Furkan Kazım Çam-21360859036

4.Hafta Rapor

Bu hafta Arrow fonksiyonlarını öğreneceğiz.

Bir app.js dosyası açınız.

```
const tasks = {
tasks: [{
text: 'Grocery shopping',
completed: true
}, {
text: 'Clean yard',
completed: false
}, {
text: 'Film course',
completed: false
}],
}
console.log(tasks.getTasksToDo())
```

Şuan da çalışmaz çünkü getTasksToDo() fonksiyonunu tanımlamadık. Tasks objemizin içine

```
getTasksToDo() {
Const tasksToDo = this.tasks.filter((task) => {
return task.completed === false
})
return tasksToDo
}
```

Bu fonksiyonu yazınız. Bu fonksiyon tasks objesinin içindeki değerleri completed özelliği false olanları filtreler ve geri döndürür. Daha da kısaltmak istersek fonksiyonu şöyle de yazabiliriz.

getTasksToDo() {

return this.tasks.filter((task) => {

return task.completed === false})

}

Tek satıra da düşürebilirsiniz.

getTasksToDo() {

return this.tasks.filter((task) => task.completed === false
}

Arrow fonksiyonları öğrendik. Şimdi ise geçen hafta yazdığımız dosyaları açıyoruz ve ordaki fonksiyonları arrow fonksiyonlara çeviriyoruz.

App.js dosyası ile düzenlemeye başlayalım.

Komutlarımızın handlerlarında ki handler: function (argv) ifadesini handler(argv) => şeklinde düzeltelim. Bunu bütün komutlarımız için yapalım.

Notes.js dosyası ile devam edelim.

getNotes fonksiyonu;

const getNotes = () => {

return ‘your notes..’

}

addNote fonksiyonu ;

const addNote = (title, body) => {

addNote fonksiyonun içindeki duplicateNotes fonksiyonu;

const duplicateNotes = notes.filter((note) => note.title === title)

removeNote fonksiyonu;

const removeNote = (title) => {

removeNote fonksiyonu içindeki notesToKeep fonksiyonu;

```
const notesToKeep = notes.filter((note) => note.title !== title)
```

saveNotes fonksiyonu;

```
const saveNotes = (notes) => {
```

loadNotes fonksiyonu;

```
const loadNotes = () => {
```

Bütün fonksiyonları arrow fonksiyona çevirdik şimdi ise kaydedip test edelim.

```
node app.js remove —title=“List”
node app.js add —title=“t” —body=“b”
```

notes.js’in içine listNotes fonksiyonu tanımlayarak uygulamaya devam edelim.

```
const listNotes = () => {
}
```

Şimdi ise bu fonksiyonu export edelim.

```
module.exports = {
..
listNotes: listNotes
}
```

app.js’e geçelim ve yeni bir komut tanımlayalım.

```
yargs.command({
command: 'list',
describe: 'List your notes',
```

```
handler: function () {
notes.listNotes()
}
} )
```

Notes.js'e dönelim ve listNotes fonksiyonunu arrow fonksiyon olacak şekilde yeniden yazalım.

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

Node app.js list komutu ile test edelim.

Son olarak ise not okuma işlemi için bir fonksiyon ve komut yazacağız. addNote fonksiyonunda ki filtreleme fonksiyonunda birkaç değişiklik yapmamız gerekmektedir.

Filtreleme işleminde aradığımız özelliği bulsa bile yine de bütün diziyi tarayacağı için çok etkili bir kod olmayacaktır. Bunun için filtreleme yaptığımız satırı değiştirerek işleme başlayalım. Sildiğimiz boşluğa ise alttaki kodu yazalım.

```
const duplicateNote = notes.find((note) => note.title === title)
```

find metodu aradığımız özelliği dizide ilk bulduğu yerden sonra kalan kısmı taramaz ve sonucu döndürür.

İf koşulunu ise if (!duplicateNote) veya if (duplicateNote === undefined) bu şekilde yazabilirsiniz. Her ikisi de aynı mantıktadır.

App.js'in içinde komudu tanımlayalım.

```
yargs.command({

    command:"read",
    describe:"Read your notes",
```

```
    builder:{
        title:{
            describe:"Note title",
            demandOption:true,
            type:"string"
        }
    },
    handler:(argv)=>{
        notes.readNote(argv.title,argv.body)
    }
})
```

Notes.js'in içinde ise readNote adlı fonksiyonumuzu tanımlayalım.

```
Const readNote = (title) => {
}
```

Export edelim.

```
Module.exports = {
…
readNote: readNote
}
```

readNote fonksiyonun içini dolduralım.

```
Const readNote = (title) => {
const notes = loadNotes()
const note = notes.find((note) => note.title === title)
if (note) {
console.log(chalk.inverse(note.title))
```

```
console.log(note.body)

} else {

console.log(chalk.red.inverse('Note not found!'))

}
}
```

Dosyadaki kaydedilen bilgileri okuması için loadNotes() fonksiyonu çağırılır. Ve girilen başlık ve dosyadaki başlıklar karşılaştırılır. Eğer ki bulunursa başlık içeriği ile birlikte ekrana yazdırılır. Bulunamaz ise kırmız şekilde ekrana yazdırılır.

Test edelim.

node app.js read —title="dd"

Debugging nasıl yapılır?

Debug için konsol kullanılabilir.

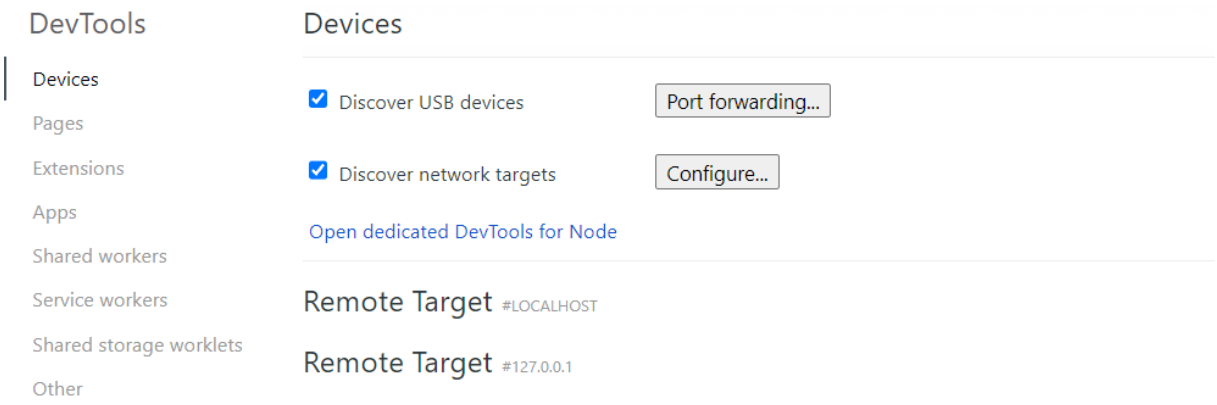
Notes.js dosyasında ki add fonksiyonun içinde olan duplicate koşulundan önce debugger yazarak debug işlemini başlatabiliriz.

Test edelim.

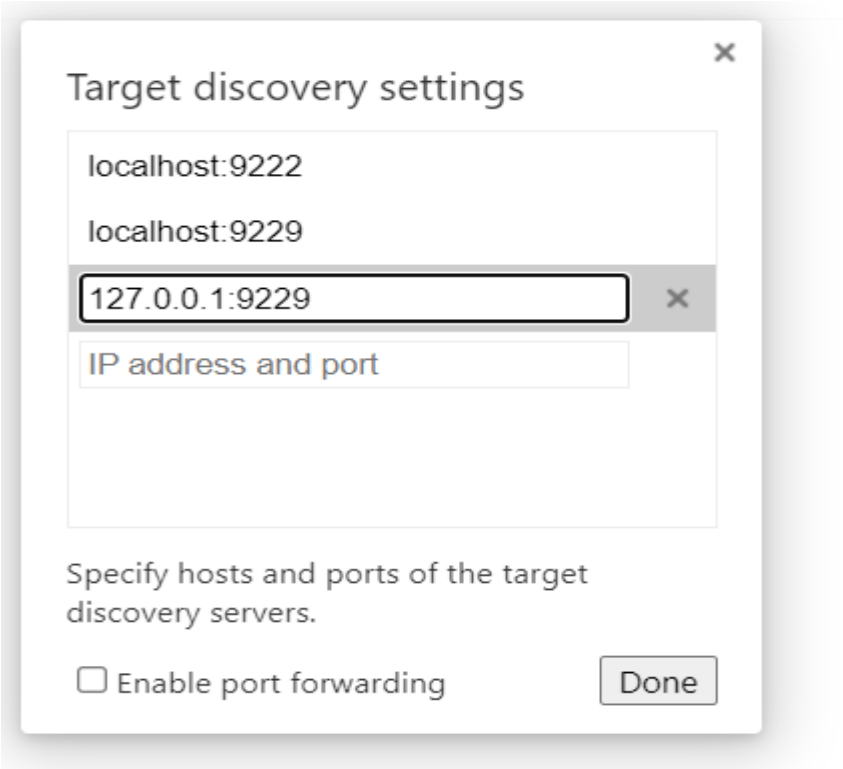
node app.js add —title="Course" —body="nodejs"

ilk çalıştırıldığında undefined sonucu alırız. Bir defa daha çalıştırıldığında ise if koşuluna girmez else'e girer.

Kullandığınız tarayıcıyı açınız. Misal brave kullanıyorsanız brave://inspect yazıp bu sayfa gidiniz.



Remote target kısmını görmüyor iseniz Configure butonuna tıklayarak



Bu ip adresini giriniz.

Açılan sayfa da ise sol taraftan kodları yazdığınız dosyayı açıp kodlarınızı görüntüleyebilirsiniz. Vs Code'a benzer bir yapısı olduğunu görmüşsünüzdür.

Sağ tarafta ise debug yapılacak olan buton vardır tek tek tıklayarak satır satır kodunuzu çalıştırabilirsiniz.