# Лабораториски вежби по Дизајн и архитектура на софтвер - Вовед во потребни технологии

Целта на лабораториските вежби е да се применат теоретските знаења од предавањата практично. За таа цел студентите ќе треба да развијат софтверска архитектура и да имплементираат веб апликација. Проектот ќе се работи во групи од најмногу 5 студенти.

## 1 Git

Проектите се предаваат преку GitHub. За да работите со GitHub потребно е да знаете да го користите системот за верзиска контрола Git. Тој ја олеснува работата на проекти во тимови и овозможува да ја видите историјата на промените, кој ги направил и како тие придонесуваат кон проектот.

### 1.1 Инсталација

Доколку го немате инсталирано Git на вашиот систем, тоа треба да го направите.
Ако користите дистрибуција на Linux базирана на Debian (пр. Ubuntu), можете да го инсталирате Git со наредбата

```
$ sudo apt install git-all
```

Ако користите Fedora или друга блиско поврзана дистрибуција базирана на RPM, инсталирајте го Git со:

```
$ sudo dnf install git-all
```

За повеќе опции, погледнете ја официјалната веб-страница (`https://git-scm.com/download/linux`).
Инсталацијата за Windows можете да ја преземете од официјалната веб-страница (`https://git-scm.com/download/win`). Освен ова постојат и други, неофицијални проекти кои Ви овозможуваат да го инсталирате Git на Widnows (Git for Windows, GitHub Desktop).
На MacOS можете да го инсталирате Git со инсталацијата од официјалната веб-страница ( `https://git-scm.com/download/mac`), со инсталација на Xcode Command Line Tools и извршување на наредбата `$ git --version` или како дел од GitHub for macOS.

## 1.2 Основни наредби

Еден git проект (repository) ги опфаќа сите датотеки и папки поврзани со некој проект, како и верзиската историја на секоја датотека. Оваа историја е покажана како снимки во дадено време наречени commits. Тие можат да бидат организирани во повеќе lines of development наречени branches. Интеракцијата со еден git repository се врши преку командната линија и овозможува преглед и интеракција со историјата, клонирање на проектот, креирање на branches, вршење на commits, спојување на branches, споредба на промени и др. Некои од најчесто користените Git наредби се:

- `git init` иницијализира нов Git проект и почнува да го следи.
- `git clone` прави локална копија на проект кој постои на далечина. Оваа копија ги вклучува сите датотеки, историја и гранки на проектот.
- `git add` ги додава промените. Git ги следи промените на кодот, но тие мора да се додадат и да се направи снимка од нив за да се вклучат во историјата на проектот. Оваа наредба го прави додавањето, првиот дел од процесот. Сите промени што ќе ги додадете ќе станат дел од следната снимка од историјата на проектот. Додавањето и зачувувањето посебно им овозможува на програмерите целосна контрола врз историјата на нивниот проект без да сменат како работат.
- `git commit` ја зачувува снимката во историјата на проектот и го комплетира процесот на следење на промените. Се што било додадено ќе биде дел од снимката. Секој commit има своја commit порака која дава краток опис на промените кои се зачувуваат.
- `git status` го прикажува статусот на промените.
- `git branch` покажува на кои гранки се работи локално.
- `git merge` ги комбинира промените направени на две различни гранки, на пр. главната гранка и гранката на некоја специфична функција.
- `git pull` ја ажурира локалната копија на проектот со промените во проектот што постои на далечина.
- `git push` го ажурира проектот што постои на далечина со промените направени на локална гранка.
- `git checkout` гранката која е дадена како аргумент се означува како гранката на која што моментално се работи.

## 1.3 GitHub

GitHub хостира git проекти. Доколку немате, креирајте корисничка сметка. Потоа креирајте нов repository за вашиот проект. За соработка, креирајте shared repository.
Работата со shared repository може да изгледа вака:

1. ажурирајте го локалниот проект со `git pull origin master`
2. креирајте ја гранката на која ќе работите `git checkout -b branchName`
3. направете промени во кодот, додавајте и зачувувајте ги почесто со `git add` и `git commit`
4. објавете ги промените со `git push origin branchName`
5. можете да ја споите вашата гранка со главната, но добро е да отворите pull request за другите членови на тимот да ги разгледаат вашите промени и да ги дискутирате пред да го сторите тоа.

# 2 Comman Line Interface (CLI)

Командната линија овозможува поефикасна и ефективна комуникација со компјутерот и постигнување на повеќе задачи отколку со графичките кориснички интерфејси.

## 2.1 Наредби за навигација

Некои од позначајните наредби за навигација се:

- pwd (Print Working Directory) - ја покажува патеката од root до сегашната локација
- mkdir (Make Directory) - направи нова папка
- ls (list) - покажи ја содржината на папката. Доколку не е даден аргумент, ја покажува содржината на сегашната папка/локација
- cd (Change Directory) - наредба за движење низ системот. Како аргументи може да има целосна патека до некоја локација, имиња на папки содржани во онаа во која моментало се наоѓате или .., со што можете да се префрлите во родителот на сегашната локација
- clear - го чисти терминалот
- $\uparrow$, $\downarrow$ - движење низ последните наредби
- со цртка по која следува буква можете да им задавате опции на наредбите. На пр., наредбата ls ги има опциите -a, -t, -l: ls -a ја принта целата содржина, вклучувајќи ги и скриените датотеки и папки, а ls -t ги принта сите датотеки и папки сортирани според времето кога последно биле модифицирани

## 2.2 Наредби за менување на системот

- cp (copy) - cp source destination ја копира содржината на датоеката или папката source во датотеката или папката destination
- mv (move) - mv source destination ја поместува датотеката source во папката destination
- rm (remove) - ја брише датотеката или папката дадена како аргумент. Со опцијата -r ја брише содржината на дадената папка

## 2.3 Редирекција на инпут и аутпут

- наредбата » го редиректира стандарниот аутпут на наредбата на лево и го додава (append) на датотеката на десно.
- наредбата | се вика pipe. Го пренесува стандардиот аутпут од наредбата лево од неа на стандардниот инпут на наредбата десно од неа.
- Симболот > го редиректира аутпутот од наредбата на лево и го предава како инпут на наредбата на десно.
- cat - ја прикажува содржината на една или повеќе датотеки во терминалот.
- grep - се користи за пребарување на датотеки. Со оваа наредба можат да се бараат линии кои одговараат на некоја шема и се враќаат резултатите.

  Наредбата grep со опцијата -i може да се користи за case insensitive пребарување. Со опцијата -R (grep -R) се пребаруваат сите датотеки во папката, вклучувајќи ги и нејзините под-папки и ги враќа имињата на датотеките и линиите кои содржат резултати кои одговараат на бараната шема.

## 2.4 Bash Scripting

- Можете да направите наредби што може повторно да се користат со bash скрипти. Овие скрипти можат да ги извршат сите наредби што можат да се извршат во терминалот.
- Датотеките на bash скриптите почуваат со #!/bin/bash. Ова му кажува на компјутерот да го користи bash за интерпретација.
- Може да се додаваат аргументи по името на скриптата. Ако се достапни, можете да пристапите до нив со $(позиција во листата на аргументи). На пр. до првиот аргумент се пристапува со $1, вториот со $2, итн.
- Во bash скриптите променливите се назначуваат со =, а до нив се пристапува со $.

- strings се споредуваат со == (еднакво) и != (различно).
- Наредбата read може да се користи за да се побара инпут од корисникот. Го чита внесот се додека не се притисне Enter.

  Со опцијата -p може да се додаде текст на барањето за инпут.

# 3 Web Architecture - Java Web Application

Во текот на овој курс ќе треба да имплементирате веб апликација. Во овој дел ќе биде опишано како да поставите ваква апликација во Java, HTML и JavaScript користејќи ги Maven, Dropwizard и Angular.

## 3.1 Вовед

### 3.1.1 Dropwizard

Порано, развивањето на веб апикации во Јава било поддржано со Јава апликациски сервери (Apache Tomcat, GlassFish, IBM WebSphere, JBoss, Jetty...), кои можат да содржат повеќе апликации истовремено. Тоа води до проблеми со разделувањето на апликациите и нивните ресурси. Од друга страна, за само една апликација не е потребен целосен апликациски сервер, па се јавуваат вградени верзии на ове Јава апликациски сервери (Jetty Embedded, Tomcat Embedded). Овие сервери можат лесно да се вградат во софтверски проекти како библиотеки, но може да немаат некои од саканите функции, како логирање или управување со корисници. Dropwizard е еден од проектите кои ги комбинираат вградените апликациски сервери со потребните библиотеки да поддржат поголем софтверски проект. Слични проекти се Spring Boot, Spark и Play.

### 3.1.2 Angular

Angular е рамка за мапирање на динамични податоци во статички распоред на страницата. Ги пополнува динамичките податоци од серверот во HTML распоредот и ги дава внесовите од корисниците на серверот. Angular манипулира со HTML-DOM дрвото и ја врши комуникацијата со серверот. Тука се покажани само мал дел од функционалностите на Angular. Со Angular се прават т.н. Singe-Page Applications (SPAs). Една SPA се состои од само една HTML страница, каде делови од HTML-DOM дрвото се менуваат да прикажат различни податоци и да реагираат на внесовите од корисниците. Во овој туторијал не се прави форматирање на HTML страницата. Во голем број проекти се користи Twitter Bootstrap3 за да помогне со форматирањето.

## 3.2 Организација и поставување на апликацијата

### 3.2.1 Maven

Maven е алатка за градење која се справува со зависностите во Java и го надгледува процесот на градење. Gradle2 е помодерна алтернатива. Можете да иницијализирате Maven проект во IDE (Integrated Development Environment) по избор. Eclipse, IntelliJ, NetBeans и други IDEs кои се фокусирани на Java и имаат поддршка за Maven би требало да функционираат без проблем. Во овј туторијал се користи верзијата на Maven 3.
Кога комуницирате со Maven директно, а не преку IDE, можете да креирате нов проект со

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes
-DgroupId=mk.finki.dians -DartifactId=SaWebApp
```

(SaWebApp е името на апликацијата, mk.finki.dians името на групата).
Во IDE, користете ги дијалогните прозорци за да го креирате проектот. Проектот можете да го компајлирате со

```
mvn compile
```

, а да го спакувате со

```
mvn package
```

Мора да ги извршите и двете наредби за Maven да може да го најде pom.xml.

### 3.2.2 Angular

За да се подеси Angular-Cli, потребно е да е инсталиран NodeJS и да е отворена конзолата на NodeJS. NodeJS има свој систем за управување со пакети. За да ги ажурирате сите инсталирани пакети на NodeJS, извршете ја наредбата

```
npm update -g
```

Потоа може да се инсталира Angular-Cli со

```
npm install -g @angular/cli
```

Сега Angular-Cli е достапен со командата ng. Порака за помош може да се прикаже со

```
npm --help
```

Креирајте нов проект со

```
ng new SaWebApp
```

Сите наредни команди треба да се извршат од папката SaWebApp креирана од Angular-Cli. Наредбата исто така ќе ја изврши наредбата npm install на NodeJS за да ги исталира сите потребни зависности.
Angular-Cli автоматски ќе започне веб сервер кога ќе се изврши наредбата

```
ng serve
```

Резултатот можете да го видите во веб прелистувач на http://localhost:4200/. Оваа наредба може да не успее доколку не ја извршите со потребните дозволи. Ако ова се случи, обидете се повторно како super user. Со наредбата

```
ng serve --host 0.0.0.0 --port 4200
```

може да го смените интерфејсот и портата на веб серверот, при што IP 0.0.0.0 значи дека серверот слуша на сите интерфејси. Вградениот веб сервер работи во т.н. watch mode, значи, ако кои било HTML или JavaScript датотеки се сменат, веб серверот ќе ги детектира промените и автоматски ќе го смени погледот во прелистувачот.
По завршување на развојот, Angular-Cli може да направи компресирана верзија на HTML и JavaScript кодот. Конечната верзија на комплетната апликација може да ги користи HTML и JavaScript датотеките генерирани од овој процес. Командата на Angular-Cli за градење е

```
ng build
```

, а со дополнителната опција

```
ng build --prod
```

овој процес го вклучува и компресирањето и tree-shakeing. Tree-shaking значи дека неискористениот код од вклучените библиотеки е изоставен од конечните датотеки. Датотеките кои резултираат од овој процес се наоѓаат во dist. Некои од овие датотеки ќе имаат случаен хексадецимален број како дел од нивнто име. Овој број случајно се генерира во процесот на градење и се осигурува дека прелистувачите ќе ја користат најновата верзија, а не некоја стара верзија во зачувана во кешот.
Angular често користи TypeScript наместо JavaScript. TypeSript е синтаксички суперсет од JavaScript и алатката за градење на Angular го компајлира во обичен JavaScript. Овој процес овозможува да се детектираат синтаксички проблеми без да мора да се изврши JavaScript кодот во прелистувач. Сите грешки при компајлирањето можат да се прикажат со

```
ng serve
```

Angular следи модуларна структура за да ја раздели апликацијата. Најважните делови се:
Модули - сите Angular апликации се организирани во модули. Самата апликација се нарекува AppModule и ги опишува сите модули од кои се состои апликацијата.

Директиви - делови кои го менуваат DOM на веб-страницата, значи ја покажуваат и менуваат содржината на веб-страницата.
Компоненти - најчесто користениот тип на директиви. КОмпонентите се состојат од TypeScript компонент и HTML темплејт. HTML темплејтот дефинира како веб-страницата (или некои нејзини делови) треба да изгледаат. TypeScript компонентот ја содржи логиката за тоа како темплејтот треба да се пополни со податоци, да ги чита внесовите од корисникот од темплејтот, да ги менува податоците и како да комуницира со другите делови од апликацијата.
Сервиси - поголемиот дел од логиката се изведува тука. Иако е можно целата да се направи во TypeScript компонентот, препорачливо е да се направи во сервис.
TypeScript класи - се користат за моделирање на податоците и логиката како и во секој друг обејктно-ориентиран програмски јазик. Сервисите се во принцип TypeScript класи.
Поважни папки во проектот:
e2e - ги содржи сите end to end тестови.
node_modules - ги содржи сите NodeJS зависности. не е потребно да се префрла содржината на оваа папка на друга машина, нејзината содржина може да се земе со NodeJS наредбата `npm install`.
dist - процесот на градење ќе ги запише резултантините датотеки во оваа папка.
scr - ги содржи изворните датотеки за Angular апликацијата.
Останатите датотеки во the top папка SaWebApp ја содржат конфигурацијата за целиот проект и различни сервиси (пр. GIT ignore config)
Следно, треба да се креираат деловите од Angular апликацијата кои недостасуваат. Прво ги креираме комонентите на апликацијата

```
ng generate component SaMain
```

Оваа наредба автоматски ги генерира HTML и Typescript делот од компонентот. Сервисите можат да се креираат со

```
ng generate service SaHttp --module=app
```

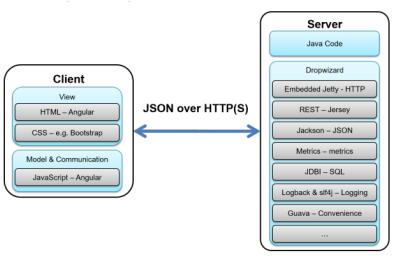
Се креираат две Typescript класи:

```
ng generate class Request
```

и

```
ng generate class Response
```

По потреба може да се креираат и други TypeScript класи.
Можно е да недостасуваат некои зависности по генерирањето на деловите. Тие може да се инсталираат со наредбата спомената погоре.




### 3.2.3 Архитектура на проектот

Целната архитектура на проектот е дадена на сликата подолу и е client-server архитектура. Клиентот целосно се извршува во веб-прелистувач. Angular дава поддршка за да ја поедностави комуникацијата со серверот, справи со внесовите од корисниците и да ги рендерира податоците од серверот.
Na страната на серверот, Dropwizard и сите Java библиотеки што ги нуди поставуваат добра платформа за Java кодот. Dropwizard ги нуди и компонентите на Java за конектирање собази на податоци или други веб сервиси. При компајлирањето, прво треба да се изгради Angular апликацијата, а потоа резултантите датотеки да се копираат во папката src->main->resources->assets на Java апликацијата. Потоа апликацијата може да се изгради со Maven.

# 4 Docker

Docker е алатка која овозможува лесно да се креираат, развиваат и инсталираат апликации преку корситењето на контејнери. Овие контејнери им дозволуваат на програмерите да ја спакуваат својата апликација со се што и е потребно, на пр. библиотеки и други зависности, и да ја инсталираат како еден пакет. Овие пакети се нарекуваат container images.

## 4.1 Инсталација

Docker можете да го преземете од официјалната веб-страница (`https://www.docker.com/`).

## 4.2 Основни наредби

Со наредбата `images` можете да ги видите сите images (слики) што се достапни локално.

```
$ docker images
```

За да преземете некоја image можете да ја користите наредбата `pull` или наредбата `run`. Ако не е достапна, наредбата `run` ќе ја преземе сликата автоматски.

```
$ docker pull postgres
// or
$ docker run postgres
```

Ако сакате да пристапите до некоја порта на вашиот контејнер, треба да ја мапирате на вашата локална мрежа со параметарот -p на наредбата run. Форматот е или -P за да ги направите сите порти достапни и да ги мапирате на истиот број на порта или -p hostPort:containerPort.

```
$ docker run -p 5000:5432 postgres
```

За да почнете контејнер во одвоен режим наместо во прежим на преден план, користете ја опцијата -d.

```
$ docker run -d -name awesome-db postgres
$ docker ps
$ docker stop e
// or
$ docker stop awesome-db
```

За да го зачувате тоа што е складирано во контејнерот можете да користите docker volumes, на тој начин не мора да се грижите ако го избришете контејнерот. Податоците ќе бидат зачувани се додека постои тој volume. Така, можеме да го отстраниме контејнерот секогаш кога ќе го запреме и да го мапираме на точниот volume кога ќе го стартуваме повторно.

```
$ docker volume create awesome-project-db
$ docker run --rm -v awesome-project-db:/var/lib/postgresql/data
postgres
```

За да ја добиете листата на сите постоечки volumes користете ја наредбата `docker volume ls`.
Со следните наредби можете да ги исчистите сите images и стопирани контејнери:

```
$ docker rm $(docker ps -a -q)
$ docker rmi $(docker images --filter dangling=true -q)
```

Првата наредба ги брише сите контејнери, а втората сите images кои не се користат повеќе. Осигурајте се дека сите контејнери се стопирани пред да ги извршите овие наредби.

### 4.3 Инсталација во облак

Docker има интеграција со Azure и AWS, со што можете да ја инсталирате вашата апликација во Azure Container Instances (ACI) и Amazon EC2 Container Service (ECS) со нативни наредби на Docker. Можете да прочитате повеќе за тоа на следните линкови:
`https://docs.docker.com/engine/context/aci-integration/`
`https://docs.docker.com/engine/context/ecs-integration/`
Бесплатни кориснички сметки за студењнти за овие сервиси можете да креирате на следните линкови:
`https://azure.microsoft.com/en-us/free/students/`
`https://aws.amazon.com/education/awseducate/`
Имајте на ум дека ќе мора да почекате неколку денови за потврдата на вашиот статус на студент, која е потребна за да ги користите сервисите.

## 5 Node.js

### 5.1 Инсталација

Препорачано е да го инсталирате node.js на вашиот систем со nvm (`https://github.com/nvm-sh/nvm`), што овозможува лесно да се инсталираат други верзии на node.js без конфликти, но можете да го симнете и од официјалната веб-страница (`https://nodejs.org/en/`).

### 5.2 Основни наредби

За да проверите дали инсталацијата е успешна, напишете ја командата `node` во командата линија. Пример:

```
$ node
> console.log("Hello World!");
Hello World!
undefined
>
```

Можете да иницијализирате нов проект со `npm init`. Ова ја олеснува соработката и ја генерира датотеката `package.json`, со која можете да ги конфигурирате името, описот, авторот, верзијата, скриптите, зависностите и др.

#### 5.2.1 Пакети

За да не мора да го пишувата истиот код повеќе пати, nodejs има систем на пакети за внесување на надворешни библиотеки. Можете да ги инсталирате овие пакети преку официјалниот nodejs package manager (npm), кој се

инсталира автоматски заедно со nodejs. Алтернативно, можете да го користете yarn. yarn ги кешира пакетите локално, па процесот на инсталација е побрз.
Имате две можности за инсталирање на пакети: или да го инсталирате пакетот глобално, за да можете да го користите во сите проекти со истата верзија на пакетот, или да го инсталирате директно во папката на проектот, за да можете да користите различни верзии на пакетот во различни проекти.
Обично само алатките за командната линија се инсталираат глобално (како is-up-cli или caniuse-cmd). Во спротивно инсталираниот пакет нема да биде означен како зависност во датотеката package.json.
За да инсталирате пакет и да го означите како зависност треба да ја извршите наредбата `npm install --save express` (express е името на пакетот). Препорачано е секогаш да ја користите опцијата `--save`, затоа што на тој начин членовите на тимот ќе треба само да ја извршат наредбата `npm install` за да ги инсталираат или ажурираат сите дефинирани зависности.
За да инсталирате пакети со yarn следете ги инструкциите на веб-страницата (`https://classic.yarnpkg.com/en/docs/cli/install/`). Доколку сте го инсталирале node со npm, можете и да го инсталирате yarn со наредбата `npm install --global yarn`.
Една предност на yarn е тоа што автоматски генерира датотека yarn.lock, која ја чува верзијата на пакетот инсталирана во проектот. На тој начин сите членови на тимот ќе ја имаат истата верзија на зависностите на проектот (иако со npm 5 и повисоко ќе добиете датотека package-lock.json file по default (npm-documentation)). Исто така, нема да мора да ја користите опцијата `--save`. Кога ќе инсталирате пакет со yarn (наредба: `yarn add express`), новата зависност автоматски се додава во package.json.
Процесот на иницијализирање на нов проект може да изгледа вака:

```
$ mkdir awesome-project
$ cd awesome-project
$ yarn init -y
$ git init
$ echo "node_modules/" > .gitignore
$ mkdir src
$ touch src/index.js
```

### 5.2.2 Simple Web Server

За разлика од PHP, на node.js не му е потребен надворешен веб-сервер, туку обезбедува пакет http со кој можете да креирате свој веб-сервер:

```
var http = require("http");

var server = http.createServer(function(req, res) {
res.writeHead(200);
```

```
res.end("Hello World!");
});

server.listen(3000);
```

За да го тестирате, одете на http://localhost:3000/ во вашиот прелистувач или користете ја наредбата curl/wget од конзолата (пр: curl localhost:3000). Треба да добиете Hello World! како одговор.
Со дадениот пристапот би морале сами да ги парсирате елементите. Пакетите како што е express можат да го олеснат ова.

### 5.2.3 Модули

Со системот на модули на nodejs можете да го поделите проектот во повеќе датотеки. Пример за проект кој ги содржи датотеките index.js и common.js:

```
// index.js
var common = require("./common");

console.log(common.add(1, 2));
```

```
// common.js
module.exports = { };

module.exports.add = function(a, b) {
return a + b;
}

/*
* If we want to use ES2015, which is possible (99%) in
* node 6.12 and beyond you could use arrow functions:
*
* module.exports = { };
* module.exports.add = (a, b) => a + b;
*/
```

Битно е да се забележи дека кога се вклучуваат пакети параметарот на require го прима само името на пакетот, додека кога станува збор за датотека, потребо е да му се даде патеката до таа датотека.

## 5.3 Пример апликација

Иницијализација на docker и postgres:

```
$ docker volume create awesome-project-db
$ docker run -e POSTGRES_PASSWORD=mysecretpassword --rm -d -p 5432:5432
-v awesome-project-db:/var/lib/postgresql/data postgres
```

Креирање на структурата на проектот:

```
$ mkdir awesome-project
$ cd awesome-project
$ yarn init -y
$ yarn install pg express mustache body-parser
$ mkdir src
$ touch src/index.js
```

Поврзување со базата на податоци:

```
// src/index1.js
const { Client } = require("pg");
const client = new Client({
user: "postgres",
host: "localhost",
database: "postgres",
password: "mysecretpassword",
port: 5432
});

client.connect()
.then(() => {
return client.query("SELECT $1::text as message", ["Hello World!"]);
})
.then(res => {
console.log(res.rows[0].message);
client.end();
})
.catch(err => {
console.error(err.stack);
});
```

Ќе го користиме mustache за внесување на вредности во html страниците. Со mustache можеме да дефинираме темплејти на страниците и да ги вчитаме темплејтите и да рендерираме вредностите во нив. Следниот пример покажува едноставен темплејт кој потоа се рендерира.

```
// src/index2.js
const mustache = require("mustache");

const template = ‘
<div>
{{name}}

<ul>
{{#todos}}
<li>{{title}}</li>
```

```
{{/todos}}
</ul>
</div>
`;

const data = {
name: "Pippi",
todos: [
{ title: "Buy milk" }
]
};

const output = mustache.render(template, data);

console.log(output);
```

По извршувањето треба да добиеме:

```
$ node src/index2.js

<div>
Pippi

<ul>
<li>Buy milk</li>
</ul>
</div>
```

Сето ова можеме да го споиме во следната апликација:

```
// src/index.js
const express = require("express");
const bodyParser = require("body-parser");

const client = require("./database");
const router = require("./router");

// Init database
client.connect()
.then(() => {
client.query(`
CREATE TABLE IF NOT EXISTS todos (
id SERIAL PRIMARY KEY,
title VARCHAR(100) NOT NULL,
done BOOLEAN DEFAULT FALSE
)
`);
```

```
})
.catch(err => {
console.error(err.stack);
});

// src/database.js
const { Client } = require("pg");

module.exports = new Client({
user: "postgres",
host: "localhost",
database: "postgres",
password: "mysecretpassword",
port: 5432
});

// src/router.js
const fs = require("fs");
const path = require("path");
const mustache = require("mustache");

const client = require("./database");

const template_dir = path.join(__dirname, "templates");

function getTemplate(template, data) {
return new Promise((resolve, reject) => {
fs.readFile(path.join(template_dir, template), (err, data) => {
if (err) reject(err);
resolve(data.toString());
});
});
}

function sendTemplate(res, template, data) {
getTemplate(template, data)
.then(page => {
res.send(mustache.render(page, data));
})
.catch(err => res.status(500).send(err.stack));
}

module.exports = (app) => {
app.get("/", (req, res) => {
sendTemplate(res, "home.html", {});
});
```

```
app.get("/todos", (req, res) => {
client.query("SELECT * FROM todos")
.then(result => {
sendTemplate(res, "todos.html", { todos: result.rows });
});
});

app.get("/todo/add", (req, res) => {
sendTemplate(res, "add-todo.html", {});
});

app.post("/todo/add", (req, res) => {
client.query("INSERT INTO todos (title) VALUES ($1)", [ req.body.title ])
.then(result => {
sendTemplate(res, "add-todo.html", {});
})
.catch(err => res.status(500).send(err.stack));
});
};
```

# References


[1] Node.js tutorial. [Online]. Available: https://github.com/SA-TU-Graz/sa-public/blob/master/node-tut/tutorial.md

[2] Git handbook. [Online]. Available: https://guides.github.com/introduction/git-handbook/

[3] Git book. [Online]. Available: https://git-scm.com/book/en/v2

[4] Docker tutorial. [Online]. Available: https://github.com/SA-TU-Graz/sa-public/blob/master/docker-tut/tutorial.md

[5] Web architecture - java web application tutorial. [Online]. Available: http://kti.tugraz.at/staff/rkern/courses/sa/2017/Web%20Architecture%20Tutorial.pdf

[6] Docker overview. [Online]. Available: https://docs.docker.com/get-started/overview/

[7] Angular tutorial. [Online]. Available: https://angular.io/tutorial

[8] Introduction to node.js. [Online]. Available: https://nodejs.dev/learn