.txt      testKlasoru/
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ cp ../test.txt ./klasordeDosya.txt
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ cat klasordeDosya.txt
Keyifler nasıl gençler
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ mv ../test.txt .
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls
klasordeDosya.txt  test.txt
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ |
```

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru

\$ echo \$PATH

/c/Users/Omur/bin:/mingw64/bin:/usr/local/bin:/usr/bin:/bin:/mingw64/bin:/usr/bin:/c/Users/Omur/bin:/e/Python/Python37/Scripts:/e/Python/Python37:/e/Program Files/doptOpenJDK/jdk-11.0.10.9-hotspot/bin:/c/WINDOWS/system32:/c/WINDOWS:/c/WINDOWS/system32/Wbem:/c/WINDOWS/System32/WindowsPowerShell/v1.0:/cmd:/c/WINDOWS/System32/enSSH:/c/Users/Omur/AppData/Local/Microsoft/WindowsApps:/c/Users/Omur/AppData/Local/gitkraken/bin:/e/Program Files/Microsoft VS Code/bin:/e/texlive/2020/bin/win32/USERPROFILE%/AppData/Local/Microsoft/WindowsApps:/usr/bin/vendor_perl:/usr/bin/perl

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru

\$ which bash

/usr/bin/bash

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru

\$ |

- Projedeki JavaScript dosya sayısını nasıl elde edersiniz?
- Bütün bu dosyalardaki satır sayısını nasıl elde edersiniz?

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Calismalar/Github/futwebapp-tampermonkey (master)
$ find . -regex '^\.*\.jsx?' -not -path */node_modules/* | wc -l
38

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Calismalar/Github/futwebapp-tampermonkey (master)
$ cat 'find . -regex '^\.*\.jsx?' -not -path */node_modules/*' | wc -l
5317
```

- "find" komutu rekürsif olarak bütün "." klasörünü arar.
 - Mac sistemlerde "-E" parametresini kullanmalısınız (find -E).
- JS/JSX dosyaları ile eşleşen regular expression:
 - `^` Dosya adının başlangıcı
 - `.*` Herhangi bir karakter (.) herhangi bir sayı (*)
 - `"\."` `"\"` karakteri (`"\"` escape karakteri olarak kullanılıyor)
 - `"\.jsx?"` x opsiyonel olarak şekilde dosya adının sonu.
 - `"-not -path */node_modules/*"` klasörü hariç
- `"| wc -l"`: dosya isimlerini satır saydırma programına verir.
- `cat 'x':` " içindeki komutu çalıştırır ve çıktısını terminale verir.
 - Böylelikle JS/JSX dosyaları içindeki her şeyi cat ile yazdırmış oluruz.

- Yukarı/Aşağı butonları ile geçmiş komutlara ulaşabilirsiniz (history ile de erişilebilir ve !komut numarası ile çağırılabilir.)
- "tab" tuşu ile yazılacak kelime tamamlanabilir.
- Bash komutları çalıştırılabilir script haline çevirilebilir.
 - ".sh" uzantısı kullanılabilir.
 - İlk satırın "#!<pathToBash>" olarak başlaması gerekmektedir. (Örnek, "#!/usr/bin/bash")
 - Diğer programlar gibi terminalde çalıştırılabilir olacaktır.

3-Düzenli İfadeler (Regular Expression)

- "erciyes.edu.tr" geçerli bir mail adresi midir?
 - @ sembolü içermemektedir. O yüzden hayır.
- "03ASD451ASD1245456ASDSAD45" geçerli bir telefon numarası mıdır?
 - Sayı olmayan ifadeler bulunmaktadır ve çok uzundur. O yüzden hayır.
- String ifade özel bir formatta kısıtlara sahip ise Regex kullanılabilir.
 - Geçerli bir mail belirten string ifadeler.
 - Genellikle HTML formlarının girdileri kontrol edilirken kullanılır.

Eşleşme Kuralları

- `"."` karakteri wildcard olarak kullanılır.
 - `"a.b"` ifadesi a ile başlayan ve b ile biten 3 harfli ifadeleri temsil eder.
- `"[]"` ifadesi set içerisindeki tek bir karakter ile eşleşir.
 - `"[abc]"` ifadesi "a", "b" ve "c" harfleri ile eşleşirken, "d", "ab" gibi karakterlerle eşleşmemektedir.
- `"[-]"` ifadesi aralığı belirtir.
 - `"[a - z]"` ifadesi bütün küçük harflerle eşleşirken, `[a - zA - Z]` ifadesi bütün harflerle eşleşir. `[0 - 8]` ise 9 hariç bütün sayılarla eşleşir.
- Özel ifadeler kullanmak için `\` kullanılmalıdır.
 - `"\\[\\.\\]"` ifadesi `"[.]"` ile eşleşir. a veya `[a]` ifadesi ile eşleşmemektedir.

- **()** regex'in sınırlarını belirlemede kullanılır.
- **|** or operatörü olarak çalışır.
- ***** önceki regex tanımını 0 veya daha fazla tekrarlar.
- **"ab*", "a", "ab" ve "abbbbb"** ile eşleşir.
- **"(ab)*", "",ab ve "abababababab"** ile eşleşir.
- **"(ab)|c" "ab" ve "c" ile eşleşir.**

- "+" en az 1 sefer demektir.
 - "x+" \rightarrow "xx*"
- "?" 0 veya 1 sefer demektir.
 - "x?" \rightarrow "boş karakter | x"
- "{" kaç kere tekrarlanacağını belirtir.
 - "x{5}" \rightarrow "xxxxx"
 - "x{2,4}" \rightarrow "(xx)|(xxx)|(xxxx)"

Örnek: Telefon Numarası

- 10 Haneli Numara:
 - Örnek: 3522076666
- + ile başlayan 2 haneli ülke kodu da opsiyonel olarak bulunmaktadır.
 - Türkiye kodu: +90

$((\backslash +)[0 - 9]\{2\})?[0 - 9]\{10\}$

- $(\backslash +)[0 - 9]\{2\})?$ → Opsiyonel olarak + ile başlayan 2 haneli sayı
- $[0 - 9]\{10\}$ → 10 adet 0-9 aralığında sayı

Regex'in kısıtları

- Regex kısıtları olan string ifadelerin geçerliliğini kontrol etmek için oldukça faydalıdır.
- Bütün kısıtları tanımlamada yeterli değildir.
- Örnek: Bir string ifadenin geçerli bir JavaScript kodu olup olmadığı Regex ile anlaşılamaz. Bunun için Context-Free-Grammar kullanılır.

4-Build Araçları

- JS geliştirmek için açık kaynak kütüphaneler oldukça önemlidir.
 - En önemli kütüphanelerden biri derste de kullanacağımız React kütüphanesidir.
- JS'te YARN, NPM, PNPM gibi araçlar bulunmaktadır.
- YARN, NPM ve PNPM aynı bağımlılık depolarına erişim sağlamaktadır.
- Pnpm'da her sürüm dosyasını ayrı ayrı tutmak yerine farklı dosyaları depolamaktadır bu da daha az yer kaplamasını sağlamaktadır.
- Terminal komutları ile kullanılmaktadır.

```
MINGW64:/f/Ders/BashExamples/InitExample
Vita@DESKTOP-DKKIB9P MINGW64 /f/Ders/BashExamples/InitExample
$ yarn init -y
yarn init v1.19.1
warning The yes flag has been set. This will automatically answer yes to all questions, which may have security implications.
success Saved package.json
Done in 0.13s.

Vita@DESKTOP-DKKIB9P MINGW64 /f/Ders/BashExamples/InitExample
$ yarn install
yarn install v1.19.1
info No lockfile found.
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
Done in 0.45s.

Vita@DESKTOP-DKKIB9P MINGW64 /f/Ders/BashExamples/InitExample
$ |
```

- npm init
 - Mevcut klasörde yeni proje oluşturmak için gerekli package.json oluşturmak için kullanılır.
- npm install
 - node_modules klasörüne package.json dosyasında tanımlı bağımlılıkları indirip kurmak için kullanılır.

- Projenin ana ayar dosyasıdır.
- Maven Java projelerindeki pom.xml dosyasına benzemektedir.
- Üç temel bileşeni bulunmaktadır.
 - **scripts**: YARN/PNPM tarafından çalıştırılabilir komutlar bulunmaktadır. (Örnek: build, run vs.).
 - **dependencies**: Projede kullanılan bağımlılıklar belirtilir.
 - **devDependencies**: Yalnızca geliştirme ortamında kullanılan bağımlılıklar belirtilir. Uygulamanın son halinde kullanılmaz (WebPack gibi).

- Bağımlılıklar kurulduğunda, yarn.lock adında bir dosya oluşturulur.
- Bağımlılıklar tanımlandığında hangi sürüm olduğuna dair bir bilgi de içerir (1.1.13 gibi).
- \wedge karakteri en yakın minör versiyon ile eşleşmektedir.
 - Versiyonlama: majör.minör.patch
 - Örnek: \wedge 1.0.3 tanımı 1.5.7 ile eşleşirken, 2.0.0 ile eşleşmemektedir.
- yarn.lock bu dosya oluşturulduğunda kullanılan versiyonları barındırmaktadır.
 - Takım çalışmalarında oldukça önemlidir. Gerçekleştirilen güncellemeler projede bug oluşmasına sebep olabilir.

Modern Build Araçları Karşılaştırması

- **Webpack:** Mature, geniş ekosistem, karmaşık konfigürasyon
- **Vite:** Hızlı HMR, basit konfigürasyon, modern ES modules
- **Parcel:** Zero-config, otomatik optimizasyon
- **Rollup:** Kütüphane geliştirme için optimize

Webpack Avantajları:

- Geniş plugin ekosistemi
- Legacy browser desteği
- Micro-frontend desteği
(büyük uygulamalar küçük bağımsız parçalara bölünür)

Vite Avantajları:

- Çok hızlı dev server
- Minimal konfigürasyon
- Native ES modules (tarayıcı doğal modül sistemini kullanır ve sadece değişen modüller tekrar yüklenir)

Vite: Modern Build Aracı

- Evan You (Vue.js yaratıcısı) tarafından geliştirilmiştir.
- ESBuild tabanlı süper hızlı dev server
- Hot Module Replacement (HMR) ile anlık değişiklik görme
- Production build için Rollup kullanır
- TypeScript, JSX, CSS pre-processor'ları built-in destekler

Dev server başlatma süreleri:

Webpack: 10-30 saniye

Vite: 1-2 saniye

Vite Projesi Kurulumu

```
1  "scripts": {  
2    "dev": "vite",  
3    "build": "vite build",  
4    "preview": "vite preview"  
5  },  
6  "devDependencies": {  
7    "vite": "^5.0.0",  
8    "@vitejs/plugin-react": "^4.0.0"  
9  }
```

- **vite**: Dev server başlatır
- **vite build**: Production build oluşturur
- **vite preview**: Build'i önizleme için server başlatır

Konfigürasyon Karşılaştırması

Webpack (webpack.config.js):

```
1 module.exports = {  
2   entry: './src/index.js',  
3   output: {  
4     path: path.resolve(__dirname,  
5       'dist'),  
6     filename: 'bundle.js'  
7   },  
8   module: {  
9     rules: [{  
10      test: /\.jsx?$/,  
11      use: 'babel-loader'  
12    }]  
13  },  
14  plugins: [  
15    new HtmlWebpackPlugin()  
16  ]  
};
```

Vite (vite.config.js):

```
1 import { defineConfig } from  
2   'vite'  
3  
4 import react from  
5   '@vitejs/plugin-react'  
6  
7 export default defineConfig({  
8   plugins: [react()]  
9 })
```

Çok daha basit konfigürasyon!

Webpack vs Vite: Hangisini Seçmeli?

- **Vite'i tercih edin:**

- Yeni projeler başlatırken
- Hızlı development deneyimi istiyorsanız
- Modern browser desteği yeterli ise
- React, Vue, Svelte gibi modern framework'ler

- **Webpack'i tercih edin:**

- Legacy browser desteği gerekiyorsa
- Karmaşık build gereksinimleri varsa
- Mevcut büyük projelerde (migration maliyeti)
- Micro-frontend mimarisi kullanıyorsanız

- **Create React App (CRA):** Artık deprecated
- **Vite Template:**
 - `npm create vite@latest my-app --template react`
 - `npm create vite@latest my-app --template react-ts`
- **Next.js:** Full-stack React framework
 - `npx create-next-app@latest`

Kod Dönüştürme (Transformation)

- Paketleme yeterli olmamaktadır. Bazen kod dönüşümüne de ihtiyaç duyulmaktadır.
- TypeScript, JSX gibi diller desteklenmektedir.
 - Browser tarafından native olarak desteklenmeyen ve JS'e dönüştürülmesi gereken dillerdir.
 - React için JSX oldukça önemlidir.
- Eski browser desteği
 - Örnek: JS'in yeni özelliklerini eski eşdeğerlerine çevirmek için kullanılmaktadır.
- Boyut Küçültme
 - Yorum satırlarını ve boşluklar kaldırılarak daha hızlı indirilebilir küçük boyutta JS dosyaları oluşturulmaktadır.
- vs.

- **Babel:** Mature, geniş plugin ekosistemi, karmaşık transformations
- **SWC:** Rust tabanlı, 20x daha hızlı
- **ESBuild:** Go tabanlı, Vite'da kullanılır
- **TypeScript Compiler:** TS için native çözüm

Transformation Hızları:

Babel: 1x (baseline)

SWC: 20x daha hızlı

ESBuild: 25x daha hızlı

Ne Zaman Hangi Tool'u Kullanmalı?

Babel'i tercih edin:

- Legacy browser desteği gerekiyorsa
- Karmaşık polyfill ihtiyaçları varsa
- Özel plugin'lere ihtiyaç varsa
- Enterprise projelerinde
- Maksimum browser compatibility

SWC/ESBuild tercih edin:

- Sadece modern browser hedefliyorsan
- Hız en önemli faktörse
- Basit transformation yeterli ise
- Vite/modern build tools kullanıyorsan

Transformation Performance

- **Build Time:** SWC ve ESBUILD çok daha hızlı
- **Bundle Size:** Babel daha optimize polyfill
- **Compatibility:** Babel en geniş browser desteği
- **Ecosystem:** Babel'in en zengin plugin ekosistemi

Kurulması Gereken Araçlar

- node kurulu olarak gelmeyen işletim sistemi kullanıyorsanız kurmanız gerekmektedir. <https://nodejs.org>
 - "node -version" komutu ile kontrol edebilirsiniz.
- <https://yarnpkg.com> adresinden YARN kurulmalıdır.
 - "yarn -version" komutu ile kontrol edebilirsiniz.
- WebPack/Babel/Jest/pnpm/npm: Manuel kuruluma gerek yoktur.
 - "yarn install" komutu kurulumu sizin için yapacak ve package.json içindeki devDependencies kısmına ekleyecektir.
 - Kurulan araçlar yerel olarak node_modules klasörüne eklenecektir. Böylelikle package.json içerisindeki scripts alanında kullanılabilir olacaktır.

5-Test

Modern JavaScript Testing

- Programın doğru çalışıp çalışmadığını kontrol etmek için test senaryolarının yazılması oldukça önemlidir.
- Dinamik yazımlı dillerde ise çok daha fazla önemlidir. Çünkü derleyicilerin yaptığı pek çok uyarıyı ve kontrolü yapmamaktadır.
- JS testi için pek çok kütüphane bulunmaktadır ancak bu ders kapsamında **Vitest** kullanılacaktır.
- Vitest modern, hızlı ve Vite ekosistemi ile uyumlu bir test framework'üdür.

Jest:

- Mature ve stabil
- Geniş ekosistem
- Facebook tarafından geliştirildi
- Babel transformation gerektirir

Vitest:

- Çok hızlı (Vite powered)
- Modern ES modules
- Jest compatible API
- Native TypeScript desteği

Test Çalıştırma Süreleri:

Jest: 10-15 saniye

Vitest: 2-3 saniye

Vitest Kurulumu ve Ayarları

- Vitest'in kurulumu ve ayarları çok daha basittir.
- Vite konfigürasyonu ile entegre çalışır.

```
1      "scripts": {  
2          "test": "vitest",  
3          "test:run": "vitest run",  
4          "test:coverage": "vitest  
              --coverage"  
5      },  
6      "devDependencies": {  
7          "vitest": "^1.0.0",  
8          "@vitest/ui": "^1.0.0",  
9          "jsdom": "^23.0.0"  
10     }
```

- Vitest minimal konfigürasyon gerektirir.
- Vite konfigürasyonu ile aynı dosyayı paylaşabilir.

```
1      import { defineConfig } from 'vitest/config'
2      import react from '@vitejs/plugin-react'
3
4      export default defineConfig({
5          plugins: [react()],
6          test: {
7              environment: 'jsdom',
8              globals: true,
9              setupFiles: ['./src/test/setup.js']
10         }
11     })
```


Vitest'in Avantajları

- **Hız:** ESBUILD tabanlı transformation
- **Hot Reload:** Test dosyalarında anında değişiklik görme
- **Native ES Modules:** Import/export'ları doğrudan destekler
- **TypeScript:** Built-in TypeScript desteği
- **Vite Integration:** Aynı konfigürasyon dosyası
- **Modern:** 2022+ geliştirilmeye başlandı
- **Watch Mode:** Sadece değişen dosyaları test eder