# Cheat Sheet

**!!! Да си направя скелет за проекта, в който после да добавя Public с css / images / views !!!**

# !!! Да видя в коя папка са static files в html-ите !!!

1. Initialize project and structure

**✔️ npm init –yes**

**✔️ Create > src/index.js**

2. Setup dev environment

✔️ npm i -D nodemon

**В package.json:**

"scripts": {

"start": "nodemon src/index.js"

},

**► ► ► TEST:   
В index.js:**

console.log('Server start')

**✔️ npm start**

3. Install and setup express

**✔️ npm i express**

**► ► ► TEST:**

**В index.js:  
const express = require('express');**

const app = express();

// Test action

app.get('/', (req, res) => {

res.send('First action');

});

app.listen(5000, console.log('Server is listening on port 5000...'));

// GO TO <http://localhost:5000/>

\* add static middleware

**✔️ Create > src/public**

**В index.js:**

// Middleware

app.use(express.static('public'));

// \* add body parser

**В index.js:**

// Body parser

app.use(express.urlencoded({extended: false}));

// \* add routes

**✔️ Create > src/routes.js**

В routes.js:

const router = require('express').Router();

// TODO: add controller router

module.exports = router;

В index.js:

const routes = require('./routes');

...

app.use(routes);

→ на мястото на // Test action.. app.get(‘/’,…..

→ В routes.js отива целия // Test action, като app.get става router.get

► ► ► TEST:

// GO TO <http://localhost:5000/>

// 4. Add static resources

**→ Вземаме от static папката и ги слагаме в нашата public**

// 5. Add views folder with ready htmls

**✔️ Create > src/views и в тях се добавят html-ите**

// 6. Add express-handlebars view engine

// \* install

**✔️ npm i express-handlebars**

// \* add to express

В index.js:

const handlebars = require('express-handlebars');

// \* config extension

В index.js:

// Add handlebars

app.engine('hbs', handlebars.engine({

extname: 'hbs'

}));

app.set('view engine', 'hbs');

// \* config views folder (for src folder)

app.set('views', 'src/views');

// \* add main layout

✔️ Create > src/views/layouts – main.hbs

**→ От home.html вземаме всичко с навигацията (obiknoweno do </header>) и го слагаме в main.hbs.**

**→ Вземаме и footer-a заедно с края.**

**--> Между тях слагаме**

{{{body}}}

// \* add partials folder

✔️ Create > src/views/partials

// \* fix navigation to home

// \* render home page

**В routes.js променяме на:**// Test action

router.get('/', (req, res) => {

res.render('home')

});

**И можем да променим home.html на home.hbs.**

► ► ► TEST:

// GO TO http://localhost:5000/

// \* fix styles and images (static paths)

В index.js:

const path = require('path');

…….

app.use('/static', express.static(path.resolve(\_\_dirname, 'public')));

// 7. Add controllers folder with home controller

**✔️ Create > src/controllers и в нея homeController.js.**

**→ Местя от routes.js тук:**const router = require('express').Router();

router.get('/', (req, res) => {

res.render('home');

});

module.exports = router;

**--> В routes.js:**

const router = require('express').Router();

const homeController = require('./controllers/homeController');

router.use(homeController);

module.exports = router;

► ► ► TEST:

// GO TO http://localhost:5000/

// 8. Add database

// \* install mongoose

**✔️ npm i mongoose**

// \* connect database

В index.js:

const mongoose = require('mongoose');

…

// TODO change DB name

mongoose.connect('mongodb://127.0.0.1:27017/petstagram')

.then(() => console.log('DB connected successfully'))

.catch(err => console.log('DB Error, ', err.message));

// 9. Authentication

// \* add user controller

✔️ Create userController.js.

// \* add controller to routes.js

*const* userController = require('./controllers/userController');

**...**

router.use('/users', userController);

В userController.js:

*const* router = require('express').Router();

router.get('/login', (req, res) *=>* {

res.render('users/login');

});

module.exports = router;

// \* render login page

✔️ Create > src/views/users

**--> login.html отива в нея и става login.hbs**

**--> Edit като остава само <main>...</main>**

► ► ► TEST:

// GO TO http://localhost:5000/users/login

// \* fix header navigation to login and register

Отиваме в main.hbs / header и оправяме линковете:

*<!--For Guest User-->*

<li *class=*"nav-item">

*<!-- Link to Login Page -->*

*<!-- TODO # -->*

<a *href=*"/users/login">

<i>Login</i>

</a>

</li>

<li *class=*"nav-item">

*<!-- Link to Register Page -->*

*<!-- TODO # -->*

<a *href=*"/users/register">

<i>Register</i>

</a>

...

*<!-- Link to Logout Page -->*

*<!-- TODO # -->*

<a *href=*"/users/logout">

<i>Logout</i>

</a>

...

<a *href=*"/">

<img *width=*"50px" *src=*"/static/images/free-30-instagram-stories-icons23\_122570.png" *alt=*"img1">

</a>

...

<a *class=*"home" *href=*"/">

<i>Petstagram</i>

</a>

...

// \* render register page

**В userController.js:**

**router.get('/register', (req, res) *=>* {**

res.render('users/register');

});

**✔️ View-то:**

**--> register.html отива в нея и става register.hbs**

**--> Edit като остава само <main>...</main>**

► ► ► TEST:

**// GO TO** [**http://localhost:5000/users/register**](http://localhost:5000/users/register)

// 10. Add user model

**✔️ Create > src/models – User.js**

***const* userSchema = new mongoose.Schema({**

username: {

type: String,

required: *true*,

},

email: {

type: String,

required: *true*,

},

password: {

type: String,

required: *true*,

}

});

*const* User = mongoose.model('User', userSchema);

module.exports = User;

// 11. Add user manager

**✔️ Create > src/managers – userManager.js**

// \* add login and register methods (empty)

const User = require('../models/User');

exports.login = () => {

}

exports.register = () => {

}

exports.logout = () => {

}

// \* require in user controller (empty)

*const* userManager = require('../managers/userManager');

// 12. Modify login and register forms

<form action="*URL*"> ако ни трябва

In login.hbs - *method=*"POST:

<!-- Start Login Form-->

<form *method=*"POST">

...

<input *type=*"text" *name=*"username" *placeholder=*"Username">

...

<input *type=*"password" *name=*"password" *placeholder=*"Password">

...

<p>Don't have account? <a *href=*"/users/register">Register</a></p>

In register.hbs:

<form *method=*"POST">

<input *type=*"text" *name=*"username" *placeholder=*"Username">

<input *type=*"email" *name=*"email" *placeholder=*"Email">

<input *type=*"password" *name=*"password" *placeholder=*"Password">

<input *type=*"password" *name=*"repeatPassword" *placeholder=*"Repeat Password">

............

<p>Have an account? <a *href=*"/users/login">LogIn</a></p>

// 13. Add login and register post actions

In userController.js:

router.post('/login', *async* (req, res) *=>* {

*const* { username, password } = req.body;

*await* userManager.login(username, password);

res.send('Logged in');

});

......

Подават се както са в модела!!!

router.post('/register', *async* (req, res) *=>* {

*const* { username, email, password, repeatPassword } = req.body;

*await* userManager.register({username, email, password, repeatPassword});

res.send('Registered!');

});

ТЕСТВА се и се гледа в MongoDB Compass:

// 14. Implement user manager login and register

In user**Manager**.js:

// \* add register method

exports.register = (email, username, password, repeatPassword) *=>*

User.create(email, username, password, repeatPassword);

► ► ► TEST:

**// GO TO** [**http://localhost:5000/users/register**](http://localhost:5000/users/register)

// \* add unique index for username

username: {

type: String,

required: *true*,

unique: *true*,

},

// \* validate repeat password

In User.js:

userSchema.virtual('repeatPassword')

.set(*function* (value) {

*if* (this.password !== value) {

*throw* new Error('Password mismatch!')

}

});

// 15. Hash password

//    \* install bcrypt

npm I bcrypt

In User.js:  
userSchema.pre('save', *async* *function*(){

*const* hash = *await* bcrypt.hash(this.password, 10);

    this.password = hash;

});

// 16. Login

//    \* find user by username

exports.login = *async* (username, password) *=>* {

*const* user = *await* User.findOne({username});

*if* (!user) {

*throw* new Error('Invalid user or password');

    }

}

//    \* Validate password

userSchema.pre('save', *async* *function*(){

*const* hash = *await* bcrypt.hash(this.password, 10);

    this.password = hash;

});

// 17. Generate jwt token

//    \* install json webtoken

npm I jsonwebtoken

// npm i express-validator

*const* util = require('util');

*const* jsonwebtoken = require('jsonwebtoken');

*const* jwt = {

    sign: util.promisify(jsonwebtoken.sign),

    verify: util.promisify(jsonwebtoken.verify),

};

module.exports = jwt;

//    \* generate token in userManager login

// 18. Return token in cookie

//    \* install cookie parser

npm I cookie-parser

//    \* config cookie parser в index.js

*const* cookieParser = require('cookie-parser');

//    \* set cookie with token (userController)

//    \* **error container in main.hbs**

{{*#if* error}}

    <div>

        <div *class=*"errorContainer">

            <p>Error</p>

        </div>

    </div>

    {{*/if*}}