Kadir ÇAPKIN 21360859023

Öncelikle bu hafta derse, geçen hafta ders sonunda bahsedilen arrow function örneğiyle başladık. Yaptığımız örnek şu şekilde işliyor: Task adında bir nesne oluşturuldu bu nesne içerisine görevlerin tutulduğu tasks adli bir array ve yapılması gereken görevleri getiren getTaskToDo() adli fonksiyonu içermektedir. Task arrayi ise içerisinde mevcut görevleri nesneler şeklinde tutar.

```
const tasks = {
    tasks : [
        {
            text : "Alışveriş",
            completed : true
        },
        {
            text : "Temizlik",
            completed : false
        },
        {
            text : "Ödev",
            completed : false
        }

    ],
    getTasksToDo : function (){
        return this.tasks.filter((tasks) => tasks.completed === false) // bu metod bize geriye bir array dondurur task arrayini taramaya baslar ve callback fonksiyonu içerisinde arrayin indexlerini tek tek tarar eger sonuc false ise geri dondurulecek arraye ekler.
    }
}
```

Yukarıdaki kodda asıl önemli kısım getTaskToDo() fonksiyonu içerisinde yazdığımız this.task.filter() fonksiyonudur. This sözcüğü burada oluşturulan içerisinde bulunan nesne'yi kullandığımızı gösterir. Bu fonksiyon içerisine aldığı callback fonksiyonuyla birlikte kullanılan array'in her bir elemanını gezer ve eğer görev nesnesinin completed attirbute değeri false ise bu nesneyi başka bir arrayin içerisine atar ve her eleman tarandığı için yapılmayan görevler belirlenmiş olur. Eğer

kod içerisinde getTaskToDo() fonksiyonunu çağırdığımızda karşımıza şu sonuç çıkmaktadır.

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi> node arrow_test.js
[
  { text: 'Temizlik', completed: false },
  { text: 'Ödev', completed: false }
```

Arrow function ile ilgili bu örneği yaptıktan sonra not uygulamamıza geri döndük ve daha önce oluşturduğumuz bütün fonksiyonları arrow function haline getirdik. Örneğin:

```
yargs.command({
  command: "list",
  describe: "mevcut notu listeler",
  handler () { // Degistirildi
    notes.listNotes()
  },
});

const listNotes = () => { // Degistirildi
  const notes = loadNotes()
  console.log(chalk.inverse('Kayitli notlar'))
  notes.forEach((note) => {
      console.log(note.title)
  });
}
```

Bu fonksiyonları değiştirdikten sonra dersin önemli kısmı olan son kısımda ise oluşturduğumuz not uygulamasındaki özellikleri biraz daha işlevsel hale getirmeye çalıştık. İlk olarak add note fonksiyonumuzu düzenledik burada yapmak istediğimiz şey daha önce not eklerken başlık bilgisini ayni girdiğimizde de notes.json dosyamıza aynı başlıkta yazıyorduk. Bu durumu istemediğimiz için kontrollü koşullar ekledik.

```
const addNote = (title, body) => {
  // notlar dosyaya kaydediliyor
  const notes = loadNotes(); //array dondurcek
  const duplicateNote = notes.find((note) => note.title === title); //eger notlar
icerisinde bulunursa o not dondurulur

  console.log(duplicateNote) // eğer array içerisinde ayni baslik bulunduysa nesneyi
donduruyor. eger bulunmadiysa undefined dondurur.
  //debbugger

  if (!duplicateNote) {
    notes.push({
      title: title,
      body: body,
    });
    saveNotes(notes);
    console.log(chalk.green.inverse("yeni not eklendi")); //

  } else {
    console.log(
      chalk.red.inverse("Bu başlık daha once kullanıldı.Not eklenemiyor!!!")
    );
  }
};
```

Yukarıdaki kodda işlev şu şekilde öncelikle daha önce yazmış olduğumuz loadNotes() fonksiyon bir degiskende tuttuk. Bu fonksiyon bize json formatta bilgileri dondurur yani nesne tutan bir array şeklinde. Ardindan önemli kısım olan:

```
  const duplicateNote = notes.find((note) => note.title === title); //eger notlar
icerisinde bulunursa o not dondurulur
```

Find metodunu inceleyecek parametre olarak aldığı callback fonksiyonu tek satırda yazılmış yani eğer belirtilen koşul varsa return edilecek. Yaptığı görev ise şu notes arrayinde sırasıyla tüm indexleri geziyor (callback içerisindeki note her bir array elemanı) ardından eğer eklenmek istenen title notlar içerisinde bulunduysa bulunan nesne duplicateNote değişkenine return eder.

```
if (!duplicateNote) {
    notes.push({
```

```
      title: title,
      body: body,
    });
    saveNotes(notes);
    console.log(chalk.green.inverse("yeni not eklendi")); //

  } else {
    console.log(
      chalk.red.inverse("Bu başlık daha once kullanıldı.Not eklenemiyor!!!")
    );
  }
```

Eğer bu değişken bir değer bulunduruyorsa girmek istediğimiz title zaten mevcut demektir o yüzden biz mevcut olmayan kısımda notumuzu ekleyebiliyor olmamız gerekir.If koşulunun içerisinde ise

```
    notes.push({
      title: title,
      body: body,
    });
    saveNotes(notes);
    console.log(chalk.green.inverse("yeni not eklendi")); //
```

mevcut note arrayimizin içerisine yeni notumuzu push edip saveNotes() yazarak json formatında kaydediyoruz.

if koşulu olarak !duplicateNote yazdık. Bu durumda not ekleyebiliyoruz veya tam tersi şekilde de if koşuluna duplicateNote yazıp else kısmında notumuzu kaydettiğimiz kodları yazabilirdik. Kodumuzu çalıştırdığımızda şu şekilde çıktı alıyoruz.

Eğer başlık aynı değilse;

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi> node app.js add --title=
"Testing" --body="test yapiliyor"
undefined
yeni not eklendi
```

Burada undefined yazmasinin sebebi şudur: addNote fonksiyonunu incelersek içerisinde şu kodun olduğunu görürüz.

```
  console.log(duplicateNote)
```

Daha önce belirttiğimiz gibi find metodu mevcut notlar arrayini tarayip eğer varsa duplicateNote içerisine bir nesne döndürüyordu. Şu an

eklediğim not bilgisinde ayni başlıkta başka bir not olmadığı için tanımsız döndürdü.

Eğer başlık aynıysa:

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi> node app.js add --title=
"Testing" --body="test yapiliyor"
{ title: 'Testing', body: 'test yapiliyor' }
Bu başlık daha once kullanıldı.Not eklenemiyor!!!
```

Ayni başlıkta bir notumuz olduğu ve bu notun içeriğinin ne olduğu bilgisini aldık.

addNote adlı fonksiyonumuzu düzenledikten sonra ayni düzenlemeyi bu sefer removeNote adlı fonksiyonumuz için yaptık.

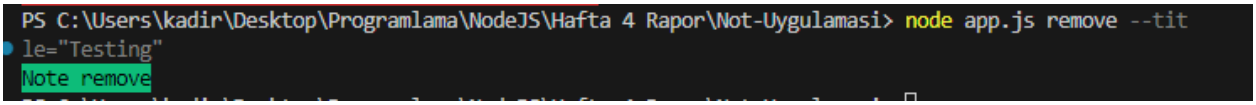
```
const removeNote = (title) =>{
  const notes = loadNotes();
  const notesToKeep = notes.filter((note) => note.title !== title);

  if (notes.length > notesToKeep.length) {
    console.log(chalk.green.inverse("Note remove"));
    saveNotes(notesToKeep);
  } else {
    console.log(chalk.red.inverse("Silmek istediiginiz not bulunamamıstır."));
  }
};
```

Burada yaptığımız şey ise addNote içerisinde yaptığımızdan farklı olarak filter metodunu kullanmaktı. Filter metodu buradaki işlevi ise şudur notes arrayinin her bir elemanını tarar ve içerisinde bulunan koşula yani eğer silinmek istenen başlık not başlığına eşit değilse başka bir array'e eleman döndürür ve bu array notesToKeep değişkeni içerisinde saklanır.

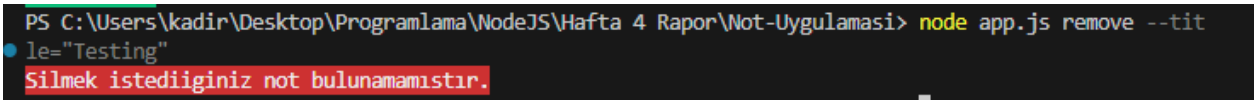
```
  if (notes.length > notesToKeep.length) {
    console.log(chalk.green.inverse("Note remove"));
    saveNotes(notesToKeep);
```

Eğer tutulan notların array boyutu Önceki notlar dizisinden küçükse notes arrayi içerisinde istenilen not başarılı bir şekilde silinmiştir demektir.

Eğer başarılıysa aldığımız çıktı:



Eğer başarısızsa aldığımız çıktı:



Bu işlemleri tamamladıktan sonra not uygulamamıza iki yeni fonksiyon daha ekledik bunlar readNotes() ve listNotes(). Öncelikli olarak bu fonksiyonların iskeletini tanımladık ardindan export ettik.

```
module.exports = {
  addNote: addNote,
  removeNote: removeNote,
  listNotes : listNotes,
  readNotes : readNotes
};
```

Ardından app.js dosyamiza gelip burada yargs modülünü kullanarak parametrelerini ayarladık. Read işlemi için sadece not başlığıyla okuma yapıldı

```
yargs.command({
  command: "read",
  describe: "secilen notu gosterir",
  builder: {
    title: {
      describe: "Not basligi",
      demandOption: true,
      type: "string",
    },
  },
  handler (argv) {
    notes.readNotes(argv.title)
  },
});
```

Notes.js içerisinde bu fonksiyonun içeriğine gelecek olursak başlık olarak aldığımız parametreyle birlikte öncelikle mevcut notları bir değişkende saklarız ve find metodu ile mevcut notlar içerisindeki başlıkları tararız eğer varsa mevcut not okunur yoksa uyarı verir.

```
const readNotes = (title) => {
  const notes = loadNotes();
  const note = notes.find((note) => note.title ===title)
  if(note){
    //note varsa
    console.log(chalk.inverse(note.title))
    console.log(note.body)
  }
  else{
    //not yoksa
    console.log(chalk.red.inverse("Bu başlığa sahip bit not bulunamadı"))
  }
}
```

Kodu çalıştırdığımızda ise eğer not varsa aldığımız çıktı:

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi> node app.js read --title
="testing"
testing
merhaba
```

Eğer not yoksa aldığımız
çıktı:

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi> node app.js read --title
="abcnot"
Bu başlığa sahip bit not bulunamadı
```

Şimdi listNotes fonksiyonumuza bakacak olursak export ettikten sonra app.js içerisinde jargs modülü sayesinde parametre girişlerini şu şekilde yaptık

```
yargs.command({
  command: "list",
  describe: "mevcut notu listeler",
  handler () {
    notes.listNotes()
  },
```

```
});
```

Parametre olarak her hangi ek almaz node app.js list yazdığımız anda notes.listNotes() fonksiyonu tetiklenir. Bu fonksiyonun içeriğinde ise şunu yaptık:

```
const listNotes = () => {
  const notes = loadNotes()
  console.log(chalk.inverse('Kayıtlı Notlar'))
  notes.forEach((note) => {
      console.log(note.title)
  });
}
```

Mevcut notları aldık bir değişkende tuttuk ve forEach fonksiyonunu kullandık bu fonksiyon içerdiği callback fonksiyonu parametresinde indexleri tutar ve sırayla tüm array elemanlarının başlık bilgileri console üzerine yazdirilir. Eğer kodu çalıştırırsak alacağımız çıktı:

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi> node app.js list
Kayıtlı Notlar
ba
bsşlsa
bsşlssfga
bsşlstgsa
baslık
kadir
kadier
skadier
skadizzer
skadizzexr
testing
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi>
```

Dersin son kısmında ise nodejs ‘de debugging nasıl yapılır kısaca buna baktık. Debugging işlemi için en kısa yol console.log fonksiyonunu kullanmaktır. Belirli noktalara yazarak test edebiliyoruz bunun dışında ise

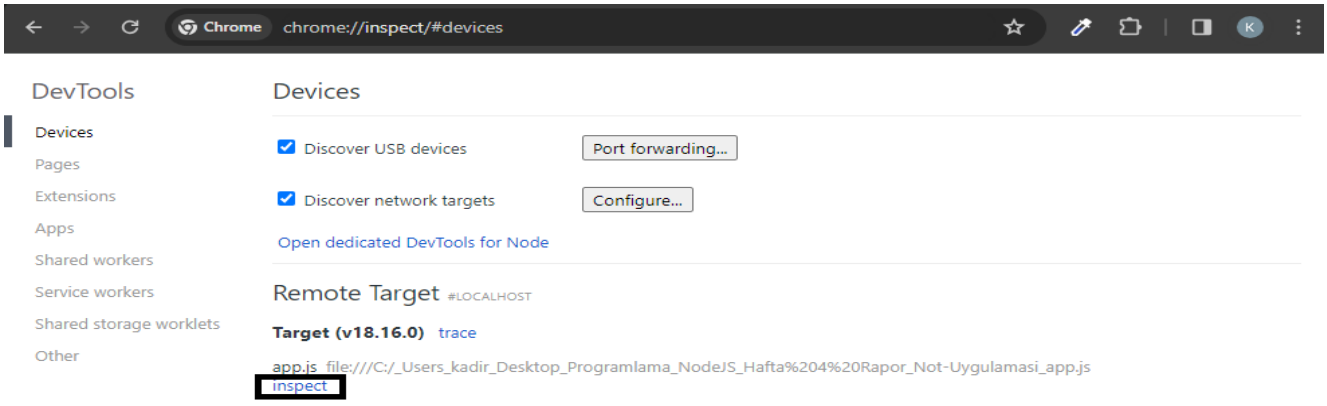
Eğer sorunun olduğunu düşündüğümüz yere bir breakpoint koyarız bu breakpoint koyma işlemi kod içerisinde doğrudan debugger şeklinde yazılarak olur. Örneğin:

```
const notes = loadNotes();
const note = notes.find((note) => note.title ===title)

debugger

if(note){
  //note varsa
  console.log(chalk.inverse(note.title))
  console.log(note.body)
}
```

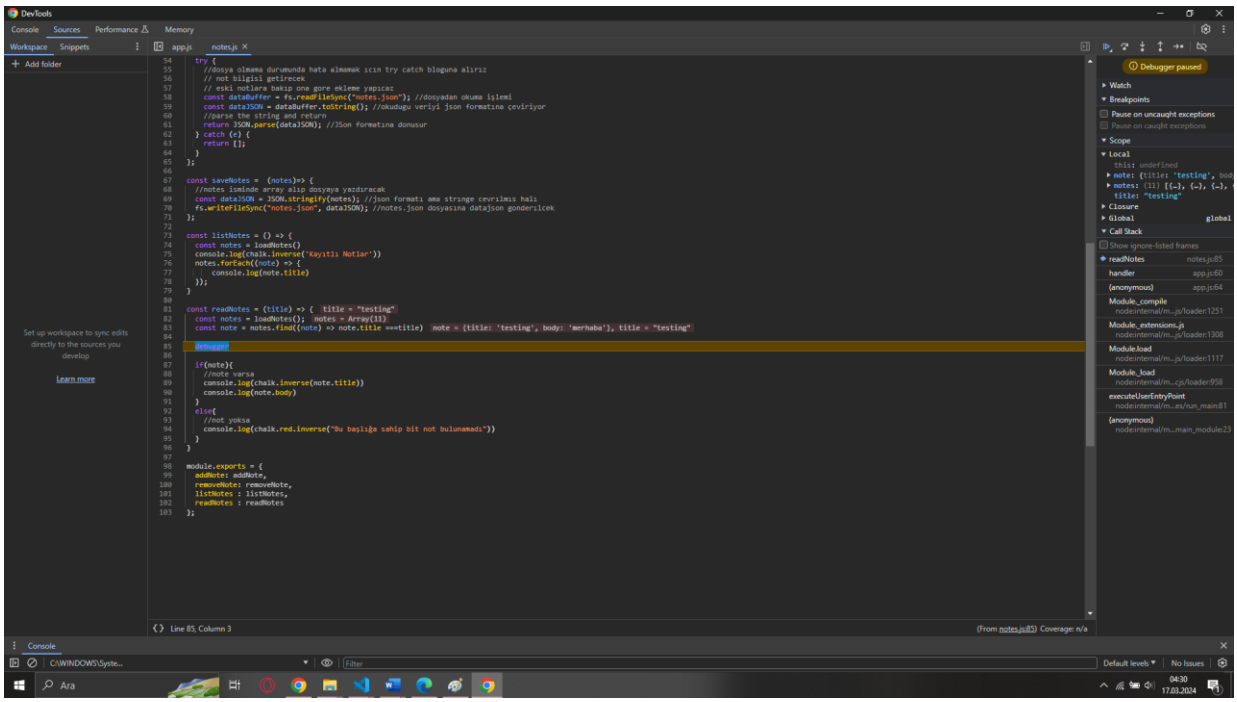
Çalıştırmak içinse şu komut kullanılır :

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Hafta 4 Rapor\Not-Uygulamasi> node inspect app.js read --title="
testing"
< Debugger listening on ws://127.0.0.1:9229/87419418-75f0-48bd-9e64-98a825d259b0
< For help, see: https://nodejs.org/en/docs/inspector
<
 ok
< Debugger attached.
<
Break on start in app.js:1
> 1 const yargs = require("yargs");
  2 const notes = require("./notes"); //nokta bulundugu konum demek oluyor
  3 //require içersinde obje geliyor bu yuzden gecer isim vermek gerekiyor
debug>
```

Ardindan Chrome üzerinden chrome://inspect/#devices adli url girip Remote Target altında bulunan inspect yazisina tıklarsak



Alacağımız ekran şu şekilde bir debug ekranı olur:



Sağ üst kısımda başlatma simgesine tıkladığımızda debug işlemini başlatmış oluruz.