# Ember.js



# 1. Sissejuhatus ja alustus

## 1. Mis on

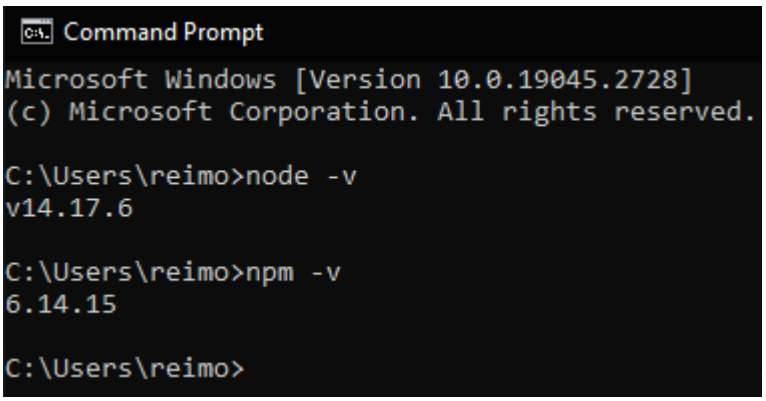
Ember.js on produktiivne, lahingu testitud JavaScripti raamistik kaasaegsete veebirakenduste loomiseks. See sisaldab kõike, mida vajate rikkalike kasutajaliideste loomiseks, mis töötavad mis tahes seadmes.

## 2. Kes kasutab

Kõigepealt võiks teada, kes kasutab antud veebiraamistikku. Ember.js-i kasutavad LinkedIn, Microsoft, Apple eriti Apple muusika, Netflix, Heroku, Square ja paljud teised.

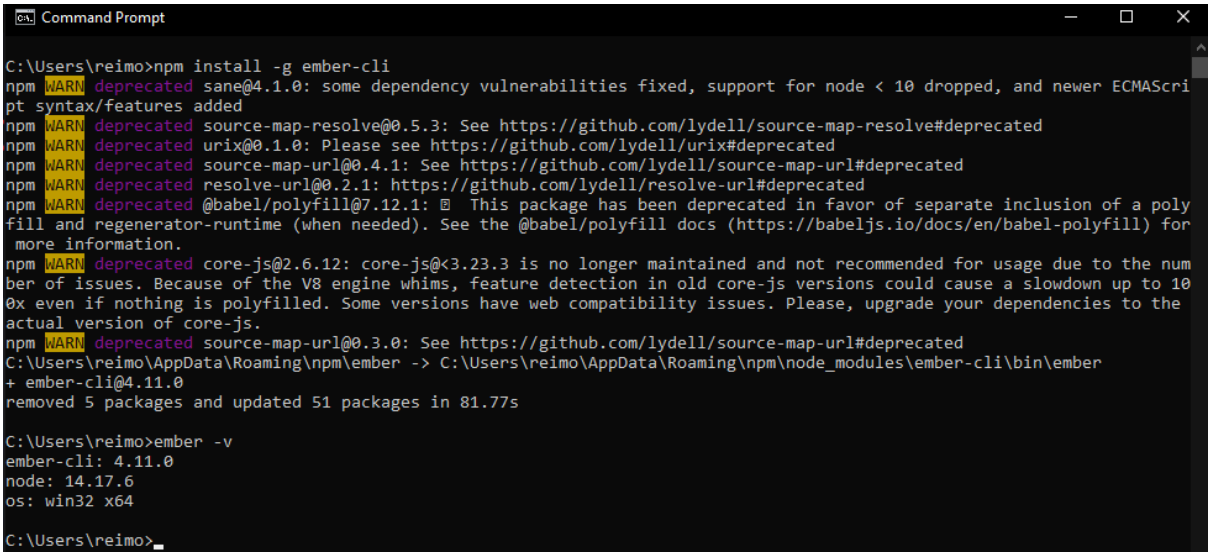
## 3. Kuidas alustada

Tehke kindlaks, et teil on installitud Node.js ja npm. Seda saab kontrollid kirjutades terminali “node -v” ja “npm -v” (Pilt 1 Kontroll) ja  Kui ei ole siis installige lehelt: https://nodejs.org/en/download



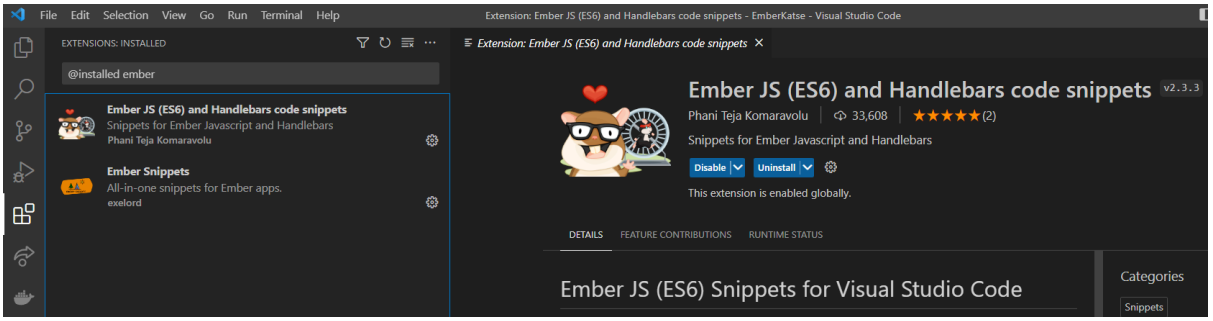
Pilt1 Kontroll

Seejärel installeerime veebiraamistiku ember.js-i globaalselt, et saaksime seda kasutad ükskõik kus oma seadmes kasutades terminalis käsklust "npm install -g ember-cli" ning kontrollime kas antud raamistik sai installeeritud käsuga "ember -v"(Pilt 2 Installeerimine)



Pilt 2 Installeerimine

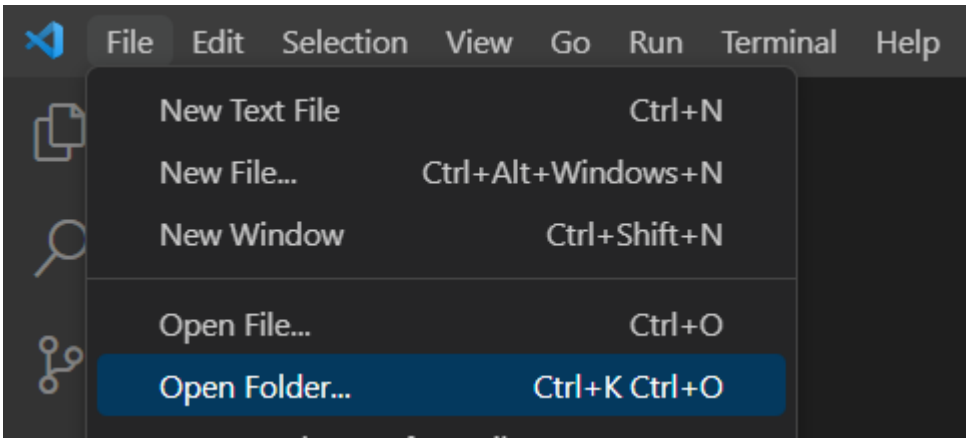
Valikuline: Installeerida Visual Studio Code kui seda veel ei ole https://code.visualstudio.com/download ja mõningad kasulikud pluginad(Pilt 3 Pluginad)



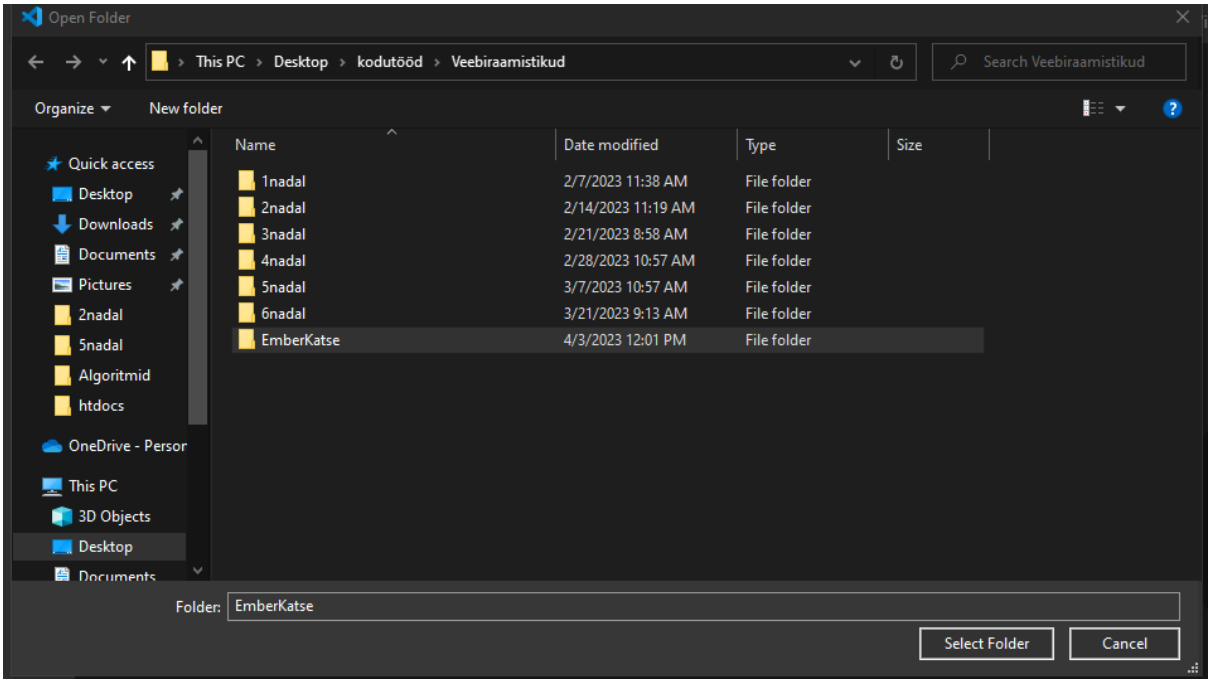
Pilt 3 Pluginad

Kui kõik vajalikud on installeeritud siis saame alustada projektiga. Tehke projekti jaoks uus folder ja avage see VSC-is (Pilt4 ava folder) Ma ise topin selle enda veebiraamistikke kausta,

mis ise asub töölaual olevas kodutööde kaustas (Pildid 4 ja 5 Ava tulevane projekti folder)




Pilt4 Ava tulevane projekti folder




Pilt 5 Ava tulevande projeti folder

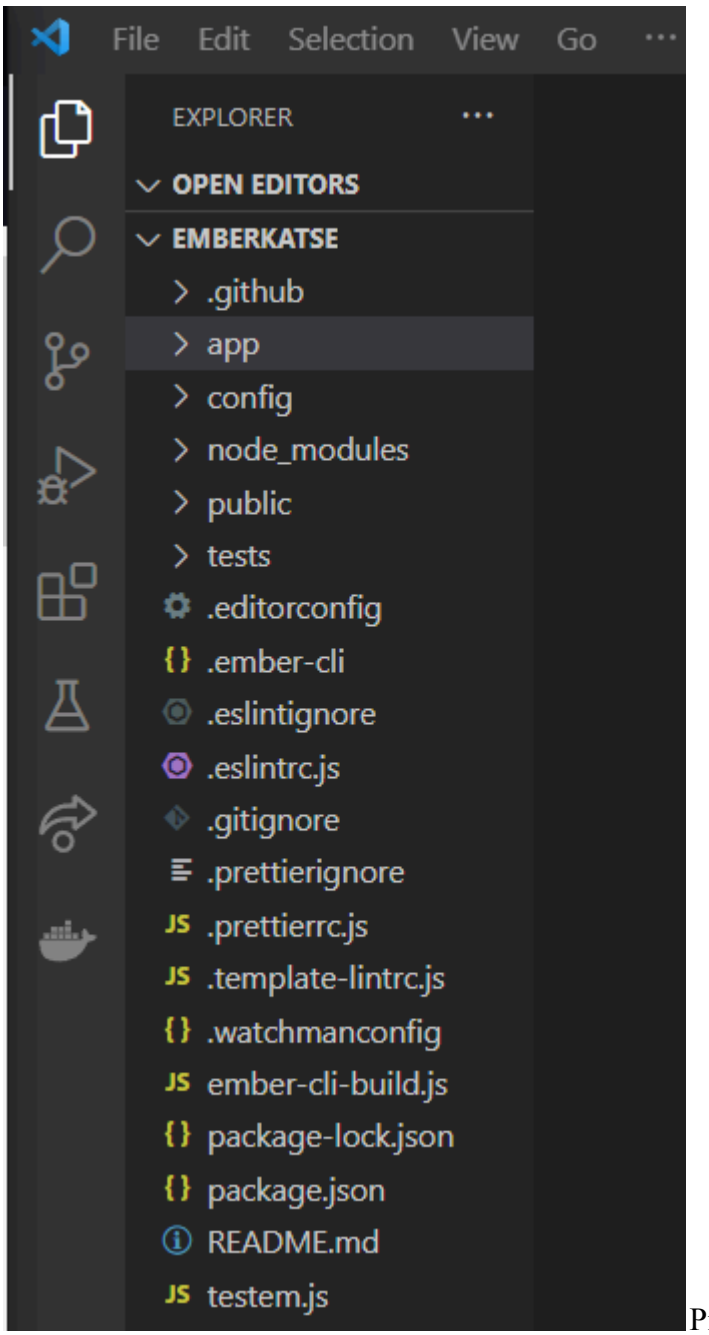
Põhjus miks ma eelistan kasutada VSC-i on see, et tema sisse ehitatud terminal valib koheselt avatud kausta/faili asukoha. Ehk kui folder on VSC-is avatud siis kirjutage VSC-i terminali "ember init". (Pilt 6 CMD Ember Init) See algatab projekti

```
PS C:\Users\reimo\OneDrive\Töölaud\kodutööd\Veebiraamistikud\EmberKatse> ember init
installing app
Ember CLI v4.11.0

Creating a new Ember app in C:\Users\reimo\OneDrive\Töölaud\kodutööd\Veebiraamistikud\EmberKat
se:
  create .editorconfig
  create .ember-cli
  create .eslintignore
  create .eslintrc.js
  create .github\workflows\ci.yml
  create .prettierignore
  create .prettierrc.js
  create .template-lintrc.js
  create .watchmanconfig
  create README.md
  create app\app.js
  create app\components\.gitkeep
  create app\controllers\.gitkeep
  create app\helpers\.gitkeep
  create app\index.html
  create app\models\.gitkeep
  create app\router.js
  create app\routes\.gitkeep
  create app\styles\app.css
  create app\templates\application.hbs
  create config\ember-cli-update.json
  create config\environment.js
  create config\optional-features.json
  create public\robots.txt
  create testem.js
  create tests\helpers\index.js
  create tests\index.html
  create tests\integration\.gitkeep
  create tests\test-helper.js
  create tests\unit\.gitkeep

Installing packages... This might take a couple of minutes.
- npm: Installing dependencies ...
npm: Installed dependencies

Successfully created project EmberKatse.
Get started by typing:

  $ cd EmberKatse
  $ npm start
```

```
 Happy coding!
 PS C:\Users\reimo\OneDrive\Töölaud\kodutööd\Veebiraamistikud\EmberKatse>
```

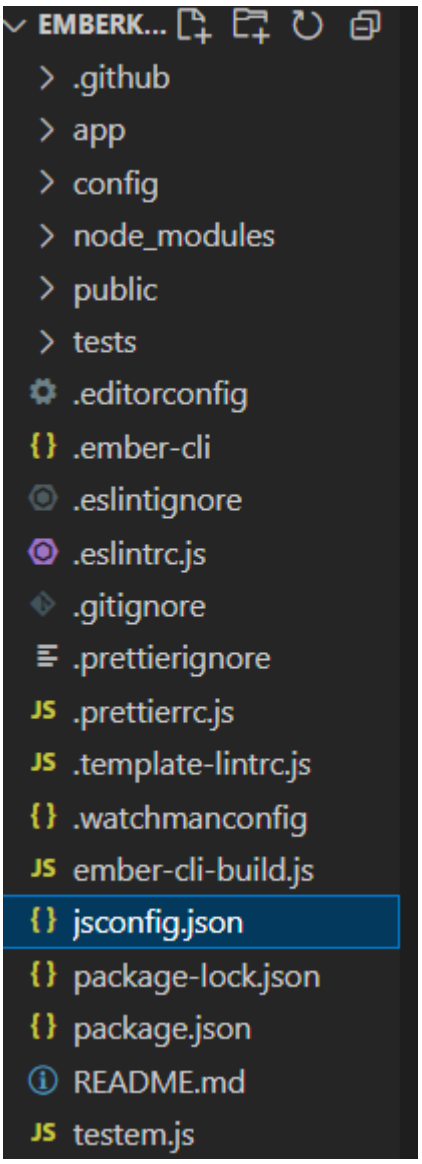
Pilt 6 CMD Ember Init

Nüüd peaks olema Emberi vajalikud failid (Pilt 7 Emberi failid)

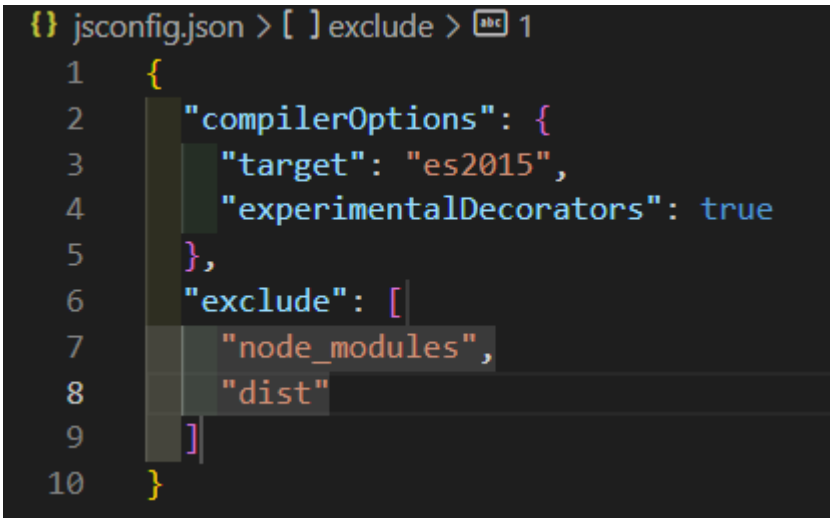


Pilt 7 Emberi failid

Loome uue faili nimega "jsconfig.json". See näitab, et praegune tööruum on JavaScripti projekt millel on keele serveri poolt toetatud eelised (mis need eelised on ei oska öelda). Esmalt peame seadistama oma kompilaatori valikud ja seejärel seadke eesmärgid es 2015-le, mis on ts6 ja siis on meil vaja, et seada meie katsed või kaunistused tõeseks ja siis välistada nodemoodul ja avalikustamine. Ehk "jsconfig.json" peaks välja nägema selline.(Pilt 8 uus fail)(Pilt 9 uue faili sisu)

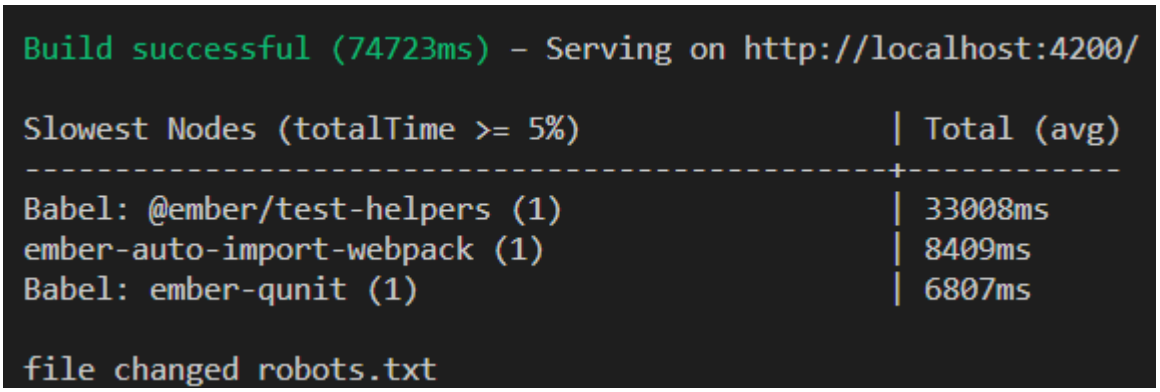



Pilt 8 uus fail



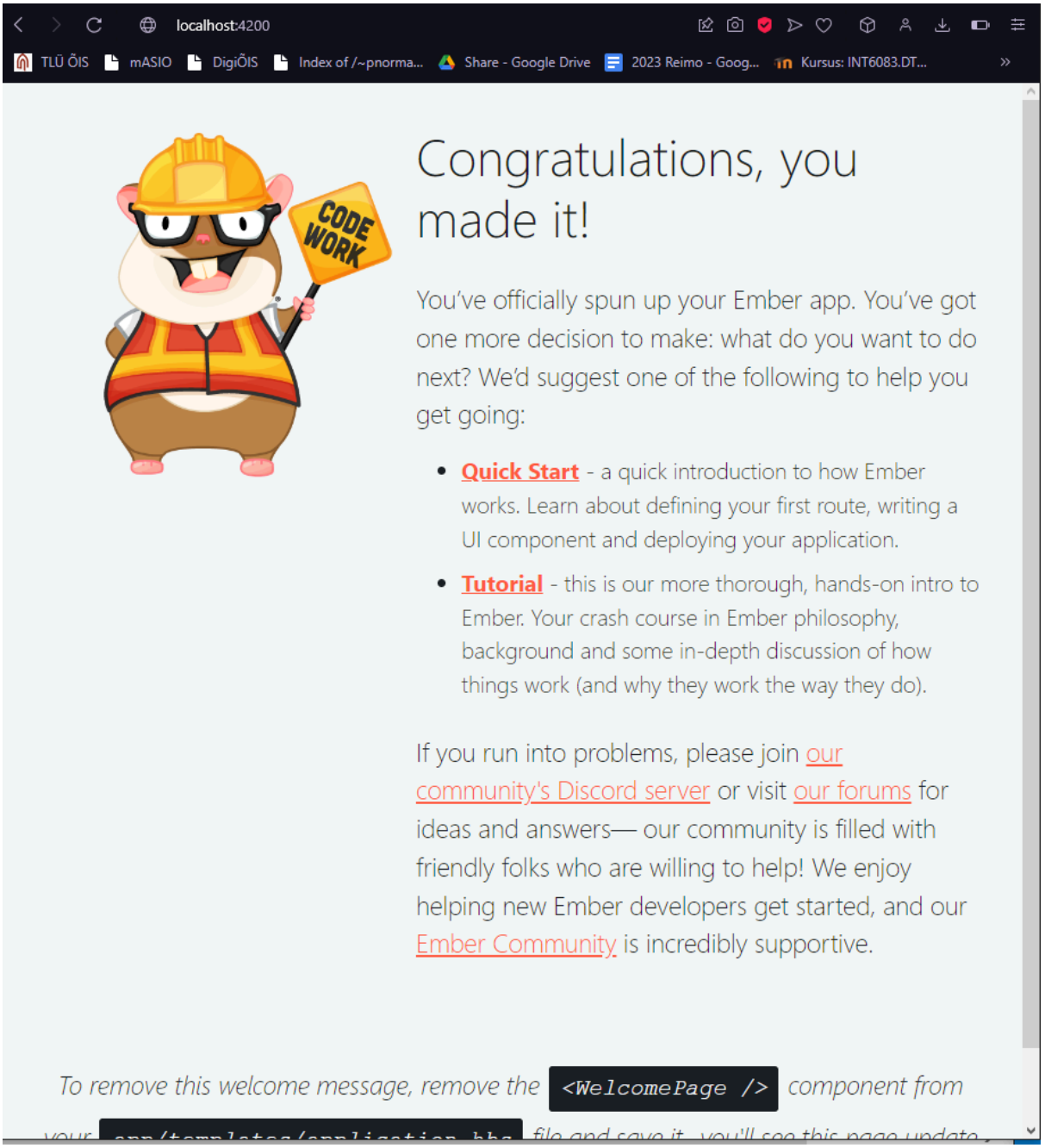

Pilt 9 uue faili sisu

Kui uus fail sai loodud ja vajalik sisu lisatud siis saame emberi severi tööle panna kirjutades terminali käsu "ember s"(Pilt 10 Serveri start).



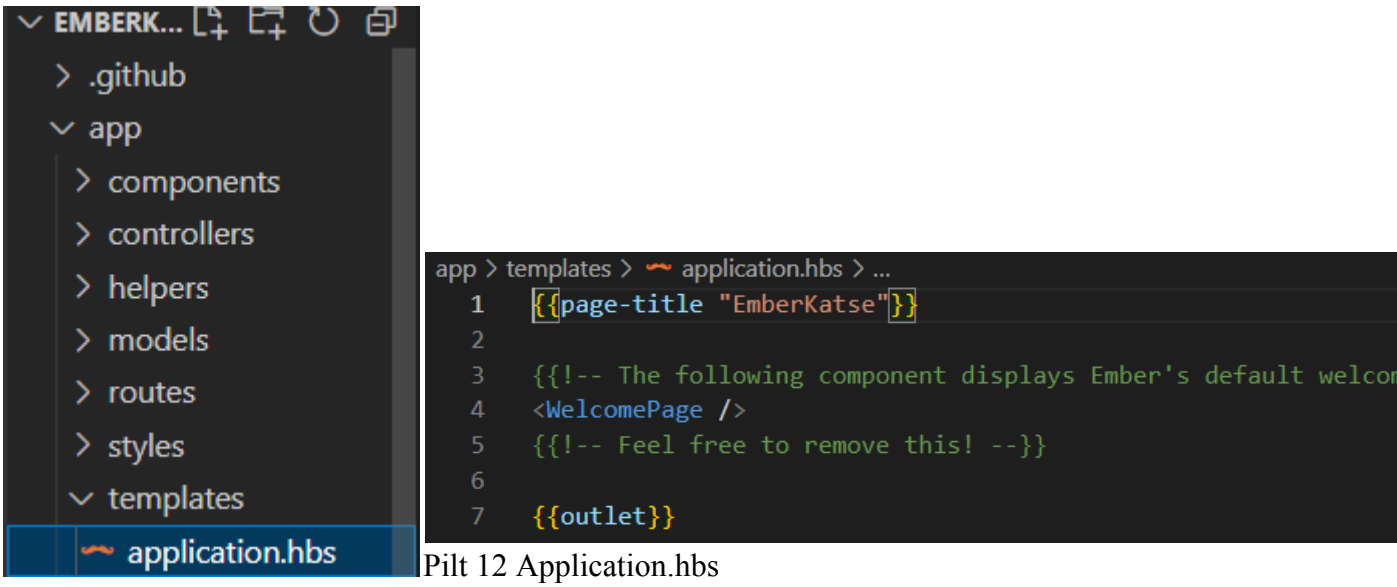

Pilt 10 Serveri start

Seejärel saame vaadata algelist lehte ja kontrollida kas server töötab aadressil http://localhost:4200/ (Pilt 11 Algne leht)



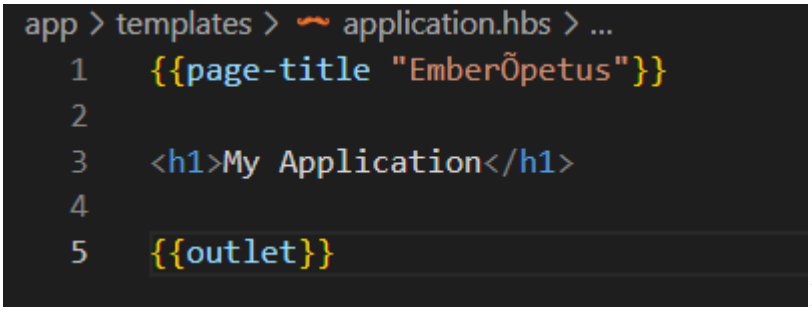

Pilt 11 Algne leht

Vahetame algse lehe enda uue lehe vastu. Selleks avage fail järgmiselt valige folder "app" sealt valige folder "templates" ning seal avage fail "application.hbs"(Pilt 12 Application.hbs)



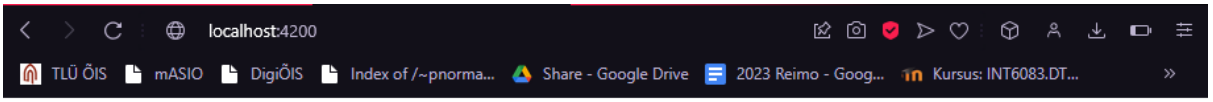
Pilt 12 Application.hbs

HBS fail on umbes 90% HTML mõne lisa funktsiooniga. Aga selleks, et saada oma leht eemaldama elemendi “<WelcomePage />” Ja lisame kõigepealt testimiseks lihtsa pealkirja Minu applikatsioon. Kood peaks olema siis selline: Kuid loomulikult võib olla lehe pealkiri kui ka lehel olev pealkiri olla erinev. (Pilt 13 Esimene pealkiri)



Pilt 13 Esimene pealkiri

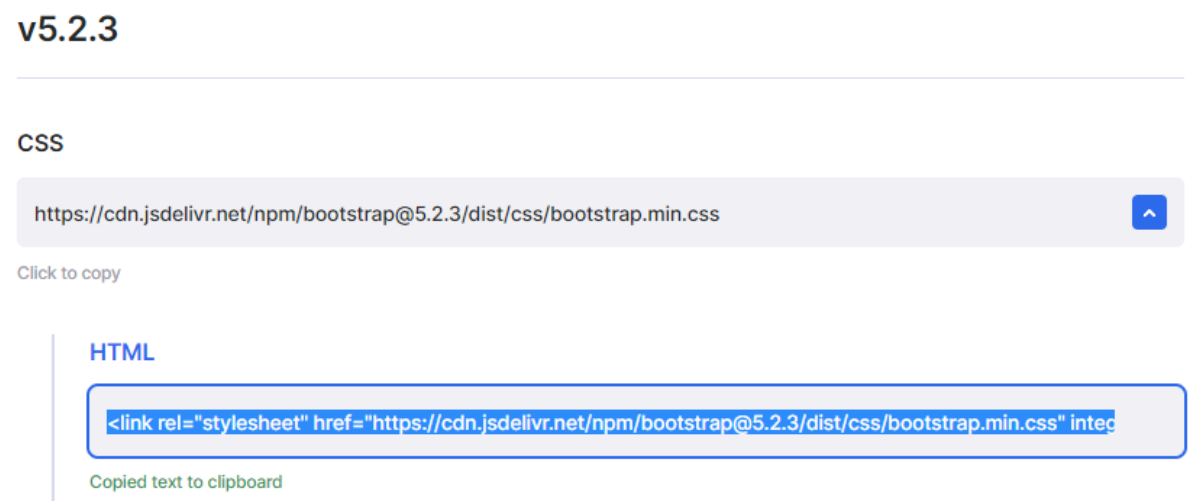
Kui failis on muudatused tehtud peaks peale faili salvestust server märkama muudatust ja seda ka localhostis pea koheselt näitama. Leht ise on nüüd selline(Pilt 13 Enda leht)




**My Application**

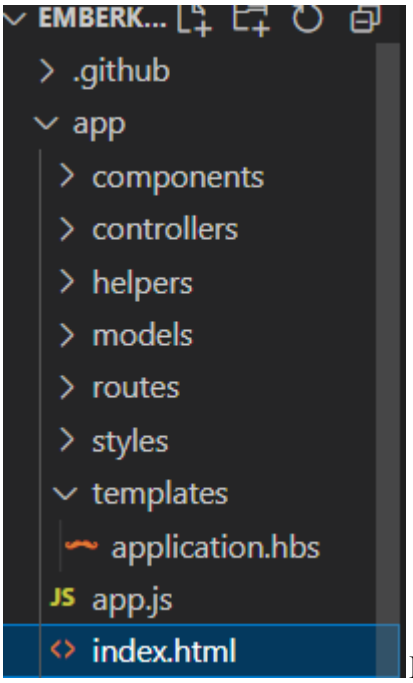
Pilt 13 Enda leht

Teeme oma lehe ka natuke ilusamaks. Selleks minge lehele “https://www.bootstrapcdn.com” ja kopeerige css-i html link(Pilt 14 Bootstapcdn css)



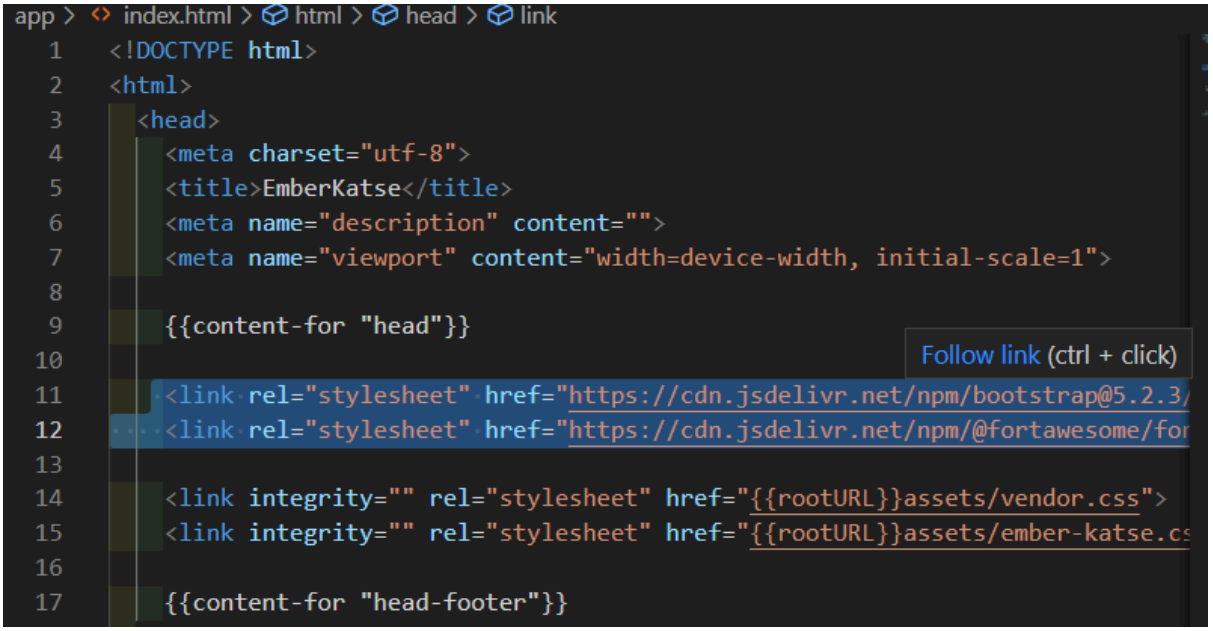
Pilt 14 Bootstrapcdn css

Kopeerige see rida koodi faili Index.html(Pilt 15 Index.html asukoht)



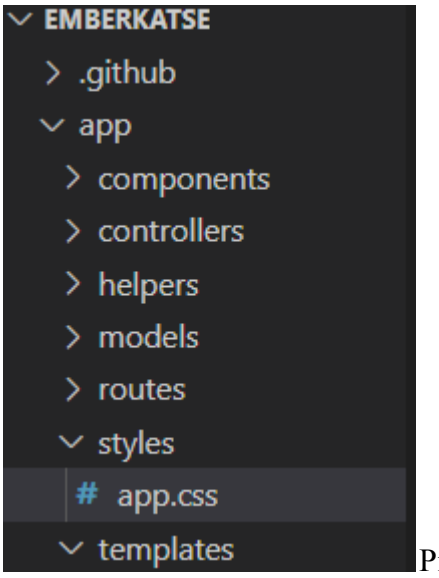
Pilt15 Index.html asukoht

Peale lingi lisamist minge lehele https://www.bootstrapcdn.com/fontawesome/ ja kopeerige index.html faili samamoodi ka font awesome css html link. ehk lõpuks peaks index.html välja nägema selline.(Pilt 16 Lisatud stiilid)



Pilt 16 Lisatud stiilid

Aga proovime ka ise manuaalselt Lehe stiili ehk css-i muuta. Selleks avage folder “app” sealt folder “styles” ning sealt avage fail “app.css”(Pilt 17 app.css asukoht)



Pilt17 app.css asukoht

Testimiseks kas css faili loeb ja teeb muudatusi võite ise muidugi css-iga mängida mina ise toppin siia enda lisatud css elemendid

```
body {
  color: white;
  display: flex;
  justify-content: center;
  background-color: red;
```

Kui kasutate sama css-i, mis mina siis peaks leht välja nägema järgmiselt(Pilt 18 Hex css)
Pilt 18 Hex css

Kui soovite kasutada css faili asemel scss-i siis muutke faili nimi app.css ist app.scss-iks ning kirjutage käsureale ”ember install ember-cli-sass (Pilt 19 scss install)

```
 install ember-cli-sass

Installing packages... This might take a couple of minutes.
npm: Installed ember-cli-sass
installing ember-cli-sass
  install package sass

Installing packages... This might take a couple of minutes.
npm: Installed sass
Installed addon package.
```

Pilt 19 scss install

Seejärel avage fail “ember-cli-build.js” ning lisame sinna valiku kasutada scss-i lisades paar koodi rida. Ehk fail võiks välja näha midagi sellist (Pilt 20 ember-cli-build muudatus)

```
JS ember-cli-build.js > <unknown> > exports > app
1   'use strict';
2
3   const EmberApp = require('ember-cli/lib/broccoli/ember-app');
4
5   module.exports = function (defaults) {
6     const app = new EmberApp(defaults, {
7       // Add options here
8
9       sassOptions:{
10        extension:"scss",
11      }
12
13    });
14
15    // Use `app.import` to add additional libraries to the generated
```

Pilt 20 ember-cli-build muudatus

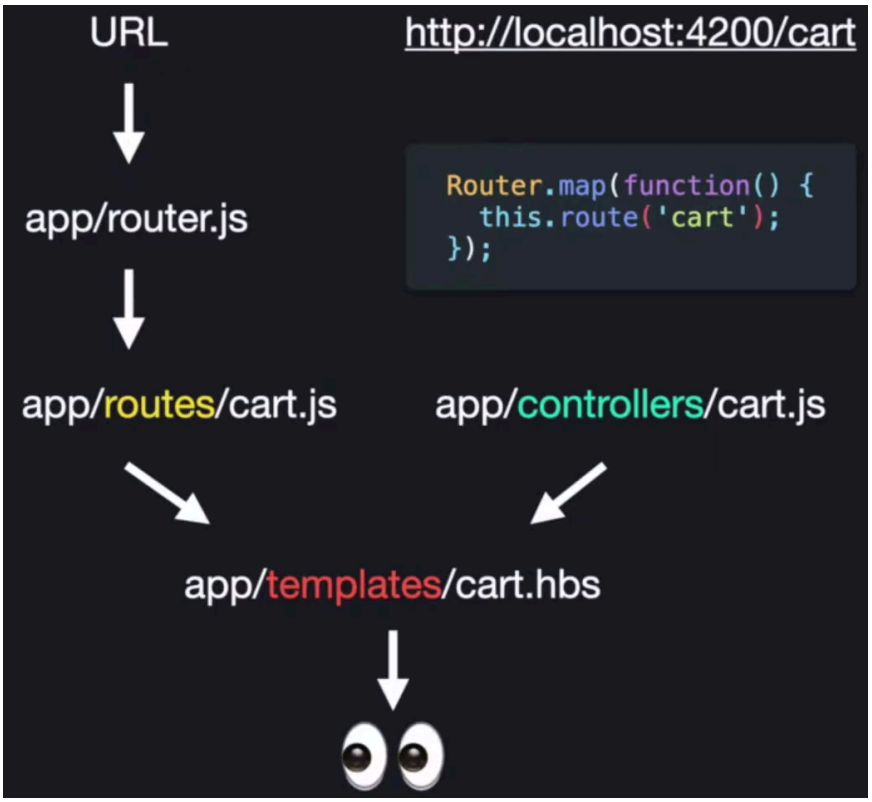
Seejärel tehke serverile restart kirjutades käsureale uuesti käsklus “ember s”. Kui saate veateate “Port 4200 is already in use” siis pange korraks VSC kinni ja avage uuesti ning kirjutage veelkord terminali ”ember s”

# 2. Routing süsteemid

Ehk siis lehekülje vahelehed v teised lehed mis iganes nende nimed on ma ei tea 😀

## 4. Kirjeldus

Ütleme, et me soovime minna leheküljele “http://localhost:4200/cart” kõigepealt otsib Ember cart teekonda failist “app/router.js” See fail on siis kui lehekülgede kaart mida arendaja saab vaadata, et aru saada kuidas antud aplikatsioon on ehitatud. Kui teekond on leitud siis otsime seotud javascripti faili mis on failis app/routes/cart.js ning failis app/controllers/cart.js. Lehe loogika on nendes kahes failis hakkame renderdama template-i mille me defineerime ning siis me saame näha muudatusi.(Pilt 21 Kirjeldus)




Pilt 21 Kirjeldus

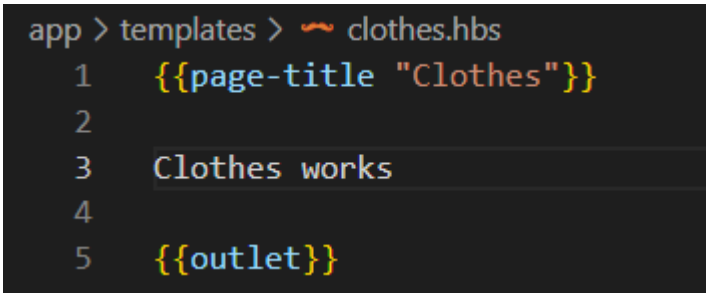
## 5. Index route

Et edasi silmadel oleks lihtsam jälgida veebilehel olevat muudan scss faili tühjaks. Ning loome siis uue lehe näiteks riiete esemete jaoks. Selleks kirjutame terminali “ember g route clothes”.(Pilt 22 uus leht) See loob antud lehe jaoks kõik vajalikud failid.

```
PS C:\Users\reimo\OneDrive\Töölaud\kodutööd\Veebiraamistikud\EmberKatse> ember
 g route clothes
installing route
  create app\routes\clothes.js
  create app\templates\clothes.hbs
updating router
  add route clothes
installing route-test
  create tests\unit\routes\clothes-test.js

Running "lint:fix" script...
PS C:\Users\reimo\OneDrive\Töölaud\kodutööd\Veebiraamistikud\EmberKatse>
```

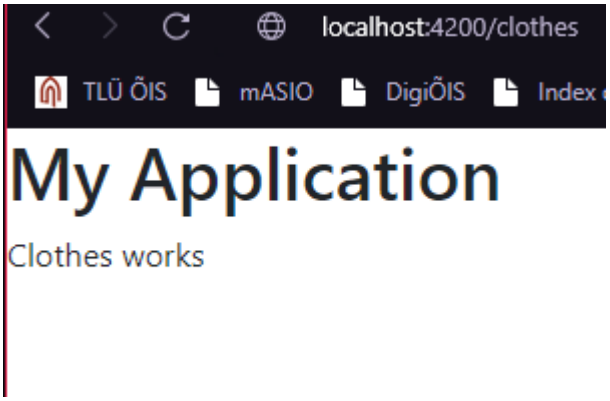
Pilt 22 Uus leht

Kui avame faili “router.js” siis näeme, et sinna on lisatud uus tee lehele clothes. Kuid selleks, et saaksime olla kindlad, et uus leht töötab lisame sinna teksti ja vaatame kas me näeme seda uuel lehel. Selleks avage templatide folderi alt uus fail nimega clothes.hbs ning kirjutage sinna midagi. Näiteks “Clothes works” või midagi muud (Pilt 23 Clothes.hbs)



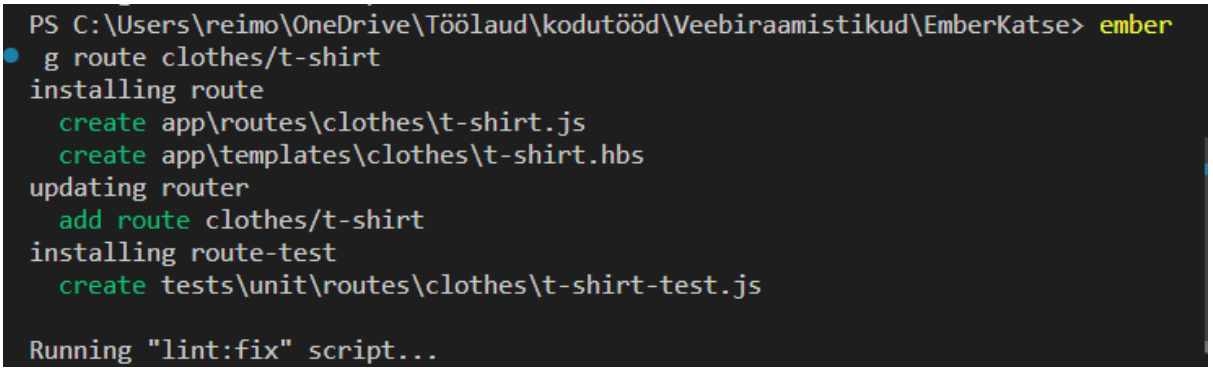
Pilt 23 Clothes.hbs

Kui muudatus on salvestatud vaatame kas seda on ka näha. Selleks avage leht http://localhost:4200/clothes. Kui lehel on näha teksti mis lisasite siis on kõik kena ja uus leht funkab (Pilt 24 Lisatud tekst lehel)



Pilt 24 Lisatud teksti lehel

Lisame lehele clothes järgneva lehe näiteks t-särkidele selleks kasutame jällegi terminali kirjutades sinna “ember g route clothes/t-shirt” (Pilt 25 Terminal T-särk)



Pilt 25 Terminal T-särk

Fail router.js peaks siis nüüd välja nägema umbes selline(Pilt 26 Router.js)

app > JS router.js > ...

```
1   import EmberRouter from '@ember/routing/router';
2   import config from 'ember-katse/config/environment';
3
4   export default class Router extends EmberRouter {
5     location = config.locationType;
6     rootURL = config.rootURL;
7   }
8
9   Router.map(function () {
10    this.route('clothes', function () {
11      this.route('t-shirt');
12    });
13  });
14
```

Pilt 26 Router.js

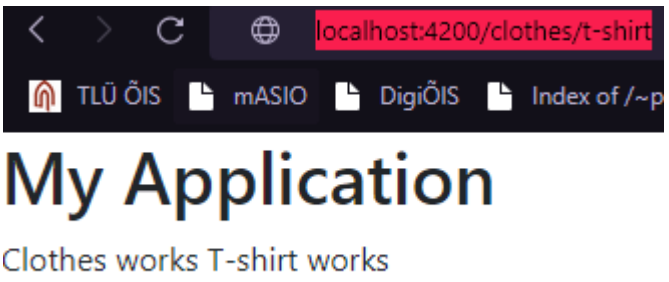
Jällegi vaatame kas veebileht töötab lisades faili “t-shirt.hbs” näiteks teksti “T-shirt works” (Pilt 27 t-shirt.hbs)

app > templates > clothes > t-shirt.hbs

```
1   {{page-title "TShirt"}}
2
3   T-shirt works
4
5   {{outlet}}
```

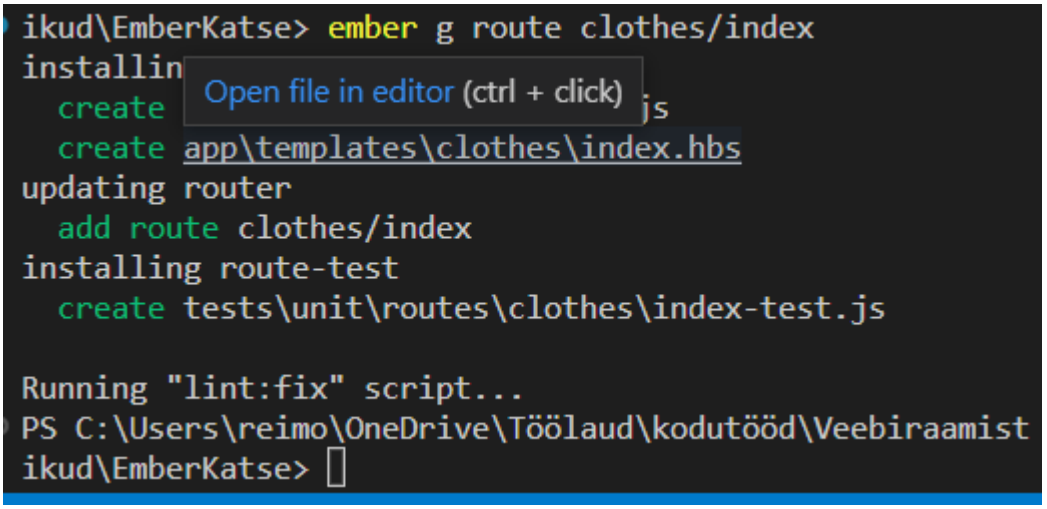
Pilt 27 t-shirt.hbs

Kui fail on salvestatud peaks http://localhost:4200/clothes/t-shirt leht välja nägema umbes selline (Pilt 28 t-shirt leht)




Pilt 28 t-shirt leht

Teema nii, et veebilehel oleks vaid ühe lehe asjad. Selleks kirjutage terminali “ember g route clothes/index” (Pilt 29 eraldi leht)




Pilt 29 eraldi leht

Avame uue faili “index.hbs ning lisame sinna teksti (Pilt 30 index.hbs)


Pilt 30 index.hbs

## 6. Dynamic route

Kui soovime mingit dünaamilist routingut ehk siis näiteks mingile esemele soovime dünaamilist ID-d, siis selleks kirjutame terminaly “”ember g route item”  ning siis kirjutame üle

router.js-is routi (Pilt 31 Dynamic route)

app > JS router.js > ...

```
1   import EmberRouter from '@ember/routing/router';
2   import config from 'ember-katse/config/environment';
3
4   export default class Router extends EmberRouter {
5     location = config.locationType;
6     rootURL = config.rootURL;
7   }
8
9   Router.map(function () {
10    this.route('clothes', function () {
11      this.route('t-shirt');
12    });
13    this.route('item', {path: '/item/:item_id'});
14  });
15
```

Pilt 31 Dynamic route

Enne uuele veebilehele minemist lisame sinna mingi teksti, et sinna minnes näha oleks, et seal muudatused toimivad. Selleks avame faili “item.hbs” ning lisame sinna näiteks “Item works” (Pilt 32 item.hbs)

app > templates > item.hbs

```
1   {{page-title "Item"}}
2
3   Item works
4
5   {{outlet}}
```

Pilt 32 item.hbs

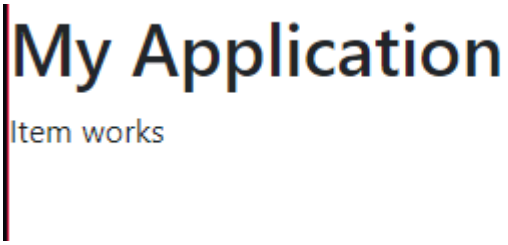
Lehel “http://localhost:4200/item/1” ei ole veel miskit näha. Emberis peame dünaamilise marsruudi jaoks määrama mudeli erineva ID alusel. Selleks lähme faili “items.js” mis ise asub “routes” folderi sees. Kui fail on lahti lisame mingi mudeli ja anname talle ID ja tagastame selle.(Pilt 33 Mudeli ID) Salvestame faili

app > routes > item.js > ItemRoute > model

```
1   import Route from '@ember/routing/route';
2
3   export default class ItemRoute extends Route {
4     model(params){
5       const {
6         item_id
7       } = params;
8       return item_id;
9     }
10  }
11
```

Pilt 33 Mudeli ID

Vaatame uuesti lehte “http://localhost:4200/item/1” ja vaatame kas lehel on miskit näha.(Pilt 34 Item 1 leht)




Pilt 34 Item 1 leht

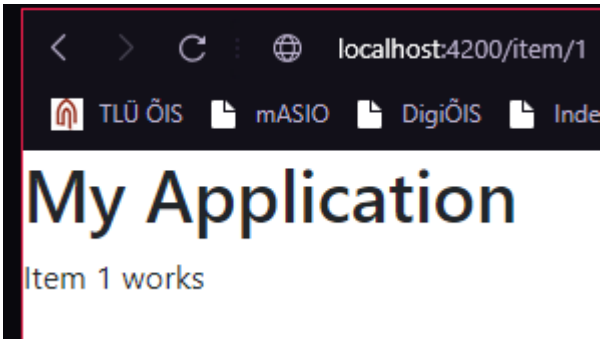
Teeme nii, et me näeme mis mudel hetkel töötab. Selleks avame uuesti faili “item.hbs” ning lisame sinna mudeli (Pilt 35 Visuaalse mudeli lisamine)

app > templates > item.hbs

```
1   {{page-title "Item"}}
2
3   Item {{this.model}} works
4
5   {{outlet}}
```

Pilt 35 Visuaalse mudeli lisamine

Ja nüüd peaks lehel olema ka nähtav mudel (Pilt 36 Nähtav mudel)




Pilt 37 Nähtav mudel

Ehk nüüd kui muudate URL-is nr 1 millegi muuga peaks see olema ka lehel nähtav.

# 7. Wildcard route ehk 404 route

Hetkel kui sisestame URL-i mingi lehe mida ei eksisteeri siis me saame lihtsalt valge lehe. Oleks tore kui veebilehe külastaja saaks ikka teada, et antud URL-i ei eksisteeri. Selleks kirjutame terminali “ember g route not-found” (Pilt 38 Terminal route not-found)

```
PS C:\Users\reimo ember g route not-found
installing route
  create app\routes\not-found.js
  create app\templates\not-found.hbs
updating router
  add route not-found
installing route-test
  create tests\unit\routes\not-found-test.js

Running "lint:fix" script...
PS C:\Users\reimo\OneDrive\Töölaud\kodutööd\Veebiraamistikud\EmberKatse>
```

Pilt 38 Terminal route not-found

Siis jällegi avame faili “router.js” ning lisame path-iks URLi mida ei eksisteeri (Pilt 39 Olematu path)

app > router.js > Router.map() callback

```
1  import EmberRouter from '@ember/routing/router';
2  import config from 'ember-katse/config/environment';
3
4  export default class Router extends EmberRouter {
5    location = config.locationType;
6    rootURL = config.rootURL;
7  }
8
9  Router.map(function () {
10   this.route('clothes', function () {
11     this.route('t-shirt');
12   });
13   this.route('item', { path: '/item/:item_id' });
14   this.route('not-found', {path: '/*path'});
15 });
16
```

Pilt 39 Olematu path

Nüüd kui läheme URL-ile mida me ei kasuta siis näeme, et meie aplikatsioon/veebileht töötab kuid lisame ka teksti mis annab teada, et antud lehte ei eksisteeri. Selleks avame faili “not-found.hbs” ning lisame sinna teatava teksti. (Pilt 40 Not found)

app > templates > not-found.hbs

```
1   {{page-title "NotFound"}}
2
3   404 not found
4
5   {{outlet}}
```

Pilt 40 Not found

Kui on soov veel teada saada siis vaadake video-si mille põhjal on õpetus tehtud “https://www.youtube.com/watch?v=eQUvN9Ujs1s&t=2s” Shawn C originaalse video materjali autor

# 3. Harjutus. Teadlaste kuvamine

https://guides.emberjs.com/release/getting-started/quick-start/

Building for production osa ei ole vaja teha!!

Teeme selleks eraldi folderi ja uue projekti. Ehk “ember init” ja siis “ember s”. Alustuseks võime “application.hbs” faili algse lehe ära kustutada.

app > templates > application.hbs > ...

```
1   {{page-title "EmberTeadlased"}}
2
3   <h1>PeopleTracker</h1>
4
5   {{outlet}}
```

{{outlet}} Näitab routi nõ child route siis

Loome rakenduse, mis näitab teadlaste nimekirja. Selleks esimene samm on marsruudi loomine. Terminali käsklus “ember generate route scientists”. Ava äsja loodud mall rakenduse kaustas app/templates/scientists.hbs ning lisa järgmine HTML-kood:

app > templates > scientists.hbs > h2

```
1   {{page-title "Scientists"}}
2   <h2>List of Scientists</h2>
```

Nüüd, kui teadlaste renderdamine töötab, andkem sellele veidi andmeid renderdamiseks. Seda teeme, määrates marsruudile mudeli, mida saame teha, muutes app/routes/scientists.js faili.

app > routes > JS scientists.js > ScientistsRoute

```
1  import Route from '@ember/routing/route';
2
3  export default class ScientistsRoute extends Route {
4    model() {
5      return ['Marie Curie', 'Mae Jemison', 'Albert Hofmann'];
6    }
7  }
```

Nüüd räägime Emberile, kuidas seda stringide massiivi HTML-ks muuta. Ava teadlaste template ja lisa järgmine kood, et massiivi läbi loopida ja selle sisu printida:

app > templates > scientists.hbs > ul

```
1  {{page-title "Scientists"}}
2  <h2>List of Scientists</h2>
3
4  <ul>
5    {{#each @model as |scientist|}}
6      <li>{{scientist}}</li>
7    {{/each}}
8  </ul>
```

Loome komponendi <PeopleList>, mida saame kasutada mitmes kohas inimeste nimekirja kuvamiseks.

Nagu tavaliselt, on meil generaator, mis teeb selle meie jaoks lihtsaks. Uue komponendi saamiseks kirjuta järgmine käsklus:"ember generate component people-list”

Kopeeri ja kleebi teadlaste template <PeopleList> komponendi templati ning muuda seda järgnevalt:

app > components > people-list.hbs > ul

```
1  <h2>{{@title}}</h2>
2
3  <ul>
4    {{#each @people as |person|}}
5      <li>{{person}}</li>
6    {{/each}}
7  </ul>
```

Oleme muutnud pealkirja hard-coded stringist ("List of Scientists") {{@title}}-ks. @ tähistab, et @title on argument, mis antakse komponendile edasi, mis muudab selle sama komponendi korduvkasutamise lihtsamaks rakenduse teistes osades

Salvesta template ja ava uuesti “scientists.hbs” template. Me ütleme oma komponendile Millist pealkirja kasutada, kasutades @title argumendiks olevat väärtust.

Millist inimeste massiivi kasutada, kasutades @people argumendiks olevat väärtust. Selle marsruudi @model esindab inimeste massiivi

# PeopleTracker

## List of Scientists

- Marie Curie
- Mae Jemison
- Albert Hofmann

Tagasi oma veebibrauserisse minnes peaksid nägema, et kasutajaliides näeb välja identne. Ainus erinevus on see, et oleme nüüd teinud oma nimekirja komponendina, mis on taaskasutatavam ja kergemini hooldatav

Siiamaani loetleb meie rakendus andmeid, kuid kasutajal ei ole võimalik nendega suhelda. Veebirakendustes soovime sageli reageerida kasutaja tegevustele nagu klikid või hiirekursori positsioon.

ember generate component people-list

Esiteks saame muuta <PeopleList> komponenti, et see sisaldaks nuppu:

app > components > people-list.hbs > ul

```
1  <h2>{{@title}}</h2>
2
3  <ul>
4    {{#each @people as |person|}}
5      <li>
6        <button type="button">{{person}}</button>
7      </li>
8    {{/each}}
9  </ul>
```

Seni on meie <PeopleList> komponent puhtalt esitluslik - see võtab mõned sisendid argumentidena ja renderdab need malli kasutades. Komponendile käitumise lisamiseks - sel juhul nupuvajutuse käsitlemiseks - peame sellele lisama mõned koodid.

Lisaks templateile võib komponendil olla ka selleks otstarbeks mõeldud JavaScripti fail. Mine edasi ja loo sama nimega .js fail ning samasse kausta (app/components/people-list.js)

app > components > JS people-list.js > PeopleListComponent

```
1  import Component from '@glimmer/component';
2  import { action } from '@ember/object';
3
4  export default class PeopleListComponent extends Component {
5    @action
6    showPerson(person) {
7      alert(`The person's name is ${person}!`);
8    }
9  }
```

Nüüd, kui me oleme soovitud käitumise rakendanud, saame minna tagasi komponendi templatei ja ühendada kõik omavahel järgmiselt:

app > components > people-list.hbs > ul > li > button

```
1  <h2>{{@title}}</h2>
2
3  <ul>
4    {{#each @people as |person|}}
5      <li>
6        <button type="button" {{on 'click' (fn this.showPerson person)}}>{{person}}</button>
7      </li>
8    {{/each}}
9  </ul>
```

# Harjutus/ülesanne kui aega jääb üle/// aegunud

Teeme selleks eraldi folderi ja uue projekti. Ehk “ember init” ja siis “ember s”. Alustuseks võime “application.hbs” faili algse lehe ära kustutada

app > templates > application.hbs

```
1   {{page-title "EmberBlog"}}
2
3   Welcome to Ember
4
5   {{outlet}}
```

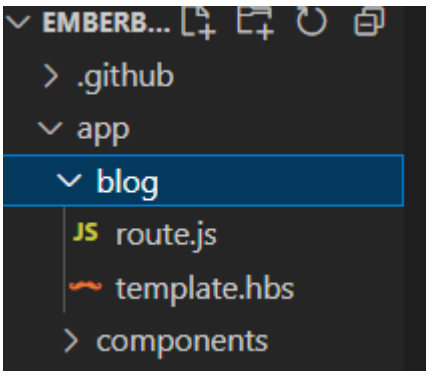
{{outlet}} Näitab routi nõ child route siis

## 4. Harjutus/ülesanne juhend pole hetke ülesandega seotud see oli hoopis pod layouti tutvustus vabandan kui keegi seda jälgis antud hetke ülesandel

Kasutame seekord rout-ide jaoks teist populaarset moodust. Selle asemel, et kõik routi failid eraldi folderitesse läheks kasutame pod layouti selleks avame faili “ember.cli”. Sisestame pod layouti

```
6       Setting `disableAnalytics` to true will prevent any
7     */
8     "disableAnalytics": false,
9
10    /**
11      Setting `isTypeScriptProject` to true will force the
12      rather than JavaScript by default, when a TypeScript
13    */
14    "isTypeScriptProject": false,
15    "usePods": true
16  }
17
```

Lisaks kustutame “routes” ja “controllers” folder-id kuna neid pole enam vaja . Sisestame terminali “ember g route blog” Nüüd uue rout-iga seotud asjad vaid ühes ainsas folderis




Lisaks kui peaks juhtuma, et olete kogemata teinud kirja veal routi loomisel ja soovite routimisega failid eemaldada siis selleks on käsklus “ember destroy route ‘RoutiNimi’ ” ilma `märgita. Siis lisame pathi router.js failis lehele

app > router.js > ...

```
1   import EmberRouter from '@ember/routing/router';
2   import config from 'ember-blog/config/environment';
3
4   export default class Router extends EmberRouter {
5     location = config.locationType;
6     rootURL = config.rootURL;
7   }
8
9   Router.map(function () {
10    this.route('blog', {path: '/myblog'});
11  });
12
```

Teeme lehe loogitat natuke ilusamaks muutes faili “application.hbs”

app > templates > application.hbs > footer > p

```
1   {{page-title "EmberBlog"}}
2
3   <h2 id="title">Welcome to Ember</h2>
4
5   {{outlet}}
6
7   <footer>
8     <p>This is a simple example of an ember app</p>
9   </footer>
```

Kuid loome nüüd postituse enda mudeli luues mudeli folderisse faili “post.js”

app > models > JS post.js > default

```
1   import DS from 'ember-data';
2
3   const{
4     Model,
5     attr
6   } = DS;
7
8   export default Model.extend({
9     title: attr('string'),
10    author: attr('string'),
11    content: attr('string')
12  });
```