



# AKADEMIJA.IT

INFOBALT IR TECH CITY

# ĮVADAS. GIT

Andrius Stašauskas

[andrius@stasauskas.lt](mailto:andrius@stasauskas.lt)

<http://stasauskas.lt/itpro2018/>

## INFORMACIJA

- 14 paskaitų - 13 teorija/praktika + 1 konsultacija
- Pirmadieniais, Antradieniais ir Ketvirtadieniais 8:55 - 12:45
- 5 savaitės:
  - Lapkričio 19, 20, 22 d. - 3 paskaitos
  - Lapkričio 26, 27, 29 d. - 3 paskaitos
  - Gruodžio 3, 4, 6 d. - 3 paskaitos
  - Gruodžio 10, 11, 13 d. - 3 paskaitos
  - Gruodžio 17, 18, 19, 20 d.
    - 1 paskaita, 1 konsultacija, 1 testas, 1 kontrolinis



# KURSO PROGRAMA

- Versijų kontrolės sistemos: Git
- Web UI technologijos, jų istorija
- Javascript moduliai plačiau: scoping, unit testai, paketu/modulių administratorius, transpiliavimas
- ReactJs biblioteka
- Verslo sistemų kūrimas
  - maven, tomcat, spring boot, istorija, mvc, request/reponse, DI, layers, rest api, unit testing
- Persistence (JPA)
  - orm, persistence api, jpql, transactions, integracija su spring, entities, relationships (multi), CRUD



# KĄ GAMINSIME

- Gaminsime supaprastintą el. parduotuvės variantą
- Vartotojo sąsaja - Javascript + ReactJs + Bootstrap CSS
- Serveris - REST Api (Spring Boot + Java 8)
- Duomenų bazė - H2 reliacinė duomenų bazė + JPA (Java Persistence Api)
- Tikslai:
  - kurso programos techninis tikslas - išmokti ir sukurti pilnai veikiančią aplikaciją
  - aukštesnis tikslas - suprasti modernaus Web kūrimo kontekstą, mokėti atpažinti naudojamas technologijas



## EL. PARDUOTUVĖS FUNKCIJOS

- Peržiūrėti prekių pasiūlą
- Peržiūrėti prekės detales
- Įsidėti prekę į krepšelį
- Peržiūrėti krepšelio turinj
- Pašalinti iš krepšelio
- Pirkti prekes krepšelyje
- Administruoti prekes: pridėti, išmesti, atnaujinti



# GIT PRADMENYS



AKADEMIJA.IT  
INFOBALT IR TECH CITY

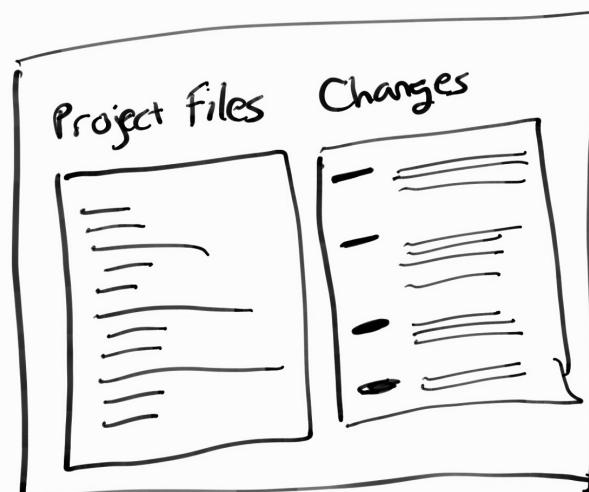
## TURINYS

- Versijų kontrolės istorija
- Git versijų kontrolė
- Git šakos
- Git nutolusios repozitorijos
- Užduotys tarp teorijos



# VERSIJŲ KONTROLĖ

- Angliškai - version control, revision control, source control
- Skirta suvaldyti dokumentų pakeitimus, kuriuos atlieka keli vartotojai
- Įsivaizduokite, jog visi turite įrašyti savo vardą į tą patį tekstinį failą

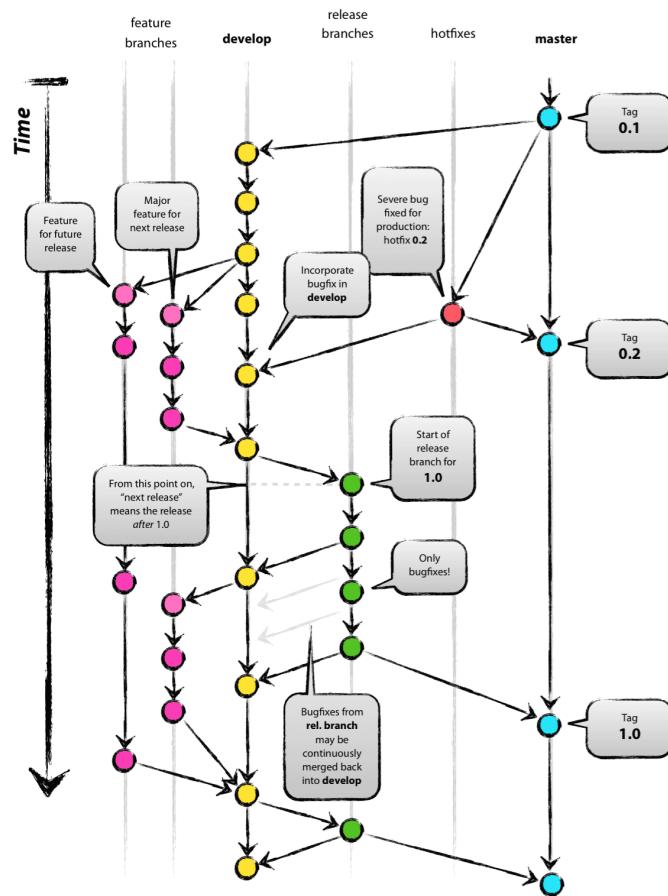


# VERSIJŲ KONTROLĖS KARTOS

Karta	Tinklas	Pvz
Pirma	Jokio	RCS, SCCS
Antra	Centralizuotas	CVS, Subversion, Team Foundation Server
Trečia	Išskirstytas	Bazaar, Git, Mercurial



# ŠAKOS



# ŠAKOS



@ashk3l

Repo arba Repository - centralizuotas katalogas



AKADEMIJA.LT  
INFOBALT IR TECH CITY

## KOMANDŲ PAGALBA

```
$ git help <command>
```

Komandos ir jos parametru aprašymas

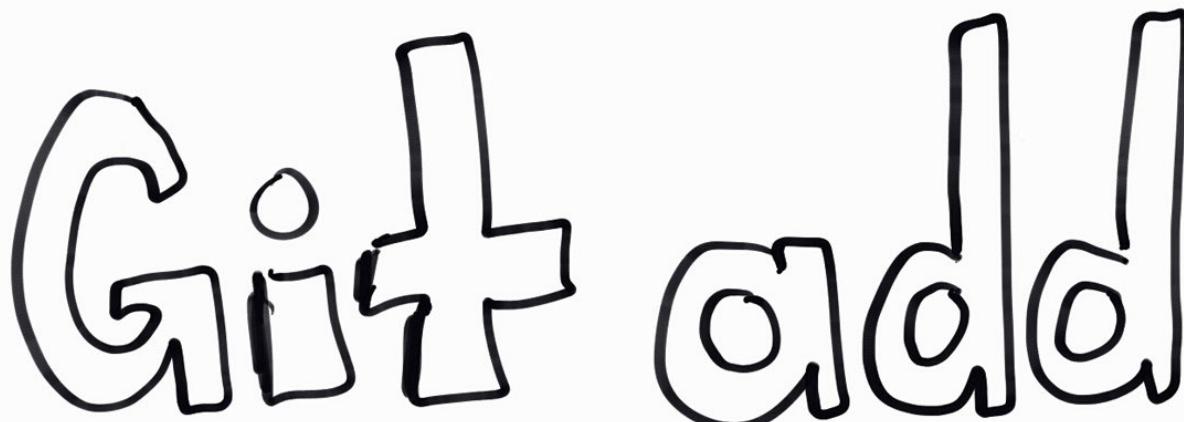


## REPOZITORIJOS PARUOŠIMAS

\$ git init - paruošia dabartinį katalogą kaip git  
repozitoriją



# FAILŲ PRIDĒJIMAS



Git add

@ashk3l

```
$ git add <path>
```

Pridėti viską iš dabartinio katalogo:

```
$ git add .
```



# FAILŲ PRIDĒJIMAS



@ashok31

\$ git status - dabartinė repozitorijos būsena.

- Po pakeitimų git status + git log



# FAILŲ PRIDĒJIMAS

```
→ testproject git:(master) ✘ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README.md
    renamed:   oldname.txt -> newname.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README.md
    deleted:   test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    newfile.txt

→ testproject git:(master) ✘
```

Bus "sucommitinti" žalių failų pakeitimai

# SUPAKUOTI Į COMMIT'Ą

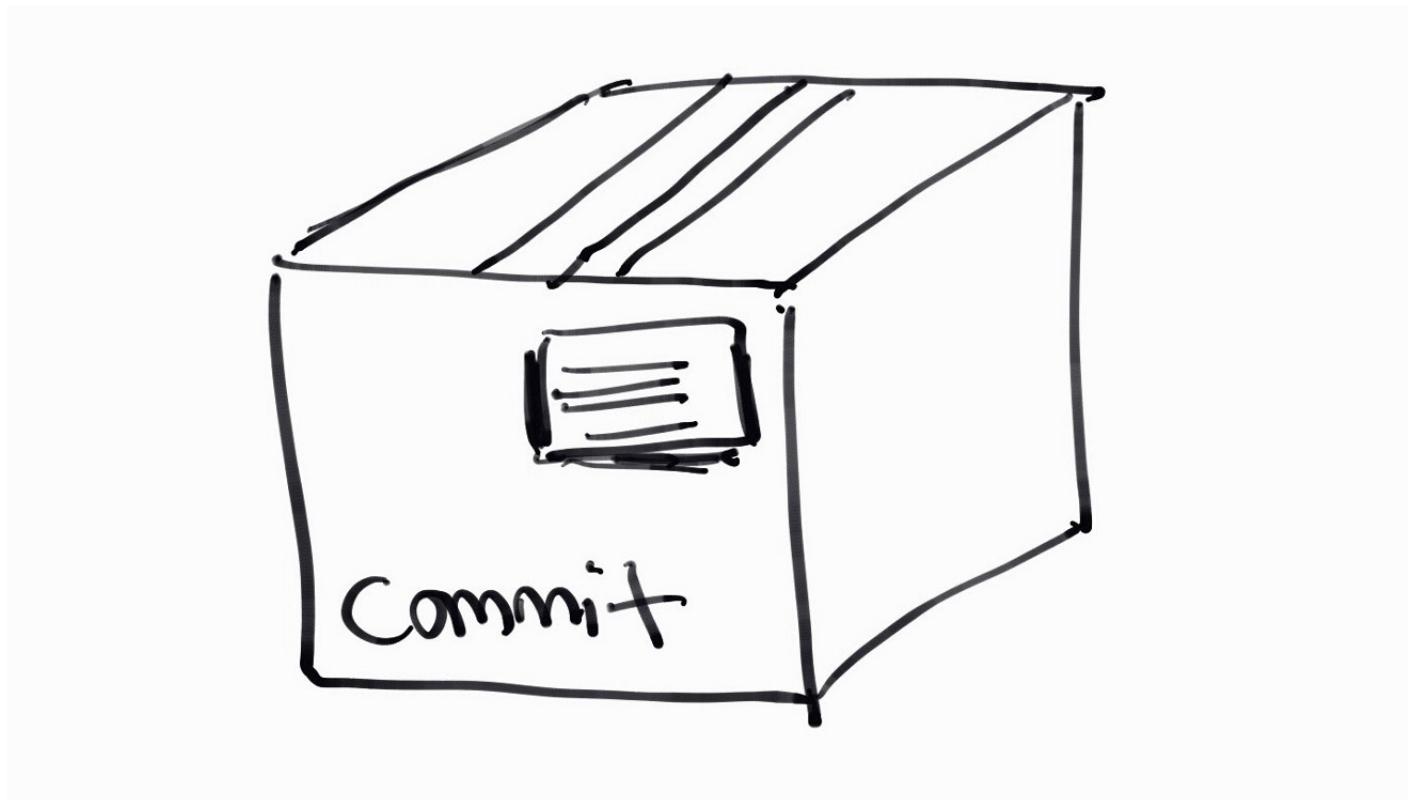
# Git Commit

@ashk31

```
$ git commit -m <message>
```

Komanda, atliekama pabaigus darbą ar jo etapą su failais.  
Susikuria changeset'as, turintis informaciją apie  
pasikeitusius failus.

# SUPAKUOTI Į COMMIT'Ą



@ashk31

Pakeitimai supakuoti į commit'ą kuris nebemodifikuojamas



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# COMMIT'Ų ISTORIJA

\$ git log - commit'ų istorija dabartinėje šakoje

```
commit c0326d2386dd1227f35f46f1c75a8f87e2e93076
Author: Hudson @ build.clojure.org <build@clojure.com>
Date:   Fri Oct 28 14:24:47 2016 -0500

[maven-release-plugin] prepare for next development iteration

commit e3c4d2e8c7538cfda40accd5c410a584495cb357
Author: Hudson @ build.clojure.org <build@clojure.com>
Date:   Fri Oct 28 14:24:47 2016 -0500

[maven-release-plugin] prepare release clojure-1.9.0-alpha14

commit b80e1fe4b14654d943e2f8b060b0bc56e18b4757
Author: Nicola Mometto <brobronsa@gmail.com>
Date:   Fri Oct 7 21:23:39 2016 +0100

    CLJ-1242: equals doesn't throw on sorted collections

    Signed-off-by: Stuart Halloway <stu@cognitect.com>

commit 2ff700ede3866f97d7b1f53342e201df94384aee
Author: Nicola Mometto <brobronsa@gmail.com>
Date:   Sat Nov 7 02:58:40 2015 +0100

    CLJ-1790: emit a cast to the interface during procol callsite creation

    Signed-off-by: Stuart Halloway <stu@cognitect.com>

commit c6b76fad4750c8f73d842cfdf882b4a05683cae
Author: Alex Miller <alex.miller@cognitect.com>
Date:   Fri Oct 28 08:42:26 2016 -0500

    CLJ-2024 stest/check should resolve function spec

    Signed-off-by: Stuart Halloway <stu@cognitect.com>
```



## UŽDUOTIS 1

- susikurkite katalogą: studentai
- inicijuokite tame git repozitoriją
- sukurkite failą studentai.dat
- atverkite failą ir išrašykite į jį savo vardą ir pavardę
- "sucommitinkite" pakeitimus
- peržvelkite, ar istorijoje atsirado commit'as



# UŽDUOTIS 1 - SPRENDIMAS

```
$ mkdir studentai
$ cd studentai
$ git init
$ echo "Jonas Petraitis" >> studentai.dat
$ git add studentai.dat
$ git commit -m "Added my name to studentai.dat"
$ git log
```



# UŽDUOTIS 1

- Pirmą kartą paleidus git

```
git config --global user.email "you@example.com"  
git config --global user.name "Your Name"
```

- Pirmas pakeitimas add, commit, bet po antro pakeitimo  
nėra add ir/ar commit



## .GITIGNORE

- dažnai turime failų, kurie neturi būti commit'inami
  - .eclipse konfigūracija
  - target katalogas atsirandantis build'o metu
  - kiti failai specifiniai konkrečiam vartotojui
- .gitignore tekstinis failas leidžia nurodyti šablonus kelių, kurie neturi būti tvarkomi Git'o



# .GITIGNORE

```
.gradle  
/build/  
!gradle/wrapper/gradle-wrapper.jar  
h2  
logs  
  
target  
*.iml  
.idea
```



# ŠAKOS (BRANCH)



AKADEMIJA.LT  
INFOBALT IR TECH CITY

## ŠAKOS (BRANCH)

- pagal nutylėjimą sukuriamas "master branch'as"
  - Mercurial default, SVN trunk, CVS HEAD
- naujus branch'us kuria vartotojai
- kiekvienai programavimo užduočiai naudojami atskiri branch'ai. Pabaigtos užduoties pakeitimai perkeliami į "master" branch'ą



# ŠAKOS (BRANCH)

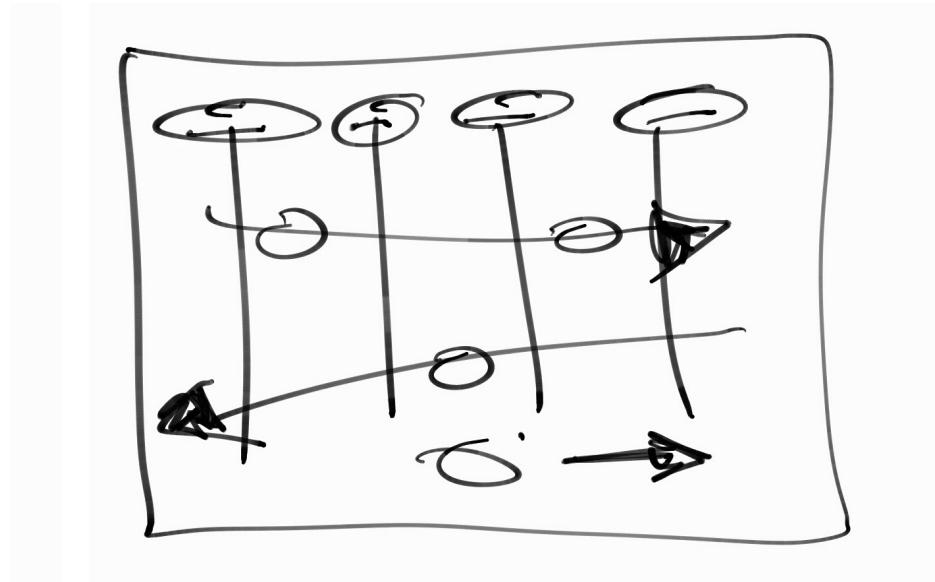
```
$ git checkout -b test
Switched to a new branch 'test'

$ git branch
  master
* test

$ git checkout master
Switched to branch 'master'

$ git branch
* master
  test
```

# ŠAKOS (BRANCH)



\$ git merge <branch-name> - Atlikti merge commit'ą,  
perkeliant nurodyto branch'o pakeitimus į dabartinį  
branch'ą

# ŠAKOS (BRANCH)

```
$ git branch
* master
  test
$ echo "master branch tekstas" >> file
$ git add .
$ git commit -m "Master commitas"
[master 2a5c8f1] Master commitas
 1 file changed, 1 insertion(+)
```



# ŠAKOS (BRANCH)

```
$ git checkout -b naujas
Switched to a new branch 'naujas'
$ echo "naujas branch tekstas" >> file
$ git add .
$ git commit -m "Naujas commitas"
[naujas 52f1927] Naujas commitas
 1 file changed, 1 insertion(+)
$ git checkout master
Switched to branch 'master'
```



# ŠAKOS (BRANCH)

```
$ cat file
master branch tekstas
$ git merge naujas
Updating 2a5c8f1..52f1927
Fast-forward
  file | 1 +
   1 file changed, 1 insertion(+)
$ cat file
master branch tekstas
naujas branch tekstas
```



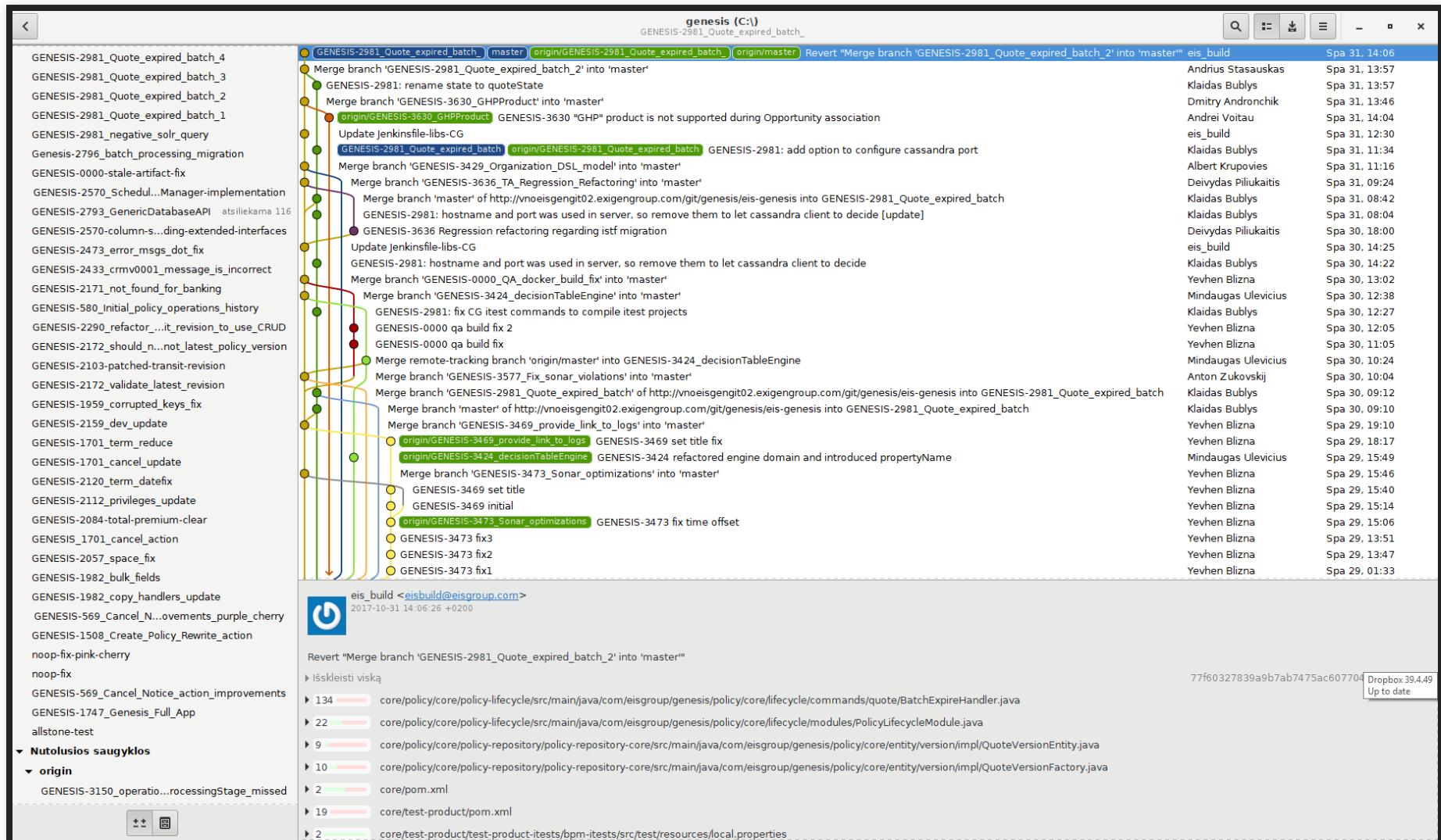
## GIT ISTORIJOS NARŠYMAS

- Dažnai labai patogu tai daryti grafiniais įrankiais
  - gitg, gitk, git gui (git-gui)
- Jei git status kažkas yra, patogiau git gui

```
$ sudo apt-get install gitg gitk git-gui  
$ gitg
```



# GITG



## GIT RODYKLĖS (POINTERS)

- HEAD - dabartinės būsenos rodyklė
- Daugiau pavyzdžių

<http://www.paulboxley.com/blog/2011/06/git-caret-and-tilde>



## COMMIT ID

- 52f192703d9951f5bbaa55fc41c0c78bba811a49
- unikaliai nusakyti commit'ą pakanka 7 pirmų simbolių  
52f1927
- žinant commit'o id, galima atlikti visokių veiksmų,  
pavyzdžiui nukreipti HEAD'ą į šį commit'ą



# COMMIT ID

```
$ git checkout 52f1927
```

You are in 'detached HEAD' state. You can look around, make experiments and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch `to retain commits you create`, you do so (now or later) by using `-b` with the checkout command again. Exa

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 52f1927... Naujas commitas
```



## MERGE KONFLIKTAI

- Įvyksta kai ta pati failo vieta buvo modifikuota kelių vartotojų
- Galimi merge ir pull komandų metu
- Ištaisius konfliktus failuose, juos reikia add'inti ir "sucommitinti" nekeinčiant commit'o pranešimo
- Konfliktų sprendimą palengvina grafiniai įrankiai (pvz kdiff, meld, tortoisemerge)

```
$ git mergetool
```



## MERGE KONFLIKTAI

Sukonfigūruoti galima .gitconfig

```
[merge]
    tool = kdiff3
[mergetool "kdiff3"]
    path = /usr/bin/kdiff3
```

- Sukonfigūravus merge'inti daug greičiau ir patogiau su git mergetool (vietoj teksto)
- Po merge'ų peržiūrai kas atsitiko patogiau naudoti gitg
  - apt-get install gitg



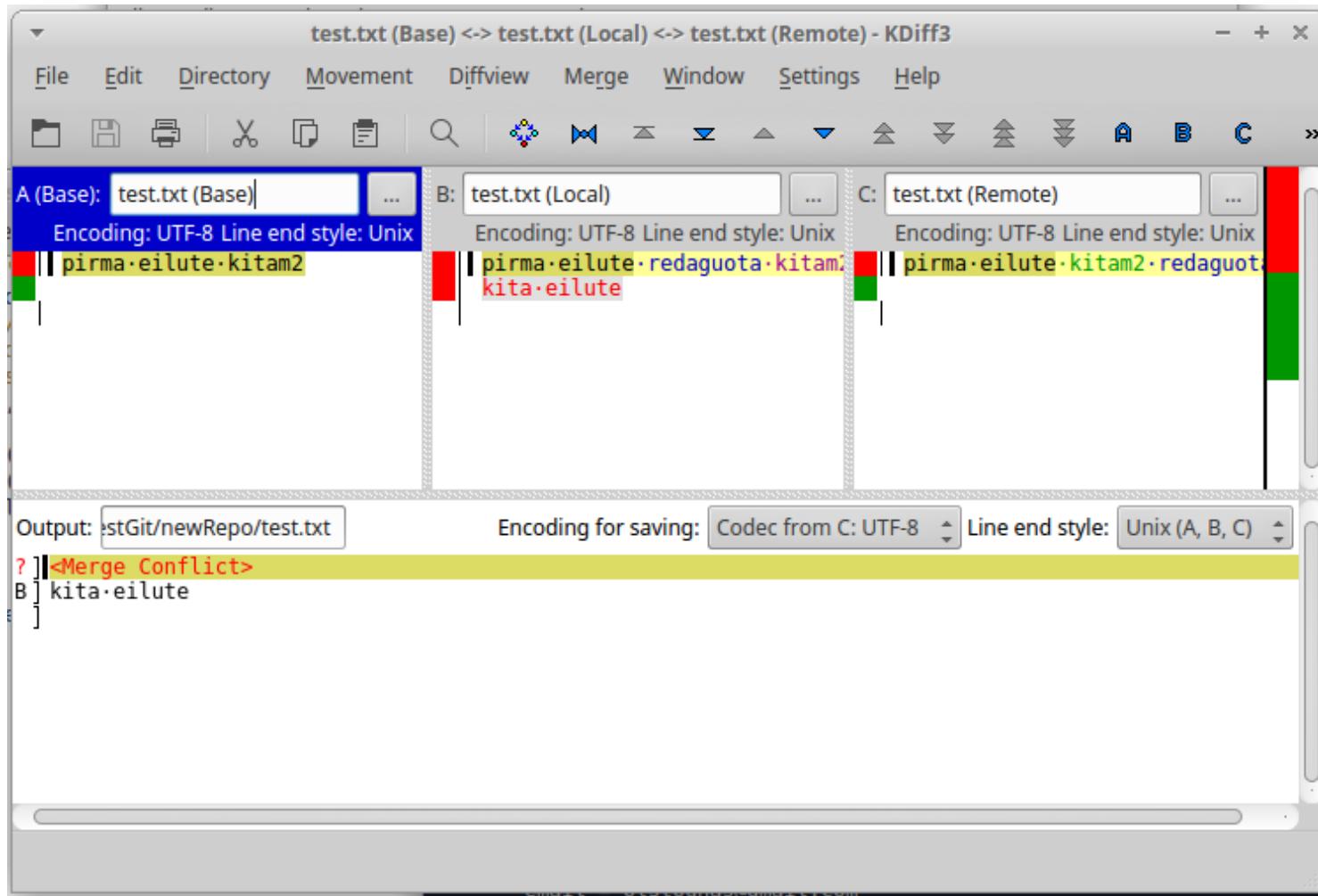
## KONFLIKTO PAVYZDYS - TEXT

```
Jonas Petraitis
<<<<< HEAD
1985-01-09
=====
Vilnius
>>>>> gimimo-vieta
```

- <<<<< ir >>>>> žymi konfliktuojančios dalies pradžią ir pabaigą
- <<<<< HEAD nurodo, jog konfliktuojantis blokas yra iš branch'o, į kurį "merginame"
- ===== atskiria dalis tarp branch'ų
- >>>>> gimimo-vieta nurodo, jog dalis tarp ===== ir >>>>> yra iš branch'o gimimo-vieta



# KONFLIKTO PAVYZDYS - KDIFF



## UŽDUOTIS 2

- Įsitikinkite, kad esate master branch'e
- Sukurkite branch'ą pavadinimu gimimo-data
- Tame branch'e į failą studentai.dat naujoje eilutėje įrašykite savo gimimo datą
- "Sucommitinkite"
- Grįžkite į master branch'ą



## UŽDUOTIS 2

- Sukurkite branch'ą gimimo-vieta
- Tame branch'e į failą studentai.dat naujoje eilutėje įrašykite savo gimimo vietą
- "Sucommitinkite"
- Grįžkite į master branch'ą



## UŽDUOTIS 2

- "Įmerginkite" gimimo datos pakeitimus į master branch'ą
- "Įmerginkite" gimimo vietas pakeitimus į master branch'ą



## UŽDUOTIS 2 - SPRENDIMAS

```
$ git checkout -b gimimo-data  
Switched to a new branch 'gimimo-data'
```

```
$ echo "1985-01-03" >> studentai.dat  
$ git add .  
$ git commit -m "Gimimo data"  
[gimimo-data 1da96ec] Gimimo data  
 1 file changed, 1 insertion(+)
```

```
$ git checkout master  
Switched to branch 'master'
```

## UŽDUOTIS 2 - SPRENDIMAS

```
$ git checkout -b gimimo-vieta
Switched to a new branch 'gimimo-vieta'
```

```
$ echo "Vilnius" >> studentai.dat
$ git add .
$ git commit -m "Gimimo vieta"
[gimimo-vieta a309462] Gimimo vieta
 1 file changed, 1 insertion(+)
```

```
$ git checkout master
Switched to branch 'master'
```

## UŽDUOTIS 2 - SPRENDIMAS

```
$ git merge gimimo-data
Updating f9f86f9..1da96ec
Fast-forward
 studentai.dat | 1 +
 1 file changed, 1 insertion(+)
```

```
$ git merge gimimo-vieta
Auto-merging studentai.dat
CONFLICT (content): Merge conflict in studentai.dat
Automatic merge failed; fix conflicts and then commit the result.
```



## UŽDUOTIS 2 - SPRENDIMAS

```
$ gedit studentai.dat # pataisome konfliktus failo
```

```
Jonas Petraitis
<<<<< HEAD
1985-01-03
=====
Vilnius
>>>>> gimimo-vieta
```

```
Jonas Petraitis
1985-01-03
Vilnius
```

```
$ git add .
$ git commit
[master 5c8fae3] Merge branch 'gimimo-vieta'
```



# NUTOLUSIOS REPOZITORIJOS

## DECENTRALIZACIJA

- Kiekviena Git'o repozitorija yra kažkurio laiko momento kopija.
- Ji gali nepriklausomai vystytis
- Dažniausiai yra Git Serveris, kuris yra centrinė repozitorija (origin)
- Populiariausi serveriai: <https://github.com>,  
<https://gitlab.com>, <https://bitbucket.org>



## NUTOLUSIOS (REMOTE) REPOZITORIJOS

- Dažniausiai būna viena centrinė remote repozitorija, vadinama 'origin'
- Kuriant naują git repozitoriją (`git init`) ją reikia prisdėti pačiam (`git remote add`)
- Klonuojant (`git clone`) egzistuojančią repozitoriją, ji pridedama automatiškai
- Kad repozitorijos veiktu kartu, jų bazė turi būti ta pati



## NUTOLUSIOS (REMOTE) REPOZITORIJOS

```
$ git remote add <name> <repository>
```

```
$ git remote add origin https://github.com/allstone/itpro.git
$ git remote -v
origin  https://github.com/allstone/itpro.git (fetch)
origin  https://github.com/allstone/itpro.git (push)
```

- Slaptažodžio pateikimas nuorodoje
  - git clone [https://username:password@github.com/..](https://username:password@github.com/)



## NUTOLUSIOS (REMOTE) REPOZITORIJOS

```
$ git clone <repository>
```

```
$ git clone https://github.com/allstone/itpro.git
```



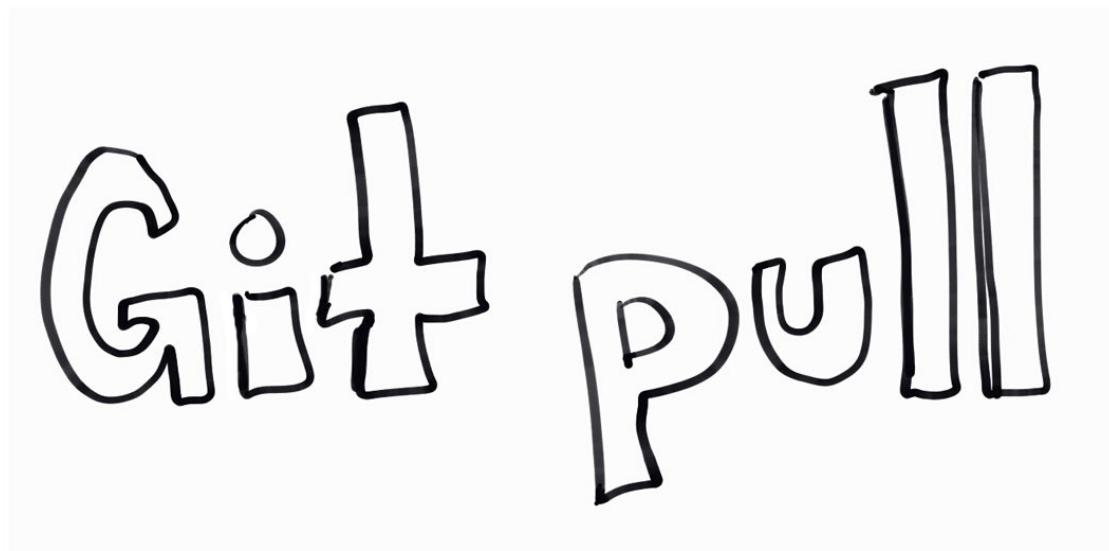
# NUTOLUSIOS (REMOTE) REPOZITORIJOS

## remote ar clone repositorijos radimas

The screenshot shows a GitHub repository page for a project named 'allstone test'. The top navigation bar includes links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. Below the navigation, a message states 'No description, website, or topics provided.' with an 'Edit' button. A 'Manage topics' link is also present. Key statistics are displayed: 1 commit, 1 branch, 0 releases, and 1 contributor. A dropdown menu shows the current branch is 'master'. Buttons for 'New pull request' and file operations ('Create new file', 'Upload files', 'Find file') are visible. A prominent green button labeled 'Clone or download' is highlighted with a black border. A tooltip for this button provides instructions: 'Clone with HTTPS' (with a link to 'https://github.com/allstone/itpro.git') and 'Use SSH'. Below the repository list, a note encourages adding a README. At the bottom right of the page, there are 'Open in Desktop' and 'Download ZIP' links.



# LOKALIOS REPO SINCHRONIZACIJA



@ashk3l

\$ git pull arba \$ git fetch



# PAKEITIMŲ ATSISIUNTIMAS



@ashk31

Gauname viską, kas keitėsi, t.y. ką kiti supakavo į commit'us  
ir yra nusiuntę į remote repository



## PULL VS FETCH

- fetch - atsiisiunčia pakeitimus iš repozitorijos, bet nepritaiko jų lokalai kopijai
- pull - padaro tą patį, ką ir fetch, bet pritaiko gautus pakeitimus, bet dėl to gali reikėti merge'o
  - techniškai git pull atveju vyksta tokia komandų seka, jei esame master'yje:

```
$ git fetch  
$ git merge remotes/origin/master
```

- dažniausiai visada darome git pull ir nesukame galvos
- atsinaujinti visus branch galima su git pull --all

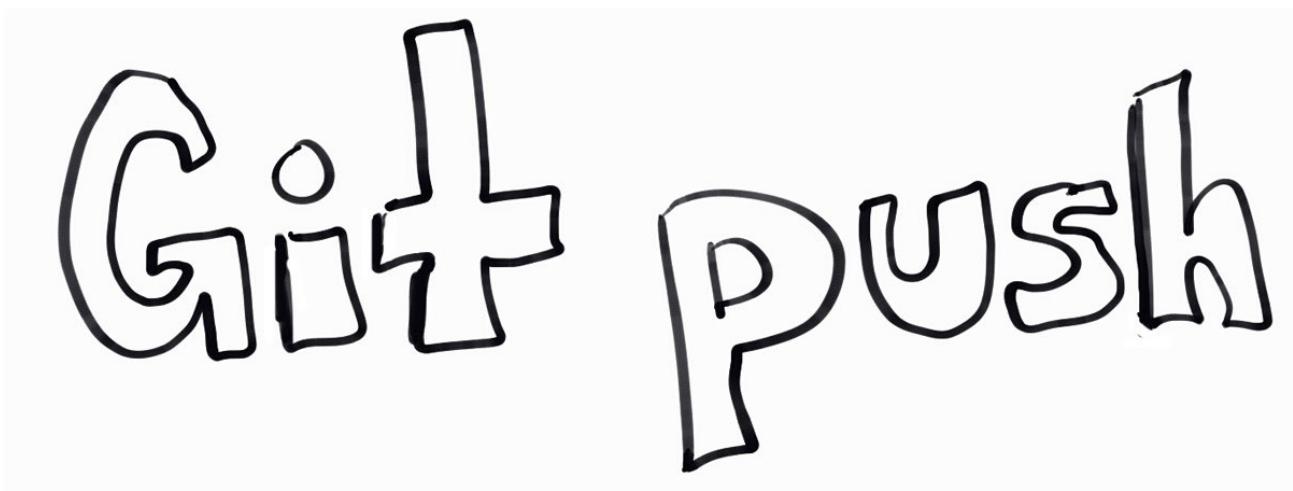


# LOKALIŲ PAKEITIMŲ IŠSIUNTIMAS Į REMOTE'Ą



- norime išsiųsti visus savo commit'us į remote repository, kad jie būtų pasiekiami kitiems

# LOKALIŲ PAKEITIMŲ IŠSIUNTIMAS Į REMOTE'Ą



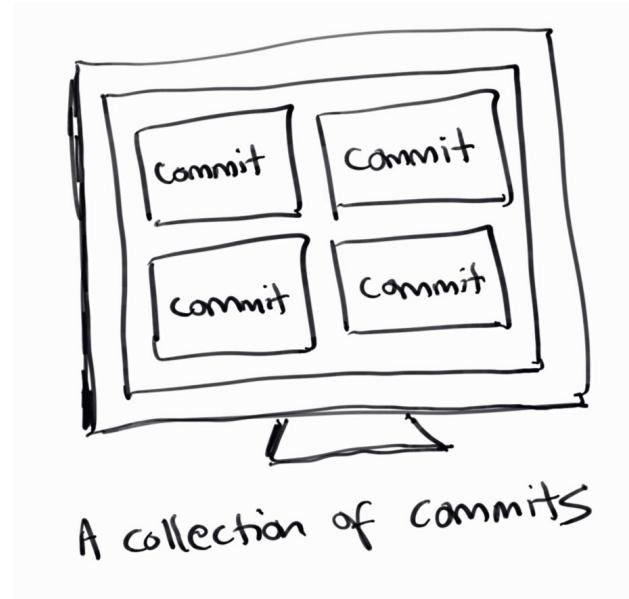
@ashk31

```
$ git push origin <branch-name>
```

- jei remote yra naujų pakeitimų, bus atspausdinta klaida
  - tokiu atveju reikia atlikti git pull, išspresti merge konfliktus ir bandyti dar kartą



# KAS BUS SIUNČIAMA



- praktiškai bus išsiunčiami visi commit'ai, esantys branche



## UŽDUOTIS 3

- Susikurkite Github'o vartotoją
- Susikurti slaptažodj-token'ą, pirmą kartą atsiradusiamė lange jį būtinai nusikopijuoti
  - <https://github.com/settings/tokens>
- Susikurkite Gihtub'e repozitoriją
- Prisidėkite remote'ą
- Perkelkite į Github'ą prieš tai užduotyse sukurtą repozitoriją



## UŽDUOTIS 4

- Susigrupuojame į komandas po 3-4 žmones
- Nusprendžiame, kieno repozitoriją "teršime"
- Vienas iš komandos narių susikuria branch'ą su vienu failu ir išsiunčia į nutolusią repozitoriją
- Kiti parsisiunčia pakeitimus - "jeina" į branch'ą
- Iš pradžių įdedame atskirus naujus 3 failus, kiekvienas savo, - išsiunčiame
- Tada darome pakeitimus pagrindiniame faile visi kartu ir bandome išsiusti bei merge'inti
- Teisių suteikimas:  
<https://github.com/<user>/<repo>/settings/collaboration>



## ŠALTINIAI

- <https://git-scm.com/book/en/v2>
  - Pirmi 2 skyriai rekomenduojami
  - 3 skyrius smalsiems
- <http://learngitbranching.js.org/>
- dalis paveiksliučių: <https://medium.com/@ashk3l/a-visual-introduction-to-git-9fdca5d3b43a>



# KITOJE PASKAITOJE

Web evoliucija. Javascript moduliai. ReactJS įvadas



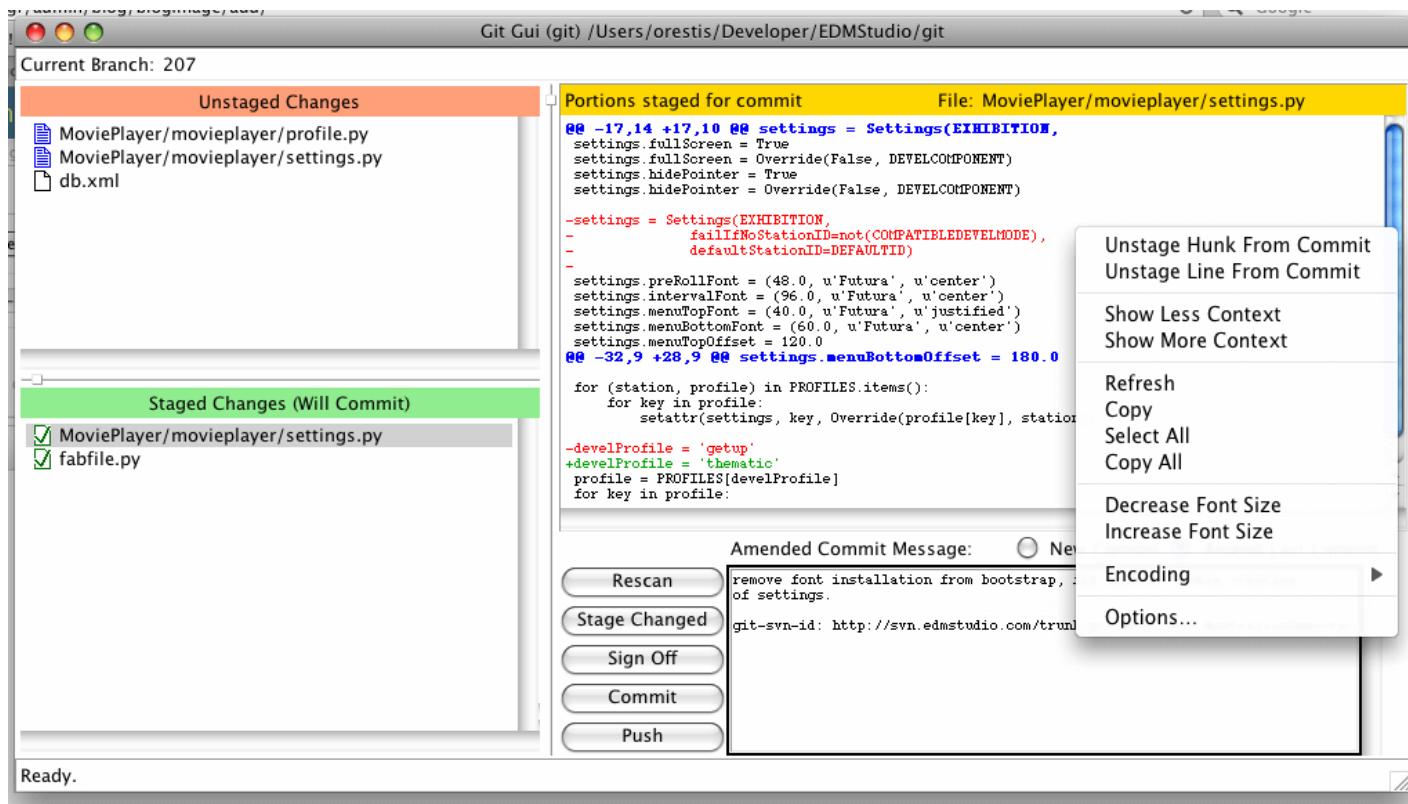
# IŠ PRAEITOS PASKAITOS

- su `git add` pridėtas pakeitimai yra valdomas git
  - norint jį pašalinti, reikia ne tik ištrinti iš katalogo, bet ir atlikti `git rm`
- konfigūracijos yra (pvz `.gitconfig` failams)
  - linux `/home/<username>`
  - windows `%UserProfile%`
  - mac `/Users/<username>`
- mac kdiff3 instaliuoti reikia su brew  
<http://macappstore.org/kdiff3/>



# IŠ PRAEITOS PASKAITOS

- ne visuomet, tačiau dažniausiai greičiau ir patogiau nei komandinė eilutė yra git gui / git-gui



# IŠ PRAEITOS PASKAITOS

- git versija virtualioje mašinoje

```
$ git --version // git version 2.17.0
```

- pasiimti vieną failą iš kito branch'o

```
$ git checkout 516e831 failas.txt  
$ git checkout master failas.txt
```

- klonuojant repozitoriją ateina tik master branch'as, o kitus reikia pull'intis
- jei nesigauna - galite supakuoti ir atsiųsti visą repozitoriją .zip formatu ir parašyti, kad tiksliai neišeina





# AKADEMIJA.IT

INFOBALT IR TECH CITY

## WEB EVOLIUCIJA. REACTJS

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# TURINYS

- Web evoliucija
  - Javascript istorija
  - Modernūs įrankiai
- React
  - JSX
  - stiliai ir kiti resursai
  - komponentai

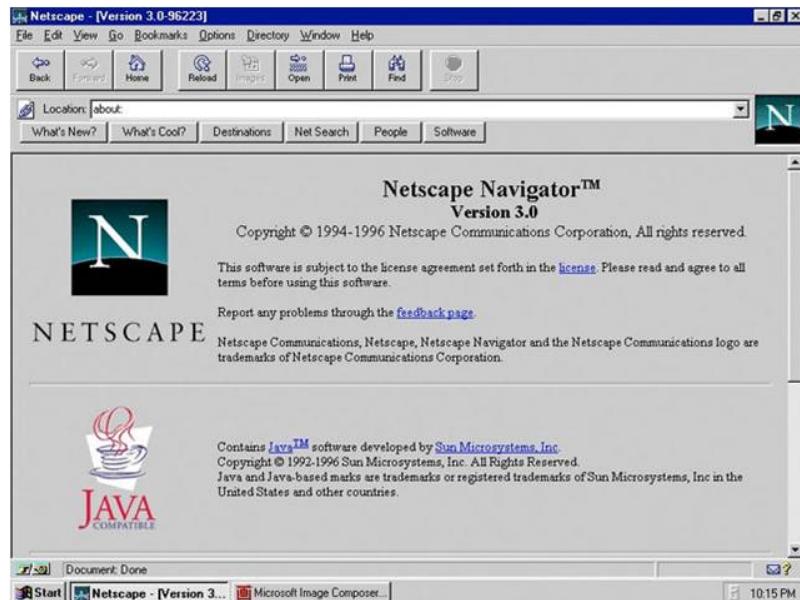


# INTRO

- 1991 m. Berners-Lee sukūrė ir paleido pirmajį web puslapį
  - <http://info.cern.ch/>
- Dramatiškas web technologijų pasikeitimas
- Modernus web kūrimas = daug įrankių, daug konceptų, galybė būdų padaryti tą patį
- Įrankiai sudėtingi, bet leidžia kurti tokias aplikacijas, kurių prieš tai negalėjome
  - ir daug greičiau
  - ir daug lengviau valdomų (maintain)



# WEB 1.0 (1991-2001)

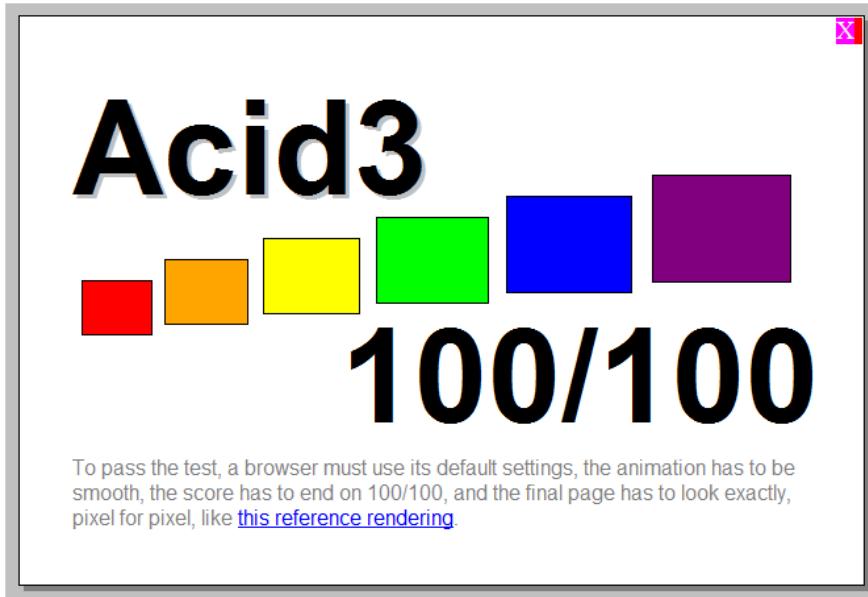


# WEB 1.0 (1991-2001)

- Tipiškas puslapis: static HTML, patalpintas GeoCities, parašytas naudojant FrontPage ar DreamWeaver
- Technologijos: /CGI-BIN -> Perl -> PHP/ASP -> Java / .NET
- Aplikacijos: skriptai ir vidutinio dydžio serveriai
- Interaktyvumas: Java appletai, ActiveX, DHTML
- Naršyklės: Mosaic, Netscape, IE
- Esminiai pasiekimai: gimė JS / HTML / CSS, išrasta Java ir Flash, multimedija per img ir iframe
- Kodo dalybos: "script list" svetainės
- Duomenų formatas: HTML su lentelėmis



# WEB 2.0 (2001-2010)



The screenshot shows a Microsoft Internet Explorer window displaying a WSDL (Web Services Description Language) document. The code is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://www.ClayShannon.com"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://www.ClayShannon.com" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.ClayShannon.com">
      <s:element name="GetDow">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="intYear" type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="intMonth" type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="intDay" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetDowResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetDowResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </types>
  <message name="GetDowSoapIn">
    <part name="parameters" element="s0:GetDow" />
  </message>
  <message name="GetDowSoapOut">
    <part name="parameters" element="s0:GetDowResponse" />
  </message>
  <portType name="DOWSoap">
    <operation name="GetDow">
      <input message="s0:GetDowSoapIn" />
      <output message="s0:GetDowSoapOut" />
    </operation>
  </portType>
</definitions>
```



ADOB  
E FLASH PLAYER



Microsoft®  
Silverlight™

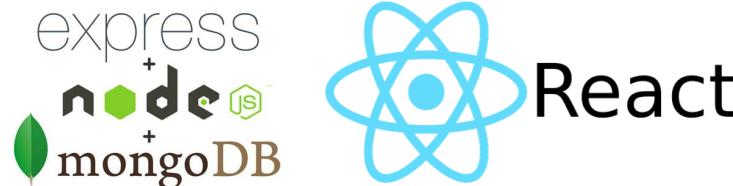


AKADEMIJA.IT  
INFOBALT IR TECH CITY

# WEB 2.0 (2001-2010)

- Tipiška svetainė: puslapiai, surenderinti serveryje, MySQL duomenų bazė, naudojama J2EE ar Rails
- Serveris: J2EE, ASP.NET, Rails, Django, LAMP, monolitinė architektūra
- Klientas: AJAX, MooTools/Prototype, jQuery
- Naršyklės: Mozilla -> Firefox, IE6 -> IE8, Opera, Chrome
- Rich Internet aplikacijos (Flash, Silverlight, GWT, EXT, YUI)
- Esminiai pasiekimai: AJAX, mobile - išmanieji įrenginiai, JSON, Stack Overflow
- Kodo dalybos: SourceForge, Google Code
- Duomenų formatas ir perdavimas: XML per SOAP/WS

# MODERNI ERA (2010-DABAR)



# MODERNI ERA (2010-DABAR)

- Tipiška svetainė: kliento JS, JSON API, mikroservisai su Node/Java paleisti per Amazon AWS, NoSQL DB
- Serveris: Node/Express, AWS, Java, ASP.NET MVC, Heroku
- Klientas: HTML5, Backbone, Angular, React, Bootstrap
- Naršyklės: Chrome, FF, Safari, Edge, visos "evergreen"
- Vieno-puslapio aplikacijos
- HTML5: canvas / WebGL, WebSockets, Web Workers, data storage, CSS flexbox, CSS grid, OS/native APIs
- Didžiausi pasiekimai: Node, naršyklės pluginų mirtis
- Kodo dalybos: decentralizuota versijų kontrolė, i.e. Github
- Duomenys: JSON -> binariniai per HTTP/WebSockets

# JAVASCRIPT ISTORIJA



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# JAVASCRIPT ISTORIJA

- 1995: Javascript kalba išrasta Brandon Eich iš Netscape
- 1999: ES3 specifikacija; XMLHttpRequest išrastas Microsoft
- 2001: JSON formatą popularina Douglas Crockford
- 2005: "AJAX" termino popularinamas
- 2006: jQuery išrandra John Resig
- 2008: ES4 bando įtraukti tipus (atšauktas)
- 2008: "Javascript: the Good Parts" parašyta Douglas Crockford



# JAVASCRIPT ISTORIJA

- 2009: ES5 specifikacija; Node.js išranda Ryan Dahl
- 2010: Underscore.js, Backbone.js ir Coffeescript išranda Jeremy Ashkenas
- 2012: Microsoft sukuria TypeScript
- 2014: Facebook sukuria Flow (silpniesni tipai)
- 2015: ES6 specifikacija (dar vadinama ES2015)
  - klasės, moduliai, iteratoriai, arrow funkcijos, binariniai duomenys, kolekcijos, generatoriai/promise'ai..



# JAVASCRIPT ISTORIJA

- 2016: ES7 specifikacija užbaigama (ES2016)
- 2017: ES8 specifikacija užbaigama (ES2017)
  - await/async kurie dirba su generatoriais/promise'ais
- 2018: ES9 specifikacija užbaigama (ES2018)
- Brian Terlson (ECMAScript redaktorius):

I do not believe types are in the cards for the near future. <...>  
Just adopting TypeScript as the standard would of course be great  
for TypeScript users <...> don't expect anything near term



# MODERNUS WEB KŪRIMAS



Kas tai yra??



# JAVASCRIPT IŠŠŪKIAI

- JS projektavimo tikslas buvo pasiekti, kad užvedus pele ant beždžionės ji imtų šokti
- dažniausiai vienos eilutės skriptai
- 10 eilučių buvo normalūs
- 100 eil. buvo jau dideli
- 1000 eil. niekas net negirdėjo apie tokius
- JS nebuvo projektuotas didesniams programavimui
  - viskas tuo ir buvo paremta



# JAVASCRIPT IŠŠŪKIAI

- nėra modulių sistemos
- nėra enkapsuliacijos
- prototipais paremtas paveldimumas
- nėra statinių tipų ar kompiliavimo
- objektai ir duomenys dinamiškai modifikuojami
- minimali standartinė biblioteka
- skirtinges naršyklės ir jų galimybės
- dokumentų atvaizdavimo modelis perdarytas aplikacijoms



# JS PROGRAMUOTOJO TIKSLAI

- minimizuoti siunčiamų baitų skaičių
- susitvarkyti su naršyklių suderinamumu
- užpildyti JS standartinės bibliotekos ir JS kalbos spragas
- dalintis kodu tarp aplikacijų
- kurti vis sudėtingesnes pilnai paruoštas aplikacijas, kurios .. tiesiog yra naršyklėje
- aplikaciją kurti naršyklei yra sunku
- reikia paskirti tiek pat laiko kiek ir kuriant duomenų bazę ir schemą ar servisu sluoksnį
- pagarbos UI!



# MODERNŪS ĮRANKIAI

- Babel - JS kompiliatorius (ES6/ES2015 -> ES5)
- TypeScript - ES6, bet statiskai tipizuotas
- SASS/LESS
- Modulių formatai: asinchroninis AMD, sinchroninis CommonJS, ES6 statinis analizavimas
- Pakavimas/leidimas
  - Node.js - V8 JS Chrome Engine
  - NPM - paketų manageris su Node



# MODERNŪS ĮRANKIAI - SURINKIMAS

- Kompiliavimas: ES6/TypeScript į ES5
- Grupavimas: daug failų į vieną
- Minifikavimas: pašalinami tarpai, komentarai, trumpiau pervadinami kintamieji
- Kodo atskyrimas: mažiau reikalinga - užkrauta vėliau
- Grunt: keletas užduočių per įskiepius
- Gulp: transformuoti duomenis žingsniais
- Browserify: CommonJS moduliai į naršykę
- Webpack: bet kokius modulius (AMD/CJS/ES6 modules, CSS, images, ... ) transformuoti naršyklei ir ne tik

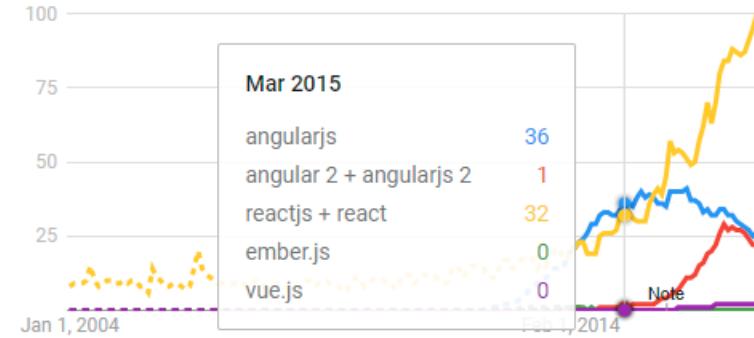


# KĄ NAUDOJA PROGRAMUOTOJAS

- Failų stebėjimas, kompiliavimas ir atnaujinimas
- Sourcemaps ir kiti naršyklės įrankiai (F12)
- Hot ("karštas") atnaujinimas
  - modulių pakeitimas, redagavimais gyvai
- Mocha, Jasmine, Tape, Jest ar Karma testai
- Chai, Expect, Sinon, JSDOM assert'ai
- Selenium, PhantomJS
- Klaidų/standartų tikrinimas-lintinimas: ESLint
- Bibliotekos: jQuery, Underscore/Lodash



# JAVASCRIPT KARKASAI (FRAMEWORKS)



# ATEITIS

- "X as a Service"
- "Serverless" backends - depends on BaaS backend as service
- Microservices, Containers
- Large-scale data transfer schemas/tools
- "Isomorphic" / "Universal" Javascript apps
- Server rendering
- Javascript everywhere
- Cross-platform toolkits (Cordova, Ionic)
- Desktop (Electron), Mobile (React Native)



# ATEITIS

- Component-based architectures
- "Virtual DOM"
- CSS-in-JS
- WebGL, Web Workers, Service Workers
- Functional Programming
- Immutable Data
- Reactive Programming / Observables
- Static typing



# UŽDUOTIS 1 - REDAKTORIAI

- MousePad arba Notepad++
- Eclipse senesnis
  - apt-get install eclipse openjdk-8-jdk
  - Help > Install New Software...
    - Eclipse Java EE Developer Tools
    - Eclipse Web Developer Tools
  - m2e - Maven Integration Plugin (maven projektui)
  - Eclipse failus su jsx kodu vadiname .jsx



# UŽDUOTIS 1 - REDAKTORIAI

- Eclipse naujesnis
  - <https://eclipse.org/downloads/eclipse-packages/>
  - Execute `eclipse-inst` per Xubuntu file manager
  - Help -> Eclipse Marketplace plugin'ai:
    - Eclipse Web Developer Tools
    - TypeScript IDE
  - Projektaj atidaryti
    - File -> Open Projects From Filesystem -> Directory



# UŽDUOTIS 1 - REDAKTORIAI

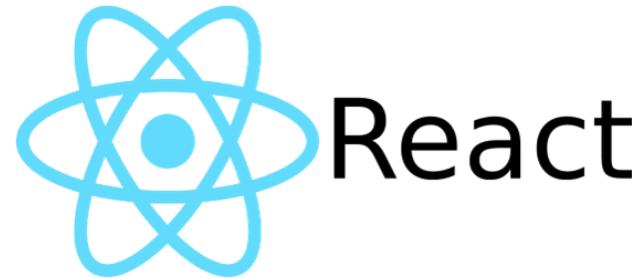
## Visual Studio Code

```
$ sudo add-apt-repository ppa:ubuntu-desktop/ubuntu-make  
$ sudo apt-get update  
$ sudo apt-get install ubuntu-make  
$ sudo umake ide visual-studio-code  
$ cd /.local/share/umake/bin  
$ ./visual-studio-code  
$ File -> Open Folder
```



# VARTOTOJO SĄSAJOS PROGRAMAVIMAS SU REACTJS





- MVC vaizdas, biblioteka, o ne karkasas
  - MVC ir susijusios architektūros - vėliau
- reikia pasirinkti dalis tam tikrai situacijai (būsenos valdymui, navigacijai, ..)
- įtakotas funkcinio programavimo ( $UI = f(būsenos)$  , vienkrypciniai duomenys)
- uždari komponentai ir žymėmis paremta JSX sintaksė ("HTML in JS")

# KODĖL BUVO SUKURTAS REACT

- Sunku buvo apjungti duomenis su UI
- Blogas UX naudojant "kaskadinus" DOM medžio atnaujinimus
  - React virtualus DOM per JS nes JS greitas
- Daug besikeičiančių duomenų
- Sudėtinga Facebook UI architektūra
- Parėjimas toliau nuo MVC mentaliteto



# PAPRASTAS 'HELLO WORLD' PAVYZDYS

```
<html>
<head>
    <title>E-Shop </title>
</head>
<body>
    <p>Labas</p>
</body>
</html>
```

\$ firefox index.html



# ĮTRAUKIAME STILIUS IR JS

```
<html>
<head>
    <title>E-Shop </title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <p class="custom-paragraph">Labas</p>

    <!-- <script src="app.js"></script> -->
</body>
</html>
```



## app.js

```
console.log('Pasileido')

for (var i = 0; i < 10; i++) {
  console.log('Pasileido ' + i)
}
```

## styles.css

```
.custom-paragraph {
  color: red
}
```



# REACT KOMPONENTAS

- Web aplikacijos blokai, panaudojami ne vieną kartą
- React komponentas - tai funkcija
  - funkcijai paduodama informacija, gaunamas output
- Funkcija apibrėžia UI laike
- Kuriami naudojant `React.createClass()`
- vienintelis būtinės metodas yra `render()`
- įterpiami naudojant `React.renderComponent()` arba `ReactDOM.render()`



# REACT KOMPONENTAS

The image shows a user interface built using React components. At the top, there is a navigation bar with three items: "Home" (purple), "Admin" (pink), and "Username" (blue). To the right of the navigation bar are two components: "UsernameComponent" and "CartSummaryComponent" (yellow). Below the navigation bar is a green-bordered container labeled "MenuItemComponent". Inside this container are three "ProductCardComponent"s, each featuring a smartphone image, a title, a description, and a "Details" button. The titles are "Samsung Phone 1", "Samsung Phone 2", and "Samsung Phone 3". Below this section is another green-bordered container labeled "ProductListComponent", containing three more "ProductCardComponent"s with titles "Samsung Phone 4", "Samsung Phone 5", and "Samsung Phone 6". Each card displays a smartphone image, a placeholder text "Desc", and a "Details" button.

MenuItemComponent

UsernameComponent

CartSummaryComponent

0 items

ProductCardComponent

Samsung Phone 1

Desc

20.6 Eur

Details

Samsung Phone 2

Desc

14.6 Eur

Details

Samsung Phone 3

Desc

14.6 Eur

Details

ProductListComponent

Samsung Phone 4

Desc

Samsung Phone 5

Desc

Samsung Phone 6

Desc



# UŽDUOTIS 2 - PRIDEDAM REACT

- I head dalį galima įsidėti Bootstrap, nors jis pačiam React nėra reikalingas:

```
<link  
    rel="stylesheet"  
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"  
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsv7Zt27NXFoaoApmYm8liuXoPkFOJwJ8ERdknLPM  
    crossorigin="anonymous"  
>
```

- Body pabaigoje React, ReactDOM, PropTypes, Babel ir appas:

```
<!-- // atkomentuoti savo kode  
<script src="https://unpkg.com/react@16/umd/react.development.js"></script>  
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>  
<script src="https://unpkg.com/prop-types@15.6/prop-types.js"></script>  
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>  
<script type="text/babel" src="app.js"></script>  
-->
```



# PIEŠIMAS (REACTDOM.RENDER)

## index.html

```
<div id="root"></div>
```

## app.js

```
const component = <h1>Hello, world</h1>

ReactDOM.render(
  component,
  document.getElementById('root')
);
```

- Susikurti branch'ą, nepamiršti git add , git commit , git push
  - kiekvieną dieną

# REACT KOMPONENTAI NAUDOJANT JAVASCRIPT

```
var HelloComponent = React.createClass({
  render: function() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
});

ReactDOM.render(React.createElement(
  HelloComponent,
  { name: "Jane" },
  document.getElementById('root')
));
```



# JSX PAGALBA

```
<MyButton color="blue" shadowSize={2}>  
  Click Me  
</MyButton>
```

sukompiliavus

```
React.createElement(  
  MyButton,  
  {color: 'blue', shadowSize: 2},  
  'Click Me'  
)
```



# JAVASCRIPT JSX

- JSX galima naudoti Javascript'ą

```
<Komponentas atributas1={2+3} atributas2={someVar} atributas3={a: 'b'}
```

- Šiuo atveju atributas1 paduodama JS išraiška, ji bus įvykdyta
- atributas2 priskiriama JS išraiška su kintamuoju. Jeigu kintamasis egzistuos JS kode, jo reikšmė bus priskirta
- atributas3 tiesiog yra priskiriamas objektas {a: 'b'}



## PASTABOS

- JSX leidžia rašyti "html" žymes, bet kai kurie dalykai daromi kitaip.
  - Norint nurodyti CSS klasę, vietoj atributo `class` rašome `className`
  - komponentų atributai dažnai keičiasi į formą, panašią java'ai:
    - pvz. iš `background-url` į `backgroundUrl`



# KOMPONENTAS NAUDOJANT JSX

```
var HelloComponent = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});

ReactDOM.render(React.createElement(
  HelloComponent,
  { name: "Jane" },
  document.getElementById('root')
));
```



# KOMPONENTAS NAUDOJANT JSX IR ES2015

```
class HelloComponent extends React.Component {  
    render() {  
        return <div>Hello {this.props.name}</div>;  
    }  
}  
  
ReactDOM.render(  
    <HelloComponent name="Jane"/>,  
    document.getElementById('root')  
) ;
```



## UŽDUOTIS 3

- pasibandyti visus pavyzdžius
  - su F12 Web/Developers Console (Ctrl+Shift+K) pažiūrėti, ką jie sukuria
- JSX pabandyti sudėtingesnį HTML kodą
  - kas neveikia ar veikia ne taip?
    - klausti arba googlinti
- turėtumėte pastebėti, kad React.createClass tiesiogiai React 16 iškvesti nebegalima
  - norint kurti su ES5 reiktų naudoti create-react-class biblioteką



# KOMPONENTŲ KOMPOZICIJA

# KOMPONENTŲ KOMPOZICIJA #1

```
class AvatarComponent extends React.Component {  
  render() {  
    return (  
      <img className="Avatar"  
        src={this.props.user.avatarUrl}  
        alt={this.props.user.name}  
      />  
    );  
  }  
}
```



# KOMPONENTŲ KOMPOZICIJA #2

```
class CommentComponent extends React.Component {  
    render() {  
        return (  
            <div className="Comment">  
                <div className="UserInfo">  
                    <AvatarComponent user={this.props.author} />  
                    <div className="UserInfo-name">  
                        {this.props.author.name}  
                    </div>  
                </div>  
            </div>  
        );  
    }  
}
```



# KOMPONENTO ATRIBUTAI (PROPS) #1

- `this.props` - komponentui perduoti atributai
  - gali būti programuotojo sugalvoti
  - React'o persiunčiami pagal nutylėjimą (`props.children`)
- Papildomai kiekvienas komponentas gali nusakyti kokių atributų tikisi
- `PropTypes` rodo klaidas tik tada, kai naudojamas React development režimas

```
ProductCardComponent.propTypes = {  
  id: React.PropTypes.number.isRequired,  
  image: React.PropTypes.string.isRequired,  
  title: React.PropTypes.string.isRequired,  
  description: React.PropTypes.string.isRequired,  
  price: React.PropTypes.number.isRequired,  
};
```



## KOMPONENTO ATRIBUTAI (PROPS) #2

kaip tokį komponentą iškvesti:

```
<ProductCardComponent  
  key={index}  
  id={product.id}  
  image={product.image}  
  title={product.title}  
  description={product.description}  
  price={product.price}  
/>
```



# KOMPONENTAS SU ATRIBUTU

```
class HelloComponent extends React.Component {  
    render() {  
        return (<div>Hello {this.props.name}</div>);  
    }  
}  
  
HelloComponent.propTypes = {  
    name: PropTypes.string.isRequired  
}  
  
ReactDOM.render(  
    (<HelloComponent name="Jane"/>),  
    document.getElementById('root')  
);
```



## UŽDUOTIS 4

- pakeisti propTypes
  - iš PropTypes.string į PropTypes.number
  - ką rodo konsolė naršyklėje?
- pakeisti name: į surname:
- pakeisti name=" į vardas="



## STILIAI

- ReactJs galima rašyti taip vadinamus `inline styles`, naudojant `style` atributą
- Vietoj to, kad aprašytume stilius CSS faile, jie yra rašomi Javascript'u
- Pagrindinė nauda - komponento stiliai įtakoja tik tą komponentą, kuriame jie aprašyti
- Ne viskas, kas įmanoma CSS'e, yra įmanoma `inline styles`



# STILIAI - PAVYZDYS

```
var styles = {
  container: { background: 'red' },
  greetingText: { color: 'green' }
};

class Component extends React.Component {
  render() {
    return (
      <div style={styles.container}>
        <p style={styles.greetingText}>Tekstas yra toks</p>
      </div>
    );
  }
}
```



# NAUJO KOMPONENTO ŠABLONAS

```
class Component extends React.Component {  
    render() {  
        return (  
            <div>  
                <!-- Component view -->  
            </div>  
        );  
    }  
}  
  
Component.propTypes = {  
    // Properties JSON  
};
```



# ŠAKNINIO KOMPONENTO NUPIEŠIMAS

```
ReactDOM.render(  
  <Component prop1={prop1Value} />,  
  document.getElementById('root')  
) ;
```

- Dažniausiai būna vienoje vietoje aplikacijoje
- Vaikiniai komponentai automatiškai nusipiešia



## UŽDUOTIS 5 - PIRMIEJI KOMPONENTAI #1

- Užduočiai turėtų užtekti 2 failų sukūrimo - index.html ir app.js
- Galite pasiimti bet kokį paveikslėlį iš interneto
- Norint nurodyti CSS klasę, vietoj atributo class rašome className
- Sukurkite statinį produktų sąrašo vaizdą



## UŽDUOTIS 5 - PIRMIEJI KOMPONENTAI #2

- Sukurkite 2 komponentus:
  - ProductCardComponent - mokantis piešti vieną produkto kortelę
  - ProductListComponent - mokantis piešti daug produktų kortelių
- Vienas produktas sąraše: paveikslėlis, pavadinimas, kaina, mygtukas į detales
- Užpildykite vaizdą testiniais duomenimis
- Kortelėms galite naudoti

<https://getbootstrap.com/docs/4.1/components/card/>



# KAIP ĮGALINTI IMPORT'US ?

- vis tik norime būtų modernūs - naudoti ES6 import
  - kitiems JS moduliams, stiliams, paveiksliukams
- reiktų modulių administratoriaus
- reiktų prijungti pvz. RequireJS patiemis, jei norėtumėm CommonJS formato..
  - bet tam reiktų loaderių..
    - o tada reiktų modulių administratoriaus..
    - be to, per script naudojanas babel negali matyti failų sistemos, reiktų kitaip prijungti babel..
- .. ir t.t.

**STOP!**



## UŽDUOTIS 6

- Pirmąsias užduotis padarysime, kad žinotume, kaip tai veikia, bet daugiau patys link ir script neberašysime
- Toliau naudosime modernius įrankius
  - es2015, npm, babel, webpack
- CSS ir paveiksliukus importuosime su import

```
import './App.css'; // iš src/ katalogo
import pav from './../public/favicon.ico';
var style = { backgroundImage: 'url(' + pav + ')', width: 100, height: 100 };
var styleAlt = { backgroundImage: `url(${pav})`, width: 100, height: 100 };
var jsx = (<img src={favicon} className="klase-is-app-css"/>);
// arba tiesiog į CSS įdedam background-image: url./logo.png);
```



# UŽDUOTIS 6 - BIBLIOTEKOS PER NPM

- Kaip pasileisti

```
$ apt-get install npm nodejs
$ npm install -g create-react-app
$ create-react-app hello-world
# Pridedam bootstrap priklausomybę į package.json dependencies
# "bootstrap": "4.1.3",
# "jquery": "1.9.1",
# "popper.js": "1.14.3"
$ npm install
$ npm run build
$ npm start
```

- "Sugadinti" projektą, kad atsirastų griaučiai:

```
$ npm run eject
```

- Pastaba: atgal grįžti nepavyks, tai tik TESTAS, ir neskirta realiam projektui

## UŽDUOTIS 6 - BIBLIOTEKOS PER NPM

- Windows ypatingai svarbu versijos

```
$ node --version // v8.10.0  
$ npm --version // 3.5.2  
$ create-react-app --version // 2.1.1
```

- NPM dar reikalauja ir windows build tools / linux build-essential

```
$ npm install -g windows-build-tools // windows  
$ apt install build-essential // linux
```

- kurios node ir npm versijos jums veikia Windows?
- taip pat gali tekti leisti iš Git Bash. Ir įsidėti node/npm katalogą į environment variables rankiniu būdu



# PATIKRINTI AR VEIKIA ES2015/ES6 IR BABEL'IS

## Modulis/Modulis.js - kataloge Modulis failas Modulis.js

```
class Polygon {  
    constructor(height=2, width=3) {  
        this.height = height;  
        this.width = width;  
    }  
    get area() {  
        return this.calcArea()  
    }  
    calcArea() {  
        return this.height * this.width;  
    }  
}  
export default Polygon;
```

## index.js

```
import Polygon from './Modulis/Modulis';  
console.log(new Polygon().calcArea());
```



## PATIKRINTI, AR VEIKIA JEST TESTAI

- npm test paleidžia App.test.js per Jest

```
import Polygon from './Modulis/Modulis';
it('calculates area correctly', () => {
    expect(new Polygon().calcArea()).toEqual(6);
});
```

- <https://github.com/facebookincubator/create-react-app/blob/master/packages/react-scripts/template/README.md#running-tests>



# PATIKRINTI AR VEIKIA BOOTSTRAP

## index.js

```
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
```

## app.js

```
<p><button className="btn btn-primary" role="button">Reload</button><
```



# KITOJE PASKAITOJE

Javascript moduliai. Arrow funkcijos. Map/filter/Reduce.  
Kolekcijos

# IŠ PRAEITOS PASKAITOS

- Jeigu atrodo, kad NIEKO nesuprantante
  - eikite po vieną skraidrę, darykite copy/paste, sekite instrukcijas, skaitykite, kas parašyta
    - jeigu ir tada neaišku - klauskite. Drąsiai!
- Jeigu atrodo, kad VISKAS suprantama (lyg ir)
  - darykite tą patį
  - atrasite, kad vis tik kažkas nesigauna
- Supratimas ateis, kai viską padarysite PATYS
- Kuo daugiau klausinėsite, tuo daugiau pateiksiu atsakymų

# IŠ PRAEITOS PASKAITOS

- Norint panaudoti PropTypes su naujausiu React 16, reikalinga

```
import PropTypes from 'prop-types';
```

- ir į package.json įsidėti

```
"prop-types": "^15.6.2"
```

- kaip teisingai pastebėjote, ES5 React 15.x turi `React.createClass()`, o React 16.x reikalinga `create-react-class` biblioteka ir turi `createReactClass()`
  - mes naudosime ES6 React 16.x `React.Component`



# IŠ PRAEITOS PASKAITOS

- Naujo komponento šablonas su PropTypes už klasės

```
import PropTypes from 'prop-types';

class Component extends React.Component {
  render() {
    return (
      <div>
        <!-- Component view -->
      </div>
    );
  }
}

Component.propTypes = {
  // Properties JSON
};
```



# IŠ PRAEITOS PASKAITOS

- Naujo komponento šablonas su PropTypes už klasėje

```
import PropTypes from 'prop-types';

class Component extends React.Component {
    static propTypes = {
        // Properties JSON
    }

    render() {
        return (
            <div>
                <!-- Component view -->
            </div>
        );
    }
}
```



# IŠ PRAEITOS PASKAITOS

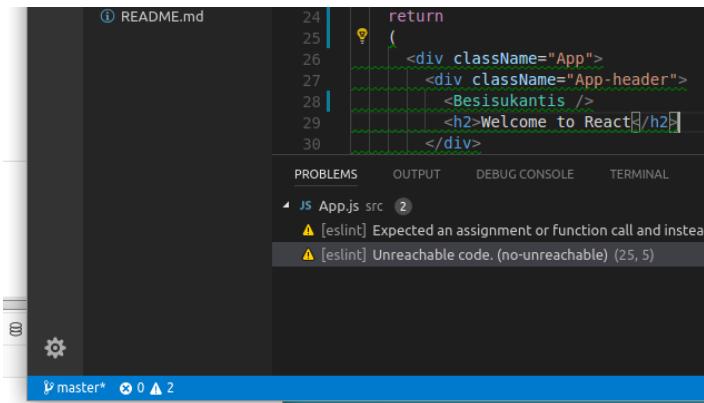
- ESLint - suponuoja taisykles, kaip reikia rašyti kodą
  - jei klaidose matote žodį LINT tai galbūt nepadėjote kur nors kabliataškio,
  - gal negalima iš naujos eilutės naudoti JSX skliaustelio
  - gal importuotą biblioteką reikia būtinai panaudoti arba neimportuoti
  - susikurtą kintamąjį reikia būtinai panaudoti



# ESLINT

- Pavyzdys kaip sukonfigūruoti VSCode ESLint extension:
  - Ctrl+Shift+X , suieškoti ESLint, install ir reload
  - būtent šiam pluginui reikia į package.json įdėti:

```
"eslintConfig": {  
    "extends": "react-app"  
}
```



# IŠ PRAEITOS PASKAITOS

- turite parašyti kodą patys
  - kitu atveju tiesiog bus per daug medžiagos, kad ją būtų įmanoma išmokti nerašius kodo





# AKADEMIJA.IT

INFOBALT IR TECH CITY

## JAVASCRIPT MODULIAI. ARROW/MAP/FILTER/REDUCE/KOLEKCIJOS

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# TURINYS

- Javascript moduliai
- Map/filter/Reduce
- Arrow funkcijos
- Kolekcijos



# JAVASCRIPT MODULIAI

# NPM - NODE PACKAGE MANAGER

- npm projektas - package.json
  - name
  - dependencies
  - devDependencies
  - scripts
- src
- public
- node\_modules



# NPM - NODE PACKAGE MANAGER

## package.json

```
{  
  "name": "hello-world",  
  "version": "0.1.0",  
  "private": true,  
  "dependencies": {  
    "react": "^16.6.3",  
    "react-dom": "^16.6.3",  
    "react-scripts": "2.1.1",  
    "prop-types": "^15.6.2",  
    "bootstrap": "4.1.3"  
  },  
  "devDependencies": {},  
  "scripts": {  
    "start": "react-scripts start",  
    "build": "react-scripts build",  
    "test": "react-scripts test --env=jsdom",  
    "eject": "react-scripts eject"  
  }  
}
```



# VERSIJŲ FIKSAVIMAS

- užfiksuoti versijas, kad nereiktų migruoti kodo, vietoj

```
"dependencies": {  
    "react": "^16.6.3",  
    "react-dom": "^16.6.3",  
    "react-scripts": "2.1.1",  
    "prop-types": "^15.6.2",  
    "bootstrap": "4.1.3"  
}
```

- naudoti (nors ^ ir reiškia compatible)

```
"dependencies": {  
    "react": "16.6.3",  
    "react-dom": "16.6.3",  
    "react-scripts": "2.1.1",  
    "prop-types": "15.6.2",  
    "bootstrap": "4.1.3"  
}
```



# NPM - NODE PACKAGE MANAGER

- pagrindinės npm komandos kurias naudosime:

```
$ npm install
    // dirba su package.json
    // https://docs.npmjs.com/cli/install
$ npm install <paketas>
$ npm install -g <paketas>
$ npm uninstall
$ npm install <paketas>@<versija>
$ npm help <komanda>
$ npm run <scripts-komanda>
$ npm run start // = npm start
```



# KLASE

```
class Polygon {  
    constructor(height=2, width=3) {  
        this.height = height;  
        this.width = width;  
    }  
    get area() {  
        return this.calcArea()  
    }  
    calcArea() {  
        return this.height * this.width;  
    }  
}  
  
var poly = new Polygon; // galima be skliaustų  
console.log(poly.calcArea());  
console.log(poly.area); // nėra skliaustų  
polygon.area = 3; // negalime - reiktu sukurti set area() metoda
```



# NAUJAS MODULIS

Sukuriamas src kataloge Modulis/Modulis.js

```
class Polygon { /* ... */ }
export var P1 = Polygon;
export var P2 = Polygon;
export default Polygon;
```

Babel'is konvertuoja į CommonJS formatą, pvz.:

```
var Polygon = function Polygon() { /* ... */ }
module.exports = Polygon;
module.exports.P1 = Polygon;
module.exports.P2 = Polygon;
```



# BABEL KOMPIILIATORIUS

- Konvertuoja JS kodą iš vieno formato į kitą
- Dažniausiai naudojamas konvertuoti iš ES2015 į ES5 formatą
- Arba TypeScript kodą į ES2015, o tada į ES5 ir t.t.
- Tiesiai nenaudosim; galim paleisti iš node\_modules

```
$ npm install --save-dev @babel/core @babel/cli @babel/preset-env  
$ node ./node_modules/.bin/babel src/babel-test.js  
--out-file compiled.js --presets=@babel/env
```

- Arba <https://es6console.com/>



# BABEL

Konvertuoja src/babel-test.js iš ES2015

```
class Person {}  
var dave = new Person
```

i compiled.js

```
"use strict";  
function _classCallCheck(instance, Constructor) {  
if (!(instance instanceof Constructor)) {  
    throw new TypeError("Cannot call a class as a function"); } }  
var Person = function Person() {  
    _classCallCheck(this, Person);  
};  
var dave = new Person();
```



# MODULIU UŽKROVIMAS

## index.js

```
import {P} from './Modulis/Modulis';
import {P as Pavadinimas} from './Modulis/Modulis';
import Polygon from './Modulis/Modulis';
import Polygon, {P} from './Modulis/Modulis';
```

Iš tikrujų babel'is konvertuoja iš ES2015

```
import express from 'express';
```

į CommonJS require formą

```
var express = require( 'express' );
```

# MODULIAI - GERA PRAKTIKA

- Atskiras komponentas - atskiram modulyje
- Atskiras modulis - savo kataloge
- Panašu į java packages ir import
- Visada export default
  - pagrindinis funkcionalumas
  - pagrindinė klasė
- Papildomos klasės, interfeisai ar konstantos
  - export var

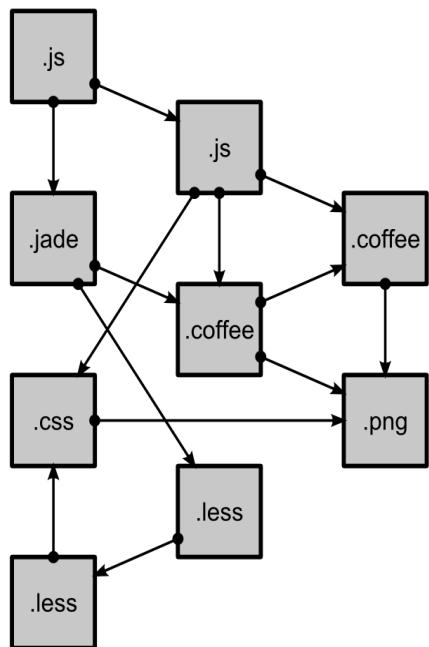


# MODULIŲ PAKAVIMAS - WEBPACK

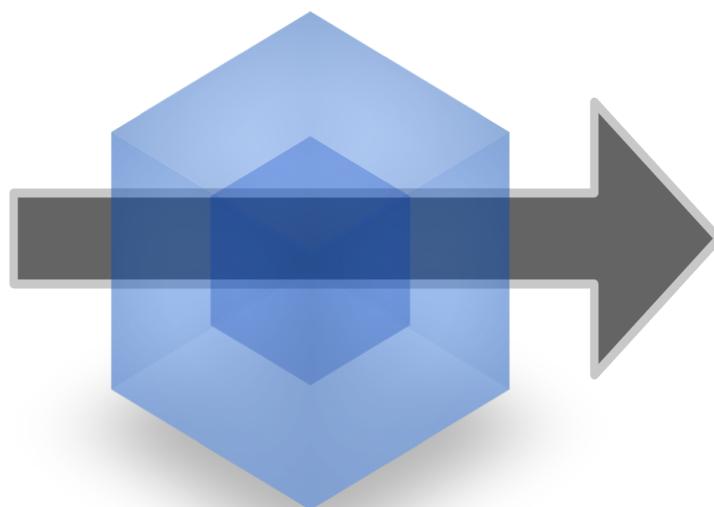
- Kas atliks visus tuos konvertavimus?
- Kas pasirūpins, kad viskas atsidurtų reikiamuose kataloguose serveryje?
- Kaip naršyklė supras mūsų "advanced" technologijas?
  - nesupras. Bent jau ne visas ir ne šiandien
- Webpack leidžia apjungti visas technologijas
  - naudojama viskas advanced, latest, etc. ir paduodama webpack'ui
  - webpack sukuria gražų ES5 kodą ir failų struktūrą, kurią supranta naršyklė



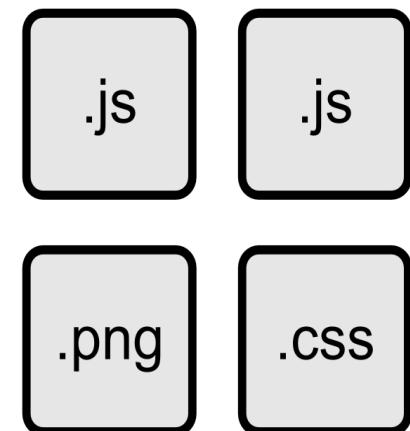
# KAS YRA WEBPACK



modules  
with dependencies



webpack  
MODULE BUNDLER



static  
assets

# NAUJO KOMPONENTO ŠABLONAS

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';

class Komponentas extends Component {
  render() {
    return (
      <div>
        {/* Component view */}
      </div>
    );
  }
}
Komponentas.defaultProps = { /* Properties JSON */ };
Komponentas.propTypes = { /* Properties JSON */ };
```



## UŽDUOTIS 1

- Konvertuoti praėitoje paskaitoje padarytus ProductCardComponent ir ProductListComponent į naujają projekto struktūrą
  - react-create-app struktūra
  - Card ir List komponentai atskiruose kataloguose ir failuose
  - App komponentas - irgi savo modulyje



# ATVAIZDIS, FILTRAS IR REDUKCIJA



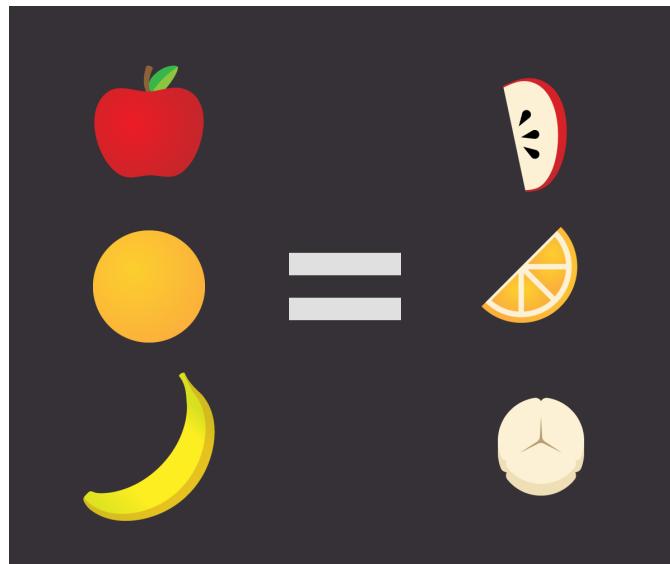
# ATVAIZDIS, FILTRAS IR REDUKCIJA



- Atvaizdis (map), filtras (Filter) ir redukcija (Reduce) naudojami elementų masyvą paversti į kažką kitą



# ATVAIZDIS (MAP)



- Turime masyvą elementų ir norime transformuoti kiekvieną iš elementų.
- Rezultatas būtų naujas tokio paties dydžio masyvas, kuriame yra pakeisti elementai



# ATVAIZDIS (MAP)



```
var people = [ {  
    name: 'Antanas Atanaitis',  
    drives: 'Car',  
    team: 'Studentai'  
}, {  
    name: 'Petras Petraitis',  
    drives: 'Truck',  
    team: 'Moksleiviai'  
}, {  
    name: 'Vilkas Vilkaitis',  
    drives: 'Formula 1',  
    team: 'Studentai'  
}, {  
    name: 'Vilija Vilimaitė',  
    drives: 'Car',  
    team: 'Moksleiviai' } ];
```



# ATVAIZDIS (MAP)

```
map(callback(item));  
map(callback(item[, index]));
```

Su funkcija:

```
function getDrives(pupil) {  
    return pupil.drives;  
}  
var automobiles = people.map(getDrives);
```



## ATVAIZDIS (MAP)

Su anonimine (be vardo) funkcija:

```
automobiles = people.map(function(pupil) {  
    return pupil.drives;  
});  
  
console.log(automobiles);
```



# ATVAIZDIS (MAP)

Su arrow funkcija:

```
automobiles = people.map(pupil => pupil.drives);  
console.log(automobiles);
```



# ARROW FUNKCIJA

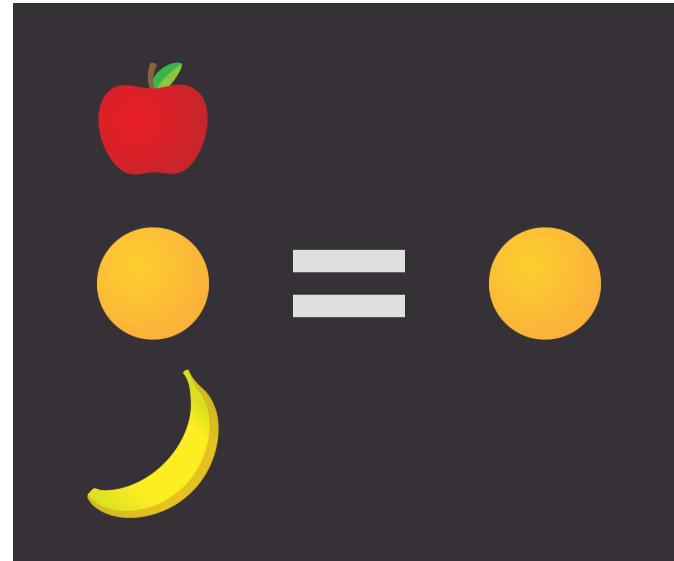
- Storosios rodyklės funkcija

```
param1 => param1;
param1 => { return param1; }
param1 => param1 * param1;
(param1, param2) => param1 + param2;
param1 => ({atributas: param1});
() => false;
var f = () => false;
var f = () => ({false}); // https://es6console.com/
```

- Svarbu: arrow funkcija išlaiko leksikologinį kontekstą - jos viduje veikia išorinio konteksto nuoroda `this`
  - todėl nebereikia "hack'ų" `bind(this)` ir `var self = this`



# FILTRAS (FILTER)



- Išrinkti iš masyvo elementus. Rezultatas yra naujas masyvas su visais elementais, išskyrus išfiltruotus
- Naujo masyvo ilgis tokis pats kaip senojo jeigu niekas neišfiltruojama arba trumpesnis

# FILTRAS (FILTER)

```
filter(callback(element)) ;  
filter(callback(element[, index])) ;
```

pvz.:

```
var studentai = people.filter(pupil => pupil.team === 'Studentai') ;  
console.log(studentai) ;
```



# ATVAIZDIS IR FILTRAS

## be formatavimo

```
studentai = people.filter(pupil => pupil.team === 'Studentai').map(pupil =>  
    console.log(studentai);
```



# ATVAIZDIS IR FILTRAS

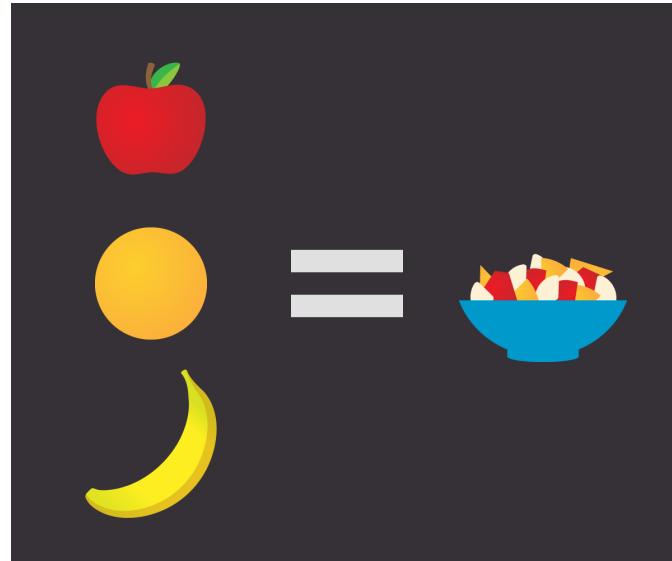
## su formatavimu

```
studentai = people
    .filter(pupil => pupil.team === 'Studentai')
    .map(pupil => pupil.name);

console.log(studentai);
```



# REDUKCIJA (REDUCE)



- Turint elementų masyvą galima apskaičiuoti naują reikšmę pereinant per kiekvienu iš elementų
- Rezultatas gali būti bet kas: kitas masyvas, naujas objektas, true/false reikšmė ir pan.



# REDUKCIJA (REDUCE)

```
reduce(callback(accumulator, currentValue)) ;  
reduce(callback(accumulator, currentValue[, currentIndex]) [, initialValue])
```

PVZ.:

```
var names = people.reduce(function(sum, pupil) {  
    return sum + ', ' + pupil.name;  
}) ;  
  
console.log(names) ;
```



# REDUKCIJA (REDUCE)

```
names = people.reduce(function(sum, pupil, currentIndex) {  
    if (currentIndex === 0) {  
        return sum + '' + pupil.name;  
    } else {  
        return sum + ', ' + pupil.name;  
    }  
, "Vardai: ");  
  
console.log(names);
```



# REDUKCIJA (REDUCE)

```
names = people.reduce((sum, pupil) => sum + ', ' + pupil.name);  
console.log(names);
```



# REDUKCIJA (REDUCE)

```
names = people
    .reduce((sum, pupil, currentIndex) =>
        sum + (currentIndex > 0 ? ', ' : '') + pupil.name,
        "Vardai: ");
console.log(names);
```



# FILTRAS, ATVAIZDIS IR REDUKCIJA

```
studentai = people
    .filter(pupil => pupil.team === 'Studentai')
    .map(pupil => pupil.name)
    .reduce((vardai, vardas) => vardai + ' ' + vardas);

console.log("Studentai:", studentai);
```



## UŽDUOTIS 2

- Sukuriame prekių sąrašo masyvą
- Elementai - produktai
- Produktą apibūdinantys laukai: title, imageUrl, description, price, quantity
- Prikuriame produktų su kaina nuo 1 eur iki 100 eur
- Sukuriam vaizdinį be imageUrl ir be description
- Išfiltruojam prekes, kurių kaina mažesnė negu 10
- Redukuojame prekių masyvą į vieną skaičių - visų prekių kainų, padaugintų iš kiekio, sumą
  - visai kaip krepšelio galutinė suma



# ES6 KOLEKCIJOS



## ES5 KOLEKCIJOS

- Masyvai
- Objektai
- Simbolių eilutės (string)
- Pseudo-masyvas - argumentai funkcijoje



## ES5 PROBLE莫斯

- Masyve elementai neunikalūs
- Objektų raktai (key) verčiami į string
- turi papildomų savybių, tokius kaip `toString`, `proto`



## ES6 KOLEKCIJOS

- Aibė (Set)
- Žemėlapis (Map)
- WeakMap (silpnas)



## AIBĖ (SET)

- Unikalios reikšmės (tiek primityvios, tiek objektai)
- Išsaugo tvarką

```
const aibe = new Set(); // Set [ ]  
aibe.add('vardas'); // Set [ "vardas" ]  
aibe.add('miestas'); // Set [ "vardas", "miestas" ]  
aibe.add('miestas'); // neįdės - dublikatas  
aibe.has('miestas'); // true  
aibe.delete('vardas'); // Set [ "miestas" ]  
aibe.size; // 1  
aibe.clear(); // Set [ ]
```



# UNIKALUS MASYVAS

- ES3 - per sunku, tai reik naudot biblioteką, pvz. Lodash

```
_ .uniq([2, 1, 2]);
```

- ES5 - filtruoti

```
[2, 1, 2].filter(function(elem, pos, arr) {  
    return arr.indexOf(elem) === pos;  
})
```

- ES6 - Set

```
Array.from(new Set([2, 1, 2]));  
[... new Set([2, 1, 2])];  
// spread operator ... turns iterables to function arguments
```



# UNIKALŪS OBJEKTAI

Nėra hash funkcijos, todėl

```
const aibe = new Set();
const obj = {a:1,b:2};
aibe.add(obj);
aibe.add({a:1,b:2}); // ok - kitas objektas
console.log(aibe); // 2 objektai
```



## ŽEMĖLAPIS (MAP)

- Kolekcijos raktas + reikšmė (key+value)
- Objektai neverčiami į string
- Naudojam, kai iš anksto nežinom raktų

```
const zem = new Map();
zem.set(1, "vardas");
zem.get(1);
zem.size; // 1
zem.has(1); // true
zem.delete(1);
zem.clear();
```



## MAP/SET ELEMENTAI

- Per elementus galima pereiti naudojant iteratorių arba su:
  - `for..of`
  - `forEach`

```
for (let item of aibe) {  
    console.log(item);  
}  
  
aibe.forEach(item => console.log(item));
```



## SILPNAS ŽEMĖLAPIS (WEAKMAP)

- Raktai - TIK objektai
- Užima mažiau vietas atmintyje
- Iteruoti per elementus negalima
- Elementai išvalomi (surenkami GarbageCollector'iaus) kai jų niekas nenaudoja
- Išvalyti viso (clear) negalima



## UŽDUOTIS 3

- Sukurti naują klasę Produktas su produkto atributais
- Prekių masyvo elementus map'inti į naują masyvą, kur elementai būtų new Produktas(...)
- Gautą produktų sąrašą paversti į objektų aibę
- Gautą produktų sąrašą paversti į objektų žemėlapį
- pasinaudojant Array.from iš objektų map'o ir vėl gauti masyvą, kurį suredukuoti iki vienos eilutės, kurioje būtų prekės title ir kaina
  - Obuolys (1 eur), Samsung (100 eur)



# KITOJE PASKAITOJE

React komponentų būsenos, gyvavimo ciklas. Navigacija.  
Darbas su serveriu



# LIVE CODING SESIJA

## KONVERTUOJAM HTML Į REACT



AKADEMIJA.LT  
INFOBALT IR TECH CITY



**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

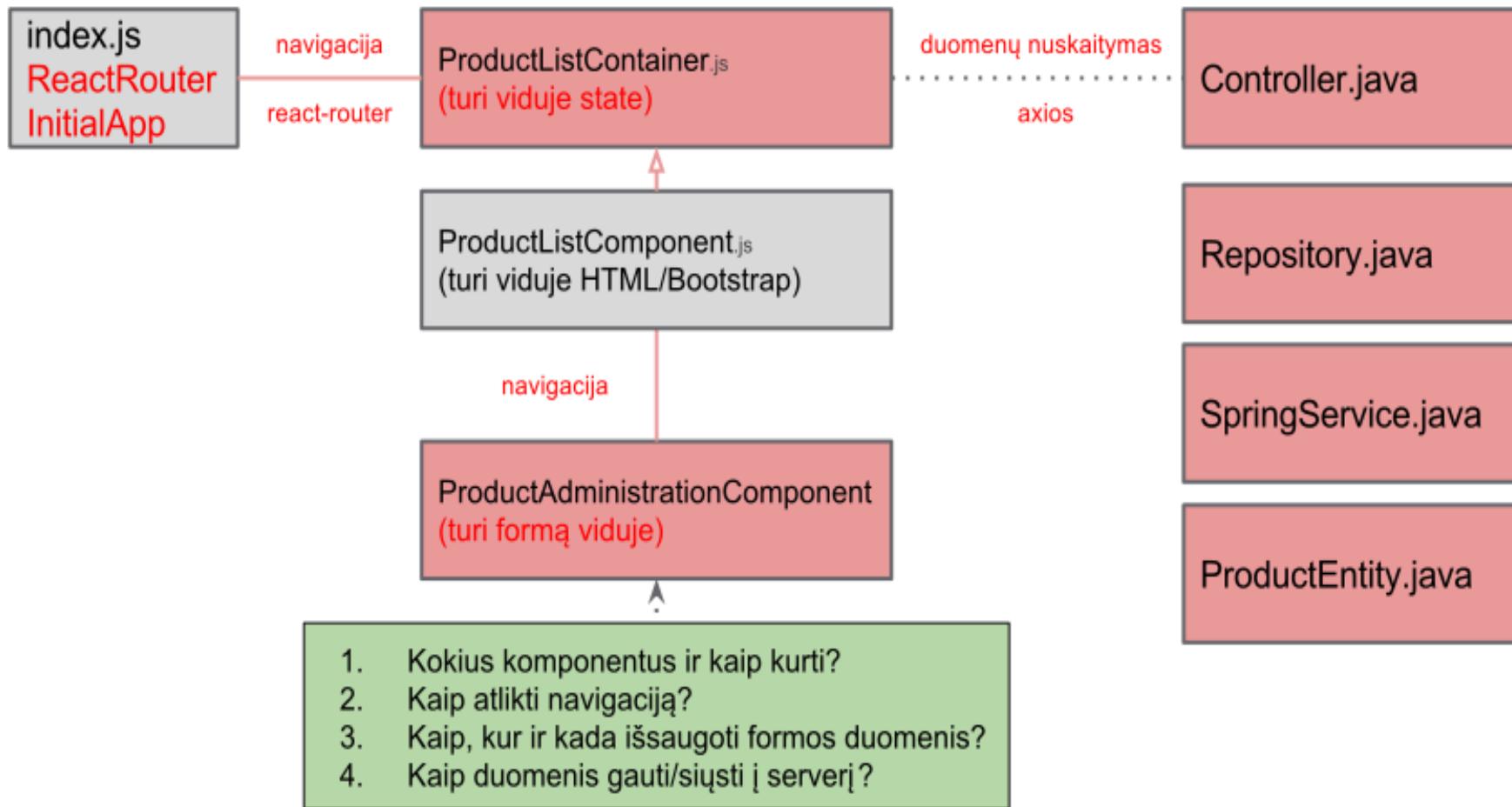
# **REACT KOMPONENTAI. NAVIGACIJA. GLOBALŪS KINTAMIEJI. DARBAS SU SERVERIU**

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# KĄ JAU MOKAME IR KO DAR NE



# TURINYS

- Komponentų būsenos ir gyvavimo ciklas
- Puslapio navigacija
- Globalūs kintamieji
- Darbas su serveriu



# REACT KOMPONENTŲ BŪSENOS

## KOMPONENTO BŪSENA

- Iki šiol komponentai neturėjo būsenos
- Vienintelis būdas atnaujinti komponentą, buvo ji periešti naudojant render() funkciją



# BŪSENOS/DUOMENŲ VALDYMAS

- Viduje
  - Backbone: modeliai/kolekcijos
  - Ember: Ember Data
  - Angular 1: servisai/valdikliai-controllers
- React - lokali komponento būsena
  - "Flux" paternas (duomenys į vieną pusę)
  - Redux (vienas būsenos medis, nekeičiami atnaujinimai)
  - MobX (priklausomybės per observables)
- Angular 2
  - servisai/valdikliai
  - Redux; Observables

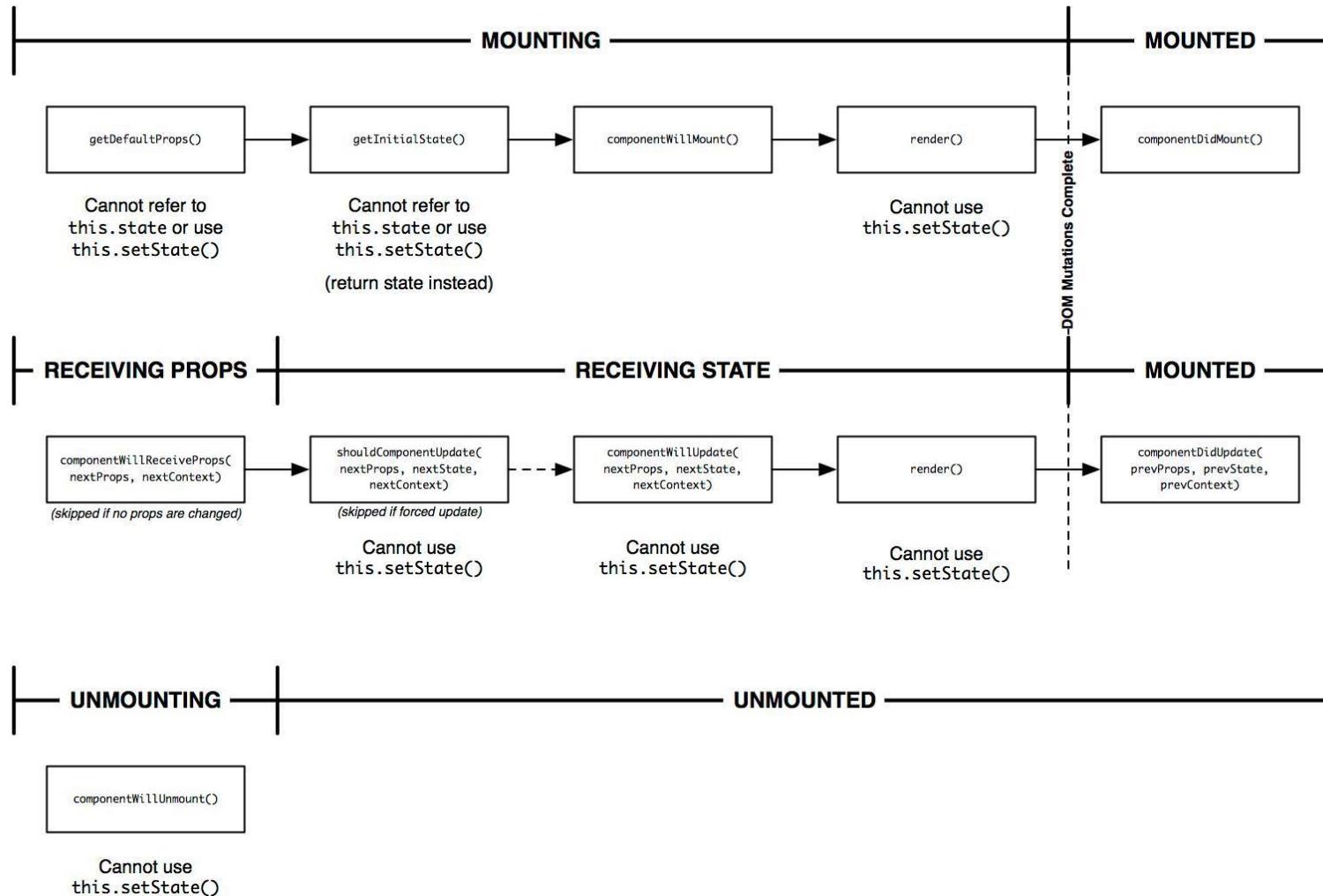


## KOMPONENTO BŪSENA

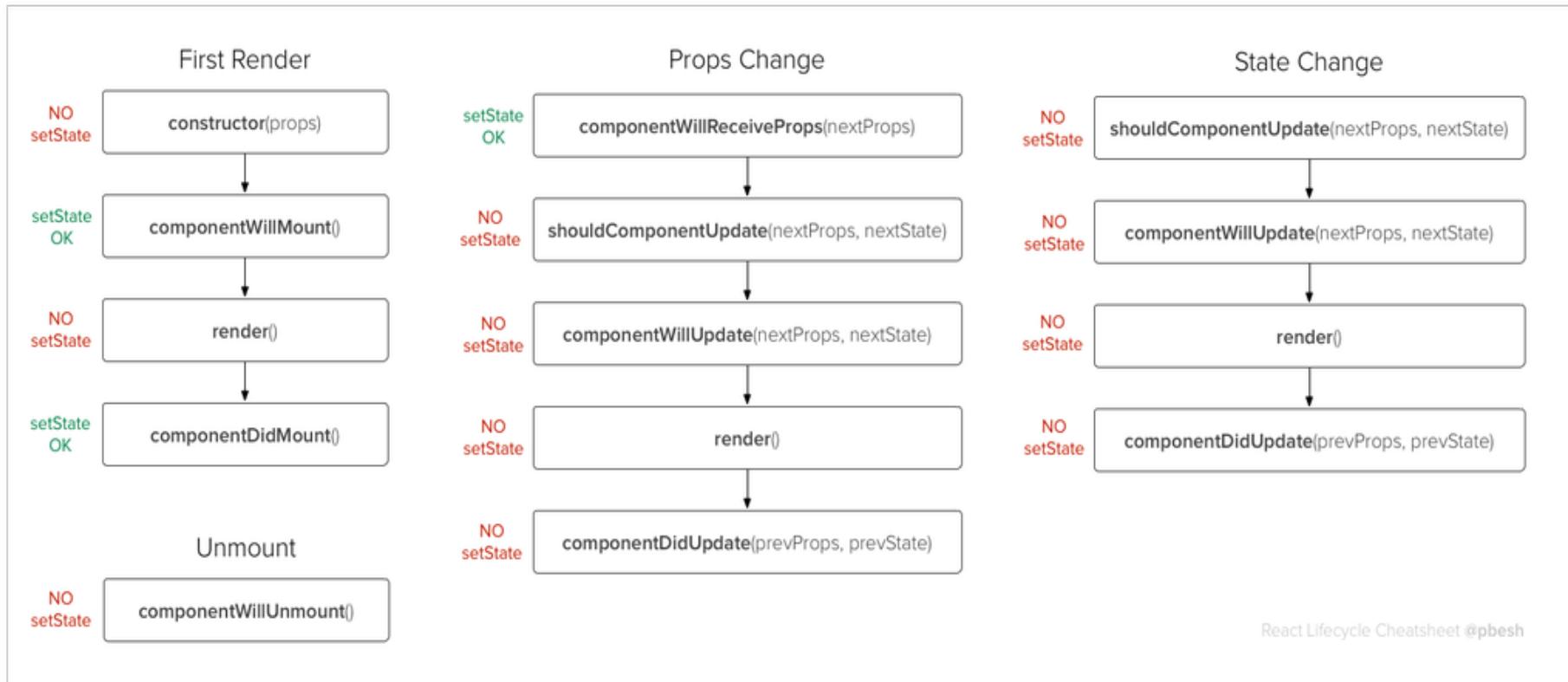
- Būsena saugoma kintamajame `this.state`
- Būsena atnaujinama kviečiant komponento funkciją `this.setState(newState)`
- React'as sulieja esamą būsenos objektą su `NewState` būsenos objektu
- React'as automatiškai perpiešia komponentą, kai yra pakviečiama `setState` funkcija
- Dažniausiai būsena keičiama reaguojant į pelės paspaudimo arba formos lauko keitimo įvykius



# REACT KOMPONENTO GYVAVIMO CIKLAS ES5



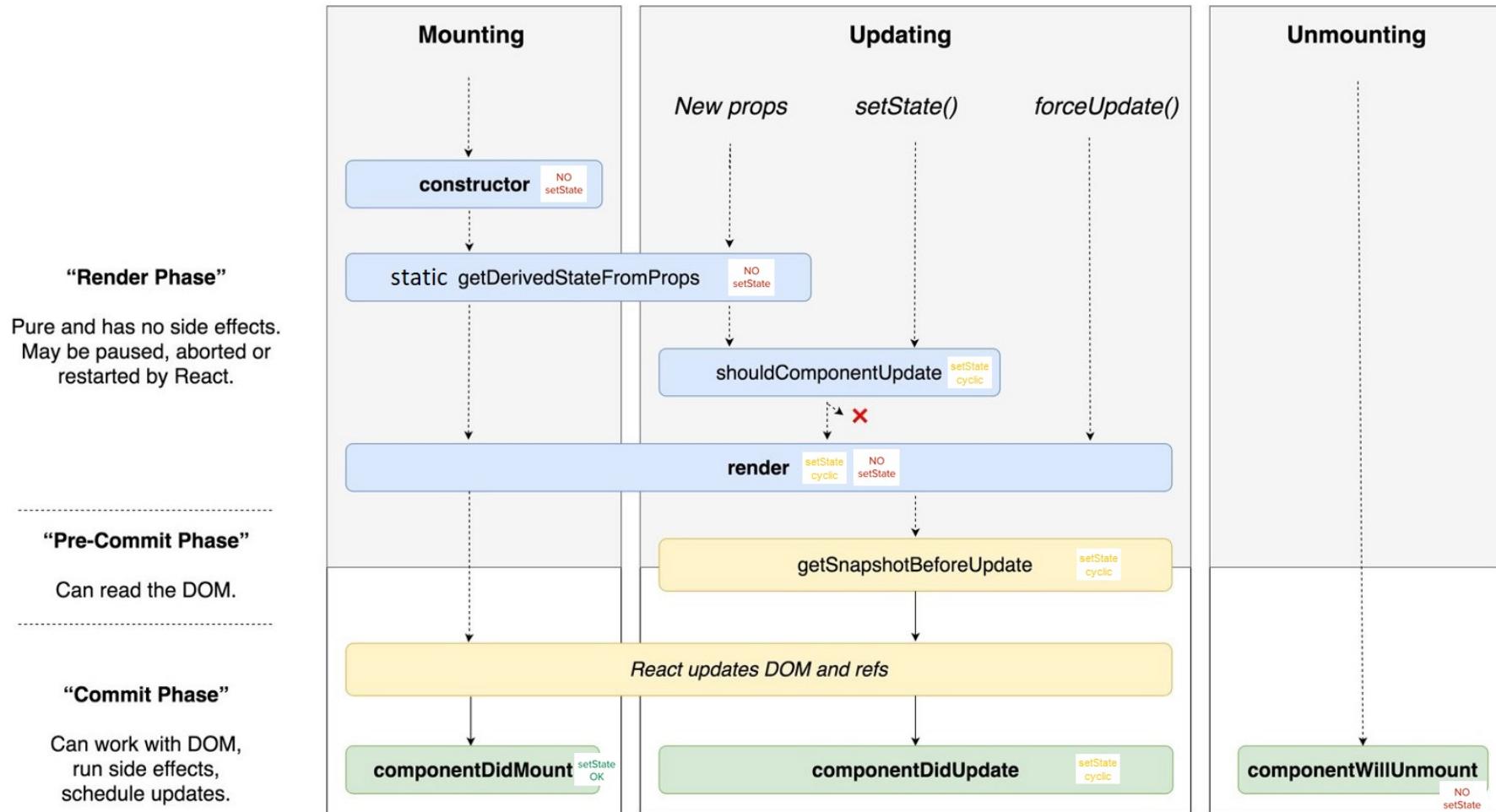
# REACT KOMPONENTO GYVAVIMO CIKLAS ES6



React Lifecycle Cheatsheet @pbesh



# REACT 16+ KOMPONENTO GYVAVIMO CIKLAS ES6



## UŽDUOTIS 1 - GYVAVIMO CIKLAS

- Paduoti props į konstruktorių ir sukurti pradinį `this.state`
- Perrašyti keletą metodų
  - bent jau `getDerivedStateFromProps`, `render`, `componentDidMount`
- Pastebėti, kas po ko ir kiek kartų kviečiasi
  - `console.log("getDerivedStateFromProps");`



# DAŽNIAUSIAI NAUDOJAMOS BŪSENOS FUNKCIJOS

- `this.state = {}`
  - turi nustatyti objektą, kuris bus pradinė būsenos reikšmė
  - naudojamas konstruktoriuje
- `render()`
  - turi grąžinti komponento html'ą
  - pasikeitus būsenai, virtualus DOM palygins/apskaičiuos, kas keitėsi, ir iškvies render



## DAŽNIAUSIAI NAUDOJAMOS BŪSENOS FUNKCIJOS

- `componentDidMount/componentWillUnmount`
  - skirtas sukurti/atšaukti globalius objektus, pvz `setInterval` ar `clearInterval`
  - pakvesti serverj užkraunant duomenis ir perduodant juos `setState`
  - post-constructor/render ir pre-destructor



## PASPAUDIMO ĮVYKIAI

- React'js attributas onClick leidžia įvykdyti funkciją paspaudus mygtuką
- onClick attributas gali būti net tik ant mygtuko, bet ir ant bet koks elemento.



# PAVYZDYS

```
class IncreasingButtonComponent extends Component {  
    constructor() { super(); this.state = { count: 0 }; }  
    handleClick = (event) => {  
        event.preventDefault(); // event.stopPropagation();  
        this.setState({ count: this.state.count + 1});  
    }  
    render() {  
        return (  
            <div>  
                {this.state.count} &nbsp;  
                <button className="btn btn-default"  
                    onClick={this.handleClick}>Increase</button>  
            </div>  
        );  
    }  
}
```



## PAVYZDYS - PAAIŠKINIMAS

- count: 0 - paspaudimų skaičius bus saugomas būsenos count lauke. Pradžioje turime 0 paspaudimų.
- &nbsp; - taip html'e rašomas tarpas
- turime arrow funkciją handleClick, kuri naudoja setState ir padidina count skaitliuką vienetu
- event.preventDefault() blokuoja numatytają onClick funkciją
- handleClick funkcija yra pakviečiama paspaudus mygtuką. Tai padaroma naudojant specialų ReactJs onClick atributą



## UŽDUOTIS 2 - SUSINAIKINIMO SKAITLIUKAS

- Sukurkite komponentą SelfDestructTimerComponent
- Atidarius puslapį, skaitliukas turėtų pradėti skaičiuoti nuo 42 sekundžių iki 0
- Pasiekus 0, komponento fonas turėtų paraudonuoti



## UŽDUOTIS 2 - UŽUOMINOS

- `setInterval` - <https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers/setInterval>
- `clearInterval` - <https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers/clearInterval>
- nepamirškite išvalyti `setInterval`
- background CSS atributas



# KOMPONENTO ŠABLONAS

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';

class Komponentas extends Component {
  constructor(props, context) {
    super(props, context);
    this.state = { };
  }
  componentDidMount() {
    // užkrauti duomenis iš serverio
    this.setState({ /* čia išsisaugome duomenis iš serverio */ })
  }
  render() {
    return (<div> {/* Component view */} </div>);
  }
}
```



## REACT FORMOS

- Formos React'e išsiskiria iš kitų elementų dėl to, kad kiekvienas laukas savyje turi būseną
- React'e būsena laikoma state lauke
- Formos elemento atributai onChange ir value naudojami sinchronizuoti elemeto būseną su state lauku
- onChange priima funkciją, kurią pakviečia pasikeitus formos lauko reikšmei
- value atspindi formos reikšmę
- jei norime pakeisti formos lauko reikšmę iš javascript'o, turime keisti state lauką, kuris yra value išraiška



# REACT PAVYZDYS

```
class NameForm extends Component {  
    constructor() { super(); this.state = { value: '' }; }  
    handleChange = (event) => this.setState({value: event.target.value})  
    handleSubmit = (event) => {  
        this.setState({value: 'reset after submit'});  
        event.preventDefault();  
    }  
    render() {  
        return (<form onSubmit={this.handleSubmit}>  
            Name ({this.state.value}):<br/>  
            <input type="text" value={this.state.value}  
                  onChange={this.handleChange} />  
            <input type="submit" value="Submit" />  
        </form>);  
    }  
}
```



## REACT PAVYZDYS - PAAIŠKINIMAS

- kaskart vedant reikšmę yra kviečiama handleChange funkcija, kuri naujai gautą reikšmę is event parametro perkelia į state.value
- paspaudus Submit nygtuką į console atspausdinama reikšmė ir pakeičiama į 'reset after submit'
- event.preventDefault() reikalingas tam, kad nebūtų vykdomas standartinis formos submit'as - formos duomenų siuntimas į serverį
- ateityje handleSubmit paskirtis būtų pakviesti serverį (REST Api) ir nusiųsti jam duomenis



## UŽDUOTIS 3 - PRODUKTO SUKŪRIMO FORMA

- sukurkite ProductAdministrationComponent
- Produktą apibūdinantys laukai: title, imageUrl, description, price, quantity
- Mygtukas Save
- Paspaudus Save išspausdinkite visą produkto informaciją
- Naudokite Bootstrap'o formas:  
<https://getbootstrap.com/docs/4.1/components/forms/>



# KOMPONENTAI BE BŪSENOS

- Stateless Functional Components
- Komponentai, kurie turi tik render funkciją
  - tai yra pats komponentas ir yra render funkcija
- Užrašomi trumpiau, kaip paprastos JS funkcijos:

```
var Komponentas = (props) => { // arba ({atributas, kitas}) => {
  var {atributas, kitas} = props; // destructuring assignment
  return (<div>{atributas} {kitas}</div>);
}
```

## Panaudojimas

```
<Komponentas atributas="reiksme" kitas="nieko"/>
```



## KOMPONENTŲ PASKIRTIS

- Dažniausia priimta turėti 2 tipų komponentus
  - Presentation (vaizdavimo)
  - Container (būsenos)
- React'o požiūriu tai yra tiesiog React'o komponentai
- Visada pora: ProductListComponent ir ProductListContainer



## PRESENTATION KOMPONENTAI

- turi tik render() funkciją
- yra Stateless Functional Component todėl gali būti apibrėžti kaip funkcija
- atsakingi tik už piešimą
- visa piešimui reikalinga informacija yra perduodama per props



## CONTAINER KOMPONENTAI

- turi render() funkciją kuri piešia Presentation componentą
  - daugiau nieko nepiešia
- gali turėti state ir kitus gyvavimo ciklo elementus
- atsakingas surinkti duomenis, registruoti click funkcijas ir valdyti būseną



# PRESENTATION KOMPONENTO PAVYZDYS

```
var SomeComponent = (props) => {
  return (
    <div>
      { /* A lot of html */}
    </div>
  );
}

SomeComponent.propTypes = {
  prop1: PropTypes.string.isRequired
};
```



# CONTAINER KOMPONENTO PAVYZDYS

```
class SomeContainer extends Component {  
  constructor(props) { super(props); this.state = {}; }  
  componentDidMount() {}  
  // Other functions  
  render() {  
    return <SomeComponent prop1={this.state.prop1} />  
  }  
})
```



# METODŲ ATSKYRIMAS Į CONTAINER'Į

- Per props galima perduoti nebūtinai laukų reikšmes, galima perduoti ir pačias funkcijas
  - pvz. perduoti metodą iš container komponento į presentation galima tiesiog per komponento atributą

```
class FormaContainer extends React.Component {  
  onReiksmeChange = (e) => this.setState({reiksme: e.target.value})  
  <..>  
  render() return <Forma onReiksmeChange={this.onReiksmeChange}>  
  
const Forma = ({onReiksmeChange}) => {  
  return (<form>  
    <input type="text" value={reiksme} onChange={onReiksmeChange}>  
  </form>);  
}
```

- visas pavyzdys: <https://jsfiddle.net/w2hz4o5n/>



## UŽDUOTIS 4 - KOMPONENTAS BE BŪSENO

- pakeisti create-react-app paveiksluką taip, kad jis suktysi tik užvedus pele
- sukirkite komponentą Besisukantis be būsenos
- komponentas turi gražinti logo paveiksluką su `className=""`
- prie `img` pridėkite `onMouseOver` ir `onMouseOut` atributus
- sukirkite arrow funkciją Besisukantis komponento viduje, kurių viena `event.target` nustato `className=""`, o kita `className="App-logo"`
- `img` pridėkite `style={{height : 70}}` dėl dydžio



# PUSLAPIO NAVIGACIJA

## PUSLAPIO NAVIGACIJA

- kolkas visi mūsų komponentai buvo viename puslapyje normalu, kad puslapio naudotojai norėtų naviguoti spausdami mygtukus, linkus, paveikslėlius ir t.t.
- šiam funkcionalumui įgyvendinti bus naudojama React Router biblioteka
- <https://github.com/reacttraining/react-router>



## KLIENTO PUSĖS NAVIGAVIMAS

- routing/navigavimas: URL adreso priskyrimas elgsenai, taip pat išskiriant duomenų parametrus iš URL
- istoriškai buvo serverio atsakomybė
- kuriant modernią aplikaciją navigavimas kliento puseje leidžia greičiau atnaujinti puslapį
- į package.json dependencies reikia pridėti:

```
"react-router": "^4.3.1",
"react-router-dom": "^4.3.1"
```



## NAVIGACIJOS KOMPONENTAI

- Navigacijos konfigūracijos
  - Switch
  - Redirect
  - Route
  - BrowserRouter
  - HashRouter
- Navigacijos: Link arba tiesiog history.push()



- Iš index.js importuojame Router

```
import { Switch, Redirect, Route } from 'react-router';
import { BrowserRouter, Link } from 'react-router-dom';
```

- komponentas AppContainer, kuriame bus navigacija

```
var AppContainer = (props) => {
  return (<div>
    <div>
      <Link to='/'>Home</Link> | &nbsp;
      <Link to='/products'>Products</Link> | &nbsp;
      <Link to={`/products/${127}`}>Product by no</Link> | &nb
      <Link to='/help'>help</Link> | &nbsp;
      <Link to='/non-existant'>Non Existant</Link>
    </div>
    {props.children}
  </div>);
};
```



- Komponentas, kurį rodysime neegzistuojančiam keliui

```
var NoMatch = (props) => {
    var goApp = () => props.history.push("/");
    return <div>Route did not match
        <button onClick={goApp}>Go Home</button></div>;
};
```

- Iš App.js esantį komponentą įsidedame mygtuką navigavimui į nuorodą /products

```
goProducts = () => this.props.history.push("products");
// o pati mygtuką kur nors į render() metodą
<p><button onClick={this.goProducts}
    className="btn btn-primary"
    role="button">
    Go to Products
</button></p>
```



- Šis komponentas skirtas pademonstruoti navigaciją (index.js)

```
var DemonstruotiNavigacija = (props) => {
  var goHome = () => props.history.push("/");
  return (
    <div>
      At route: {props.location.pathname}
      <button onClick={goHome}>Go Home</button>
      <pre>
        {JSON.stringify(props, null, 2)}
      </pre>
    </div>
  );
};
```



- Pakeisti ReactDOM.render iš <App/> į BrowserRouter:

```
ReactDOM.render( (
  <BrowserRouter>
    <AppContainer>
      <Switch>
        <Route exact path='/' component={App} />
        <Route path="/products/:id" component={DemonstruotiNavigacija} />
        <Route path="/products" component={DemonstruotiNavigacija} />
        <Route path="/help" component={DemonstruotiNavigacija} />
        <Route path="*" component={NoMatch} />
        <Route component={NoMatch} />
      </Switch>
    </AppContainer>
  </BrowserRouter>
), document.getElementById('root'));
```

- Dabar pradinis taškas bus BrowserRouter, o ne App



## PAAIŠKINIMAS

- Router'io konfigūracija prasideda eilute  
`<BrowserRouter>`
- path pasako koks keliais url'e atitiks kokį komponentą
  - Tarkime, kai url'e bus /products bus piešiamas Demonstruoti Navigacija
- path atributas gali būti šablonas
- /products/:id atitiks visus kelius tokius kaip products/1, products/new
- :id reikšmė bus patalpinta props.params.id



# BROWSERROUTER YRA ROUTER SU BROWSER HISTORY

- tai yra šis trumpesnis kodas

```
import { BrowserRouter } from 'react-router-dom'  
<BrowserRouter/>
```

- atlieka tą patį ką ir šis ilgesnis kodas

```
import { Router } from "react-router";  
import { createBrowserHistory } from "history";  
const history = createBrowserHistory(props);  
<Router history={history} />
```

- naudingas, jei reikia history perduoti toliau



# NAVIGACIJA SU HISTORY

- Navigacija į /products

```
history.push('/products')
```

arba

```
history.push({pathname: '/products', search: '?p=1', state: {}})
```

- Dokumentacija

<https://github.com/ReactTraining/history#navigation>



# NAVIGACIJA SU LINK

- Navigacija į /products

```
<Link to="/products">Go to Products</Link>
```

arba

```
<Link to={{ pathname: '/products', query: { p: '1' } }}>Go to Products</Link>
<Link to={`/products/${127}`}>Go to Products</Link>
```

- Dokumentacija <https://knowbody.github.io/react-router-docs/api/Link.html>



## UŽDUOTIS 5 - NAVIGACIJA #1

- Naudokite prieš tai užduotyje sukurtus komponentus
  - ProductListComponent ir ProductAdministrationComponent
- /ir/products turi rodyti produktų sąrašą (ProductListComponent)
- /admin/products/new - ProductAdministrationComponent
- /admin/products/:id - ProductAdministrationComponent



## UŽDUOTIS 5 - NAVIGACIJA #2

- papildykite ProductAdministrationComponent, kad jis rodytų priklausomai nuo kelio puslapio antraštė:
  - Kuriamas naujas produktas, jei url  
`/admin/products/new`
  - Atnaujinamas produktas 'id', jei url  
`/admin/products/:id`



# GLOBALŪS KINTAMIEJI



AKADEMIJA.LT  
INFOBALT IR TECH CITY

# GLOBALŪS KINTAMIEJI

- index.js nustatyti į props ir perduoti atributus toliau komponentams per props
  - **teisingas**, bet "užteršia" kodą
- variantas senuoju būdu (**nenaudotina**): susidėti servisus į var window
  - t.y. panaudoti globalų naršyklėje egzistuojantį kintamąjį var window
- React 15 **experimental** static .contextTypes (**deprecated**)
  - ji pakeitė **teisingas** React 16 Context API



# GLOBALŪS KINTAMIEJI

- React kiekvienam komponentui gali perduoti kontekstą, tada nereikia vis perduot per props. Konteksto sukūrimas

```
const ServicesContext = React.createContext(null);
```

- deklaravimas šakniniame komponente

```
<ServicesContext.Provider value={{userService: userService}}>
  <AppContainer/> <!-- konteksta gaus šis ir visi viduje --&gt;
&lt;/ServicesContext.Provider&gt;</pre>
```

- ten, kur reikia panaudoti

```
<ServicesContext.Consumer>
  {({userService}) => <span>{userService.name}</span>}
</ServicesContext.Consumer>
```

- dokumentacija <https://reactjs.org/docs/context.html>



# REACT CONTEXT

- Iš dokumentacijos - kodėl nenaudoti context

*Context is primarily used when some data needs to be accessible by many components at different nesting levels.*

*Apply it sparingly because it makes component reuse more difficult. Context lets us pass a value deep into the component tree without explicitly threading it through every component.*



## UŽDUOTIS 6 - GLOBALŪS KINTAMIEJI

- sukurkite UserService implementaciją kad būtų įmanoma išsaugoti username
- įsidėkite UserService į globalų kontekstą
- kuriame nors paslapyje parašykite Labas, `<username>`!, kur username būtų panaudotas UserService username
- kuriame nors kitame puslapyje onClick nustatykite UserService username reikšmę
  - grįžkite į puslapį su pasisveikinimu ir username turi būti pasikeites



# DARBAS SU SERVERIU



## DARBAS SU SERVERIU

- Naudosime Axios (<https://github.com/mzabriskie/axios>)
- Sužinosime, kas yra Promise'as
- Pamatysime asynchroinij kodo vykdymą



## PAVYZDINIS API

- Duomenis krausime iš  
<https://itpro2017.herokuapp.com/swagger-ui.html#/>
- Tarkime šiuo metu parduodamų produktų sąrašas yra čia:  
<https://itpro2017.herokuapp.com/api/products>
- Kaip tai pakrauti į mūsų React'o appą?



# PAVYZDINIS API

- package.json

```
"axios" : "0.18.0"
```

- kaip naudoti: importuoti, įvykdyti užklausą, apdoroti atsakymą ir klaidas

```
import axios from 'axios';
axios.get('https://itpro2017.herokuapp.com/api/products')
  .then( (response) => {
    console.log(response);
  })
  .catch( (error) => {
    console.log(error);
} );
```



- `axios.get('URL')` grąžina Promise tipo objektą
- Promise objektas yra kaip pažadas, kad kažkas į jo vidų kažkada įdės reikšmę
- `then(fn)` ir `catch(fn)` yra Promise objekto funkcijos priimančios funkcijas (callback'us) kurios bus įvykdytos, kai kas nors į promise'ą įdės rezultatą



- `then(fn)` - fn yra pakviečiamas, kai serveris grąžina sėkmingą rezultatą (HTTP 200 OK)
- `catch(fn)` - fn yra pakviečiamas, kai serveris grąžina nesėkmingą rezultatą (HTTP >400)
- serverio kvietimas vyksta asinhroniskai - kodas vykdomas toliau, nors serveris dar neatsakė



## THEN(FN)

- callback funkcijai yra perduodamas response objektas
- <https://github.com/mzabriskie/axios#response-schema>
- data - serverio rezultato JSON'as
- status - serverio vykdymo kodas. 200 reiškia sėkmingą vykdymą.



- Kiekvienas HTTP method turi atitinkamą funkciją Axios bibliotekoje
- axios.get, axios.post, axios.delete, axios.put



## INTEGRACIJA SU REACT'U

- Dažniausiai serveris kviečiamas kai:
  - užkraunamas komponentas ir jį reikia užpildyti duomenimis
  - įvyksta vartotojo veiksmas (formos išsaugojimas) ir reikia nusiųsti serveriui duomenis



# UŽKRAUNANT PRADINĮ KOMPONENTO VAIZDĄ

```
componentDidMount() {  
    axios.get('https://itpro2017.herokuapp.com/api/products')  
        .then((response) => {  
            this.setState({ product: response.data });  
        })  
    }  
}
```



## UŽDUOTIS 7 - RODYKITE PRODUKTŲ SĄRAŠĄ IŠ SERVERIO

- Sukurkite ProductListContainer, kuris atsakingas už būsenos valdymą
- ProductListContainer container turi pakviesti serverį ir išpiešti ProductListComponent perduodamas jam produktų sąrašą
- Pakeiskite ProductListComponent taip, kad jis rodytu produktų sąrašą iš <https://itpro2017.herokuapp.com/api/products>
- Paveikslėliai nesimatys šiuo atveju.



## **UŽDUOTIS 8 - IŠSAUGOKITE NAUJĄ PRODUKTĄ SERVERYJE**

- ProductAdministrationComponent iškelkite būseną į ProductAdministrationContainer komponentą
- Spaudžiant mygtuką Save padarykite POST  
<https://itpro2017.herokuapp.com/api/products>



# KODO STRUKTŪRA

```
.  
|   └── public  
|       └── index.html  
└── src  
    ├── index.js  
    └── components  
        ├── Navigation  
        |   └── NavigationComponent.js  
        ├── ProductList  
        |   ├── ProductCardComponent.js  
        |   └── ProductListContainer.js  
        └── ProductAdministration  
            ├── ProductAdministrationComponent.js  
            └── ProductAdministrationContainer.js
```

index.js importuotumėm taip:

```
import ProductListContainer  
from './components/ProductList/ProductListContainer';
```



## KODO STRUKTŪRA - TASYKLĖS

- Iškelkite komponentus į atskirus failus
- index.js aprašykite React Router taisykles
- NavigationComponent.js aprašykite navigacijos meniu
- failo pavadinimas turi sutapti su komponento pavadinimu



## UŽDUOTIS 9 - PERTVARKYKITE KODĄ

- Atskirkite komponentus į atskirus failus
- sukurkite components katalogą ir tame sukurkite katalogus skirtiniems komponentams
  - komponento container'is irgi yra komponento dalis
- index.js aprašykite React Router taisykles



## UŽDUOTIS 10 - UŽBAIKITE EL. PARDUOTUVĘ

- <https://itpro2017.herokuapp.com>
- pabandykite patys sukurti tai, ko trūksta
- jei užstrigote, žiūrėkite į šalia skaidrių esančių dalies užduočių sprendimus
  - sprendimai ne visi
  - gali tekti persidaryti



## UŽDUOTIS 11 - PRIJUNGTI SASS

- React nerekomenduoja naudoti SASS. Pvz. vietoj to, kad .Button klasę naudotumėte <Ok> ir <Cancel> mygtukuose, React rekomenduoja sukurti vieną <Button> mugtuką, kurį <Ok> ir <Cancel> galėtų nupiešti render() metode
- Dėl to SASS/LESS mažai naudingi, nes vietoj to naudojama komponentų kompozicija
- Kaip prijungti SASS
  - <https://facebook.github.io/create-react-app/docs/adding-a-sass-stylesheet>



## SAVARANKIŠKAM PASIMOKYMUI

- <https://hackr.io/tutorials/learn-react> - nemokami kursai
  - pvz <https://www.codecademy.com/learn/react-101>
- React Biblioteka ir jos dokumentacija
  - <https://reactjs.org/>
- dalies užduočių sprendimai
  - <http://stasauskas.lt/itpro2018/>



# KITOJE PASKAITOJE

XML. Karkasai. Sistemų architektūros



AKADEMIJA.LT  
INFOBALT IR TECH CITY

# IŠ PRAEITOS PASKAITOS

- Daugiausiai klausimų dėl to, kad prieš tai nebandėte atlikti kokios nors užduoties
  - medžiaga yra koncentruota ir turi tam tikrą eilės tvarką, ir jei kažką praleidžiate, vis tiek teks sugrįžti
  - taigi jeigu nesuprantate užduoties - klauskite



# IŠ PRAEITOS PASKAITOS

- React "key" padeda React'ui nuspresti, kas pasikeitė DOM medyje, kad galetų atnaujinti tai, ką reikia
  - naudoti, pvz., generuojant įvairius sąrašus
  - unikalus tame masyve (ne visame puslapyje)

```
class Vaisiai extends Component {  
    render() {  
        var vaisiai = this.props.vaisiai.map(vaisius => {  
            return (<div className="col-sm-4" key={vaisius.pavadinimas}>  
                <Vaisius paveikslukas={vaisius.paveikslukas} ...>  
            </div>);  
        })  
        <..>  
    }  
}
```





**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

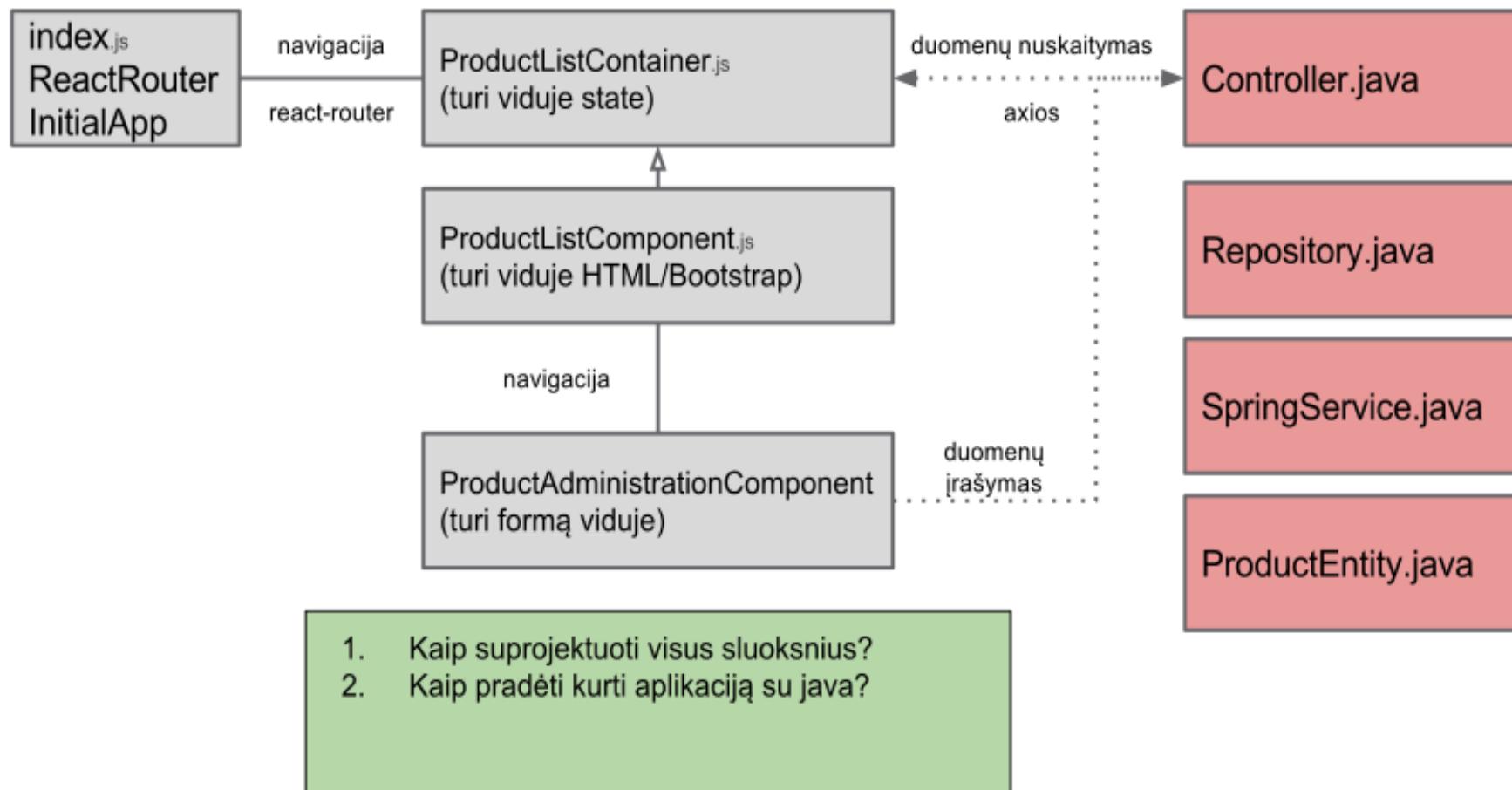
# XML. KARKASAI. SISTEMŲ ARCHITEKTŪROS

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# KĄ JAU MOKAME IR KO DAR NE



# PRIEŠ NAUDOJANT JAVA

- Ką reikia išmokti:
  - JSON
  - Rest API
  - XML/namespaces
  - kokią funkciją atlieka kokie karkasai
  - daugiasluoksnė architektūra
    - React pozicija joje
  - Tomcat
  - Maven - java "paketų manageris"
  - Spring



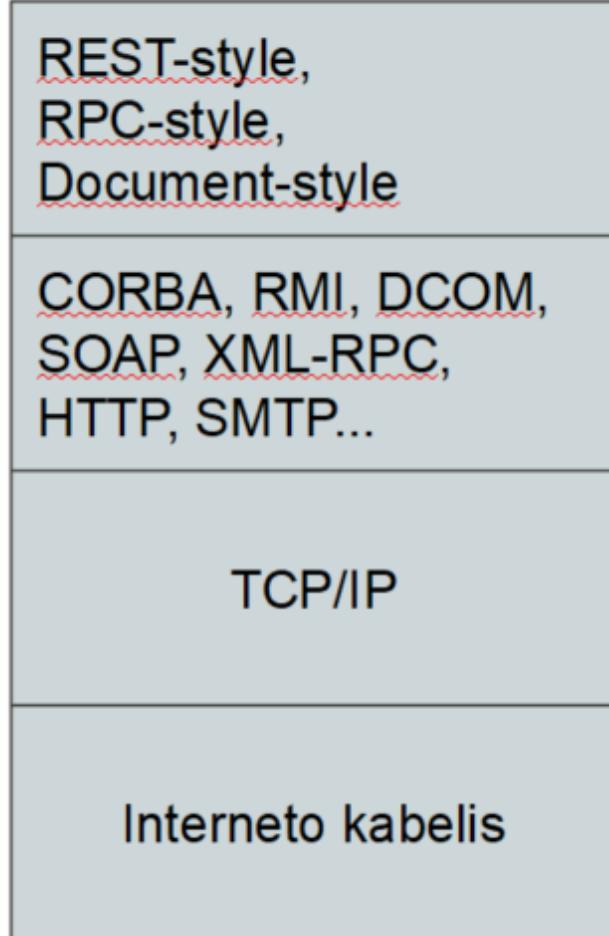
# TURINYS

- Rest API
- Duomenų reprezentavimo formatai
- XML ir vardų erdvės (zonos/namespaces)
- Karkasai
- Sistemų architektūros



# REST API

# KOMUNIKACIJOS PROTOKOLAI



Pranešimų siuntimo stiliai

Komunikacijos ir serializacijos protokolai

Duomenų transportas



## TERMINAI

- Pranešimai/Užklausos (angl. message/requests) - tinkle esančios programos bendrauja siūsdamos viena kitai pranešimus/užklausas.
- Galinis taškas (angl. endpoint) - tai URL adresą turintis servisas, kurį gali pasiekti klientas.
- URL (angl. Uniform Resource Locator) - tai nuoroda/adresas į kažkokį resursą tinkle, nurodant būdą, kaip tą resursą pasiekti.
- URI (angl. Uniform Resource Identifier) - tai nuoroda/adresas į kažkokį resursą tinkle.



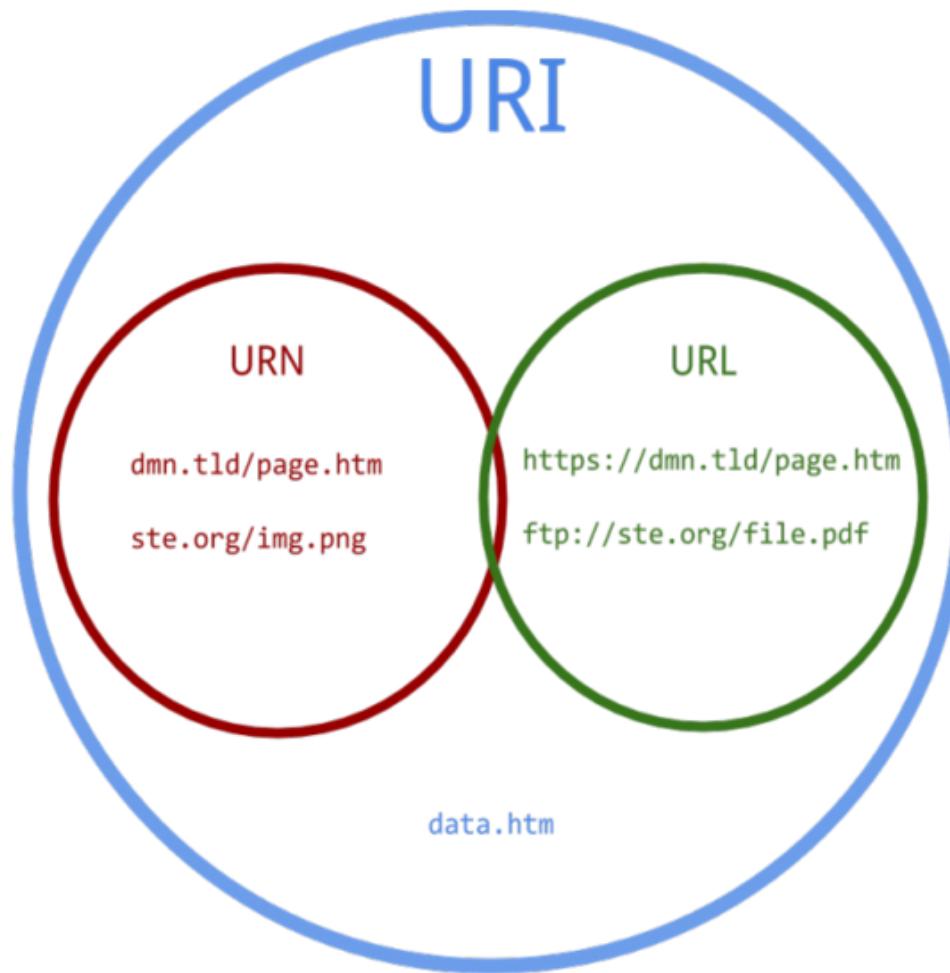
# URL

- Bendrinė URL schema:

```
schema : [ //mašinosvardas[:portas]] [/]kelias[?parametrai][#fragmentas]
```

- Schema - nurodo protokolą, kuriuo bus pasiekiamas resursas. Galimi protokolai: HTTP, FTP ir pan.
- Mašinos vardas - unikalus serverio vardas (registruotas vardas arba IP adresas)
- Portas - skaičius, nurodantis loginę jungtį (programos adresą) toje mašinoje.
- Kelias - relatyvus resurso adresas
- Parametrai - užklausos parametrai
- Fragmentas - nuoroda/adresas dokumento lygyje

# URI, URL, URN



# DUOMENŲ REPREZENTAVIMO FORMATAI

- Duomenų reprezentavimo formatai:
  - XML
  - JSON
  - YAML
  - Text
  - ASN.1
  - Ir t.t.



## JSON FORMATAS

*Tai atviro standarto duomenų formatas, naudojantis atributo/reikšmės poras. Kadangi formatas yra minimalistinis, hierarchinis bei skaitomas, tai jis yra paprastas ir lankstus. Tai yra pats populiausias duomenų apsikeitimo formatas vykdant asinchronines web užklausas (AJAX), kuris didžiąja dalimi pakeitė kitą plačiai naudojamą formatą XML.*

Autorius: Douglas Crockford



# JSON FORMATAS

- Duomenų tipai:
  - Skaičius: 1, 2, 3 ...
  - Simbolių eilutė: “a”, “AbC”, ...
  - Boolean: true, false
  - Masyvas: [1, 2, 3], [“a”, “AbC”]
  - Objektas: { name: “Student”, age: 25 }
  - Null



# JSON FORMATAS

- kelių lygių JSON formato pavyzdys

```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "isAlive": true,  
    "age": 25,  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": "10021-3100"  
    },  
    "phoneNumbers": [  
        {  
            "type": "home",  
            "number": "212 555-1234"  
        },  
        {  
            "type": "mobile",  
            "number": "123 456-7890"  
        }  
    ],  
    "children": [],  
    "spouse": null  
}
```



# KAS YRA RESTFUL SERVISAI?

- Orientuotas į resursą (angl. Resource based)
- Reprezentacijos (angl. Representations)
- Apribojimai
  - Vienodas interfeisas (angl. Uniform Interface)
  - Be būsenos (angl. Stateless)
  - Klientas-Serveris (angl. Client-Server)
  - Kešuojamas (angl. Cacheable)
  - Sluoksniuota sistema (angl. Layered System)
  - Kodas pagal pageidavimą (angl. Code on Demand)

Originaliai aprašė Roy Fielding savo [disertacijoje](#)



## KAS YRA RESTFUL SERVISAI?

- Resursai identifikuojami pagal URI
  - URI formuojamas hierarchijos principu
  - Naudojami daiktavardžiai
- Naudojami HTTP metodai
  - Šie metodai - tai veiksmažodžiai
  - Nusako veiksmą, kuris bus atliekamas su resursu: (GET, POST, PUT, DELETE)
- Rezultato identifikavimui naudojami HTTP statusų kodai



## REST-STILIAUS SERVISŲ KŪRIMO GAIRĖS

- Naudoti HTTP veiksmažodžius - tai servisams suteikia daugiau prasmės:
  - GET - Resurso skaitymas (pagal identifikatorių) arba resursų aibės.
  - POST - Resurso sukūrimas.
  - PUT - Resurso modifikavimas (pagal identifikatorių) arba resursų aibės. Gali būti naudojamas resurso sukūrimui, jei identifikatorius yra žinomas iš anksto.
  - DELETE - Resurso trynimas.

Pastaba: GET užklausos neturi pakeisti resurso būsenos.



## **REST-STILIAUS SERVISŲ KŪRIMO GAIRĖS**

- Kiekvienas resursas yra identifikuojamas URI adresu. Reikia stengtis tokius adresus kurti prasmingus, išnaudojant hierarchinę struktūrą, kurią mums suteikia URI sintaksė.
- Vietoj HTTP parametrų naudoti URI identifikatorius
  - OK: /users/123456
  - NOT OK: /api?type=user&id=123456



## REST-STILIAUS SERVISŲ KŪRIMO GAIRĖS

- Naudoti URI hierachijos galimybes struktūros išreiškimui (kažkas yra kažko loginė dalis)
- Projektuoti klientams, ne duomenims
- Resursų pavadinimai turėtų būti daiktavardžiai.  
Veiksmažodžiai - tai HTTP metodai.
- Atskirus žodžius atskirti vienu iš simbolių: ‘\_’ arba ‘-’
- Stengtis minimizuoti adreso ilgj. Jis turi būti maksimaliai aiškus ir trumpas



# **REST-STILIAUS SERVISŲ KŪRIMO GAIRĖS**

- Užklausos rezultatas identifikuojamas naudojant HTTP statusų kodus:

## **Statusas      Aprašymas**

200 OK	Bendro pobūdžio sėkmės kodas. Dažniausiai naudojamas statusas.
201 CREATED	Sėkmingas sukūrimo veiksmas (naudojant PUT arba POST metodus).
204 NO CONTENT	Nurodo sėkmingą veiksmo atlikimą, tačiau HTTP atsakymas yra be jokio turinio.
400 BAD REQUEST	Bendrinis klaidos kodas: duomenys nėra validūs, nėra prieinami ir pan.
404 NOT FOUND	Resursas nerastas.



# **XML IR XHTML**

**.. IR VARDŲ ERDVĖS**



# XML

- Panašus į HTML
- Turi žymes - tačiau žymes galime susikurti patys
- XML failo atvaizdavimui naudojamos XSLT transformacijos
- HTML taip pat gali būti XML failas
- Dažniausiai HTML, kuris yra XML failas, vadinamas XHTML ir naudojamas XHTML standartas
- Naudojamas Maven, Tomcat, Spring konfigūracijai



# XML

- Galimi įvairūs variantai:

```
<?xml version="1.0" encoding="UTF-8"?>
<saknis/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<saknis></saknis>
```

```
<saknis/>
```



# XML VALIDAVIMAS

- Validavimas kelių lygiu:
  - geras suformavimas (well formedness)
    - <http://www.validome.org/xml/validate/>
  - validacija pagal schema
    - <https://validator.nu/>
- XML taisyklės (žodynas), o ir validavimas - kelių tipu:
  - pagal DTD (Doctype)
    - HTML yra XML, gali būti suvaliduotas pagal DTD
  - pagal XSD (XML Scema)



## XML VARDŲ ZONOS

Jeigu norėtume sudaryti dokumentą pagal keletą taisyklių rinkinių (ir vėliau jį validuoti, patikrinti) - pagal HTML ir mūsų žymes:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
    <manoHtml></manoHtml>
</html>
```



## XML VARDŲ ZONOS

O jeigu formuotumėme dokumentą taip, kad turinys būtų body žymėje? Kaip nurodyti, iš kurios vardų zonas?

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
    <body>
        html turinys
        <manoHtml>
            <body>
                mano turinys
            </body>
        </manoHtml>
    </body>
</html>
```



# XML VARDŲ ZONOS

Naudojame atributą xmlns be prefikso:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/TR/html4/">
    <body>
        html turinys
        <manoHtml>
            <body>
                mano turinys
            </body>
        </manoHtml>
    </body>
</html>
```



# XML VARDŲ ZONOS

Pridedame ir asmeninę vardų zoną kitu prefiksui:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/TR/html4/"
      xmlns:prefiksas="http://mano.lt">
<body>
    html turinys
    <prefiksas:manoHtml>
        <prefiksas:body>
            mano turinys
            </prefiksas:body>
        </prefiksas:manoHtml>
    </body>
</html>
```



# PRADINIS PUSLAPIO KODAS

```
<HTML>
    <HEAD>
        <TITLE>Infobalt mokykla</TITLE>
    </HEAD>
    <BODY>
        Rodomas tekstas.
    </BODY>
</HTML>
```



# PRADINIS PUSLAPIO KODAS

```
<HTML xmlns="http://www.w3.org/TR/html5/">
    <HEAD>
        <TITLE>Infobalt mokykla</TITLE>
    </HEAD>
    <BODY>
        Rodomas tekstas.
    </BODY>
</HTML>
```



# PRADINIS PUSLAPIO KODAS JSF

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Infobalt mokykla</title>
    </h:head>
    <h:body>
        Rodomas tekstas.
    </h:body>
</html>
```



# SVG VARDŲ ZONOS PAVYZDYS

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>SVG</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg">
      <rect stroke="black" fill="blue" x="45px" y="45px"
            width="200px" height="100px" stroke-width="2" />
    </svg>
  </body>
</html>
```



## UŽDUOTIS 1 - XML

- Pabandykite kiekvieną iš anksčiau rodytų pavyzdžių
  - patikrinkite kodo well formness
  - patikrinkite validaciją pagal schemą



# KARKASAI

## KOMPONENTŲ KARKASAI

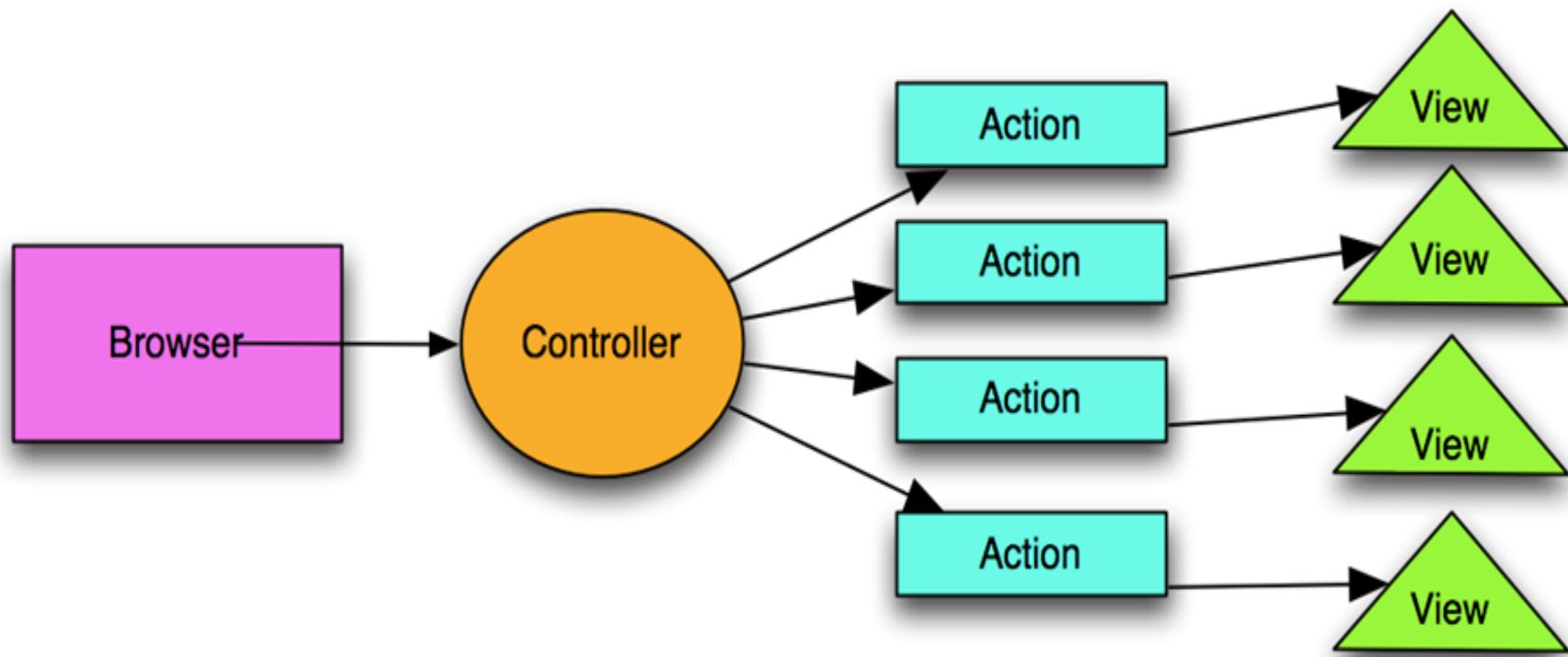


## KARKASAI

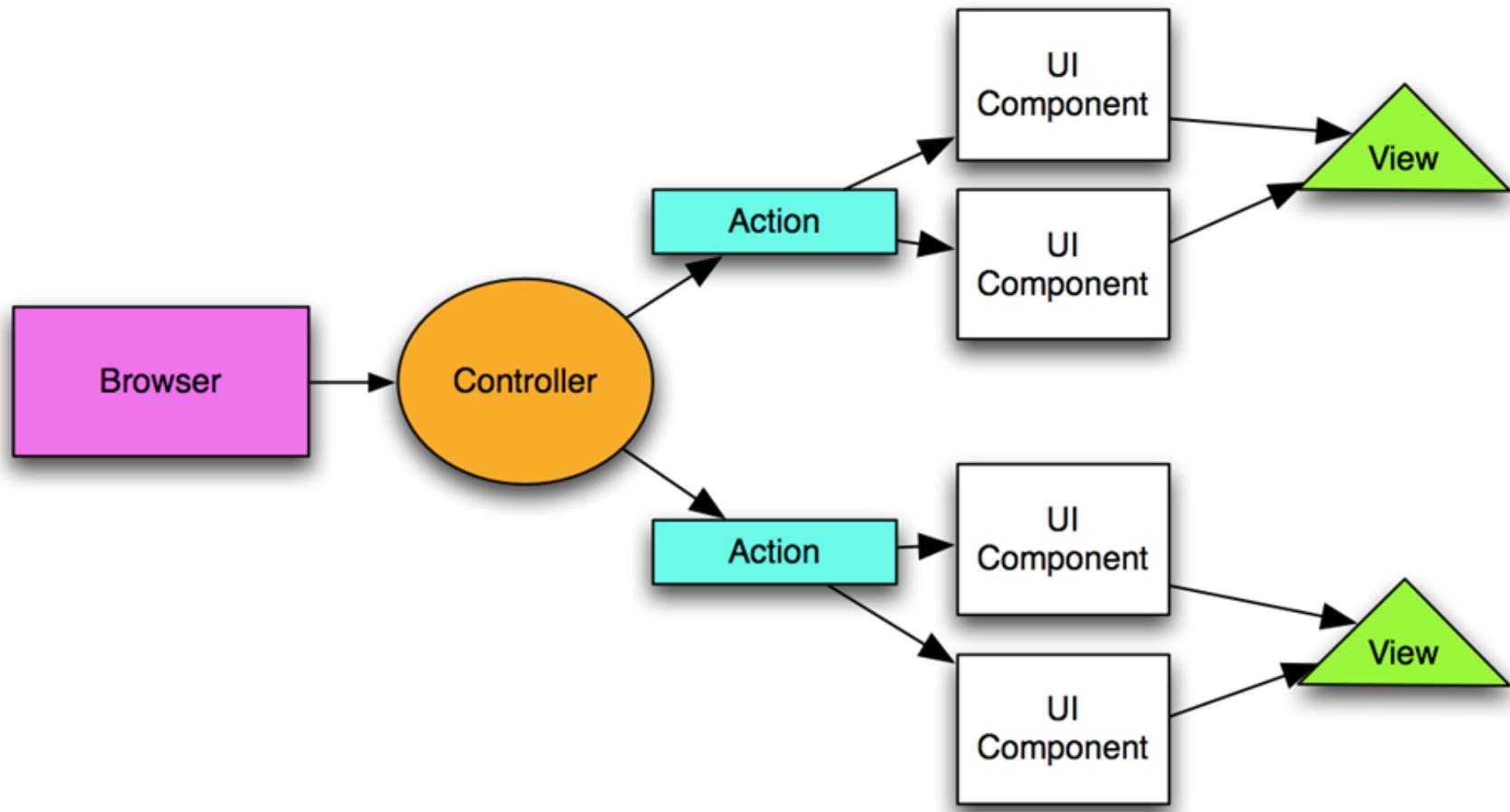
- Veiksmų karkasai (action)
  - Spring MVC, Struts, Struts2, Rails
- Hibridiniai karkasai (hybrid)
  - Tapestry, Wicket
- UI komponentų karkasai
  - JSF, Rife, Echo2
  - Dauguma UI JS karkasų: ReactJS, AngularJS ir kt.



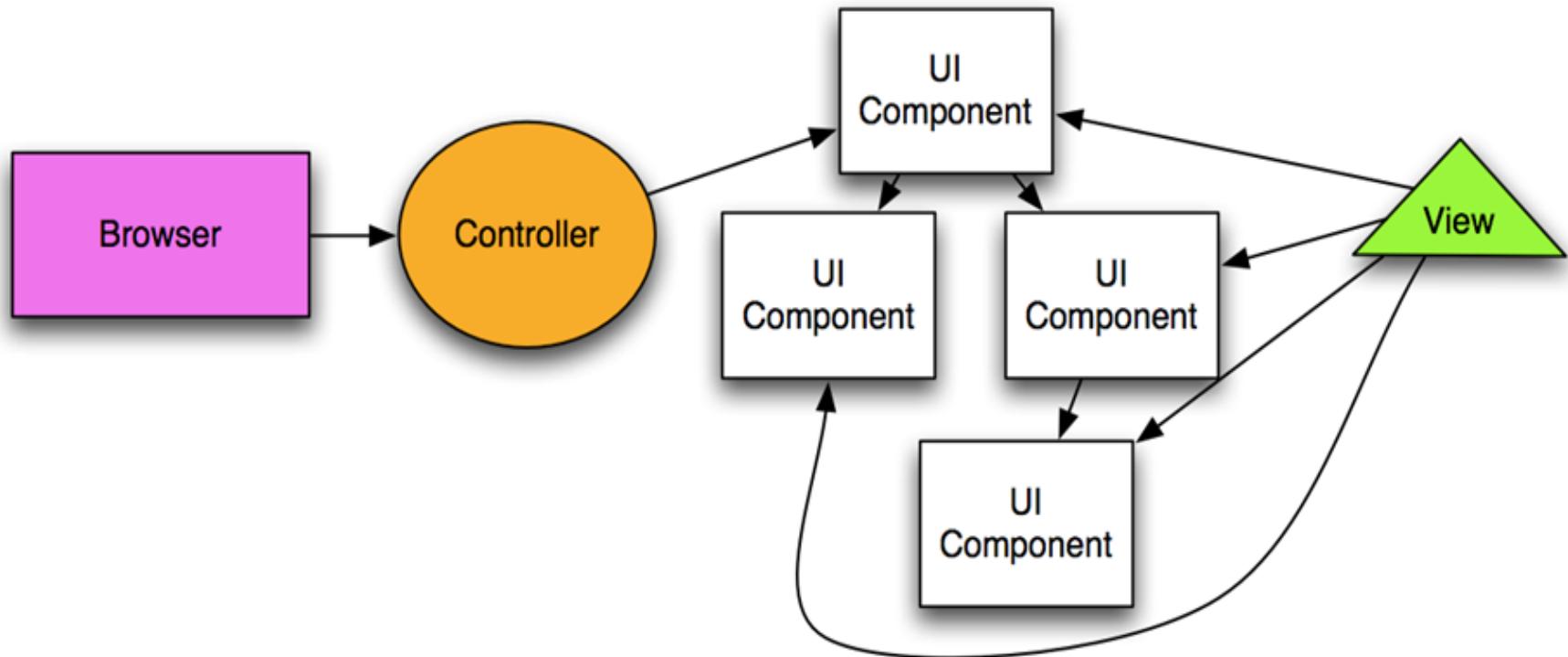
# VEIKSMŲ KARKASAI



# HIBRIDINIAI KARKASAI



# UI KOMPONENTŲ KARKASAI



## KOMPONENTŲ KARKASAI

- Karkasas su komponentais, kurie yra vaizde, komunikuojant tiesiogiai
- Egzistuoja komponentų medis, todėl lengva eiti per komponentus ir valdyti jų gyvavimo ciklą
- Saugoma komponentų būsena (state)
- Modelis dinamiškai atsinaujina pagal komponentus, esančius vaizde
- Gerai tinka sudėtingesniems puslapiams, kuriuose reikalingas didelis komponentų skaičius



# DAUGIASLUOKSNĖ SISTEMŲ ARCHITEKTŪRA



## SISTEMŲ ARCHITEKTŪRA

- IT sistemų architektūra yra aukšto lygmens dizainas, kuriuo yra paremta sistema.
- Architektūra turi sekančias savybes:
  - komponentai,
  - bendradarbiavimas (kaip komponentai saveikauja),
  - jungtys (kaip komponentai bendrauja).



## POPULIARIAUSIOS ARCHITEKTROS

- Kliento-serverio,
- Sluoksniuota (angl. tiered),
- Lygiarangių (angl. peer-to-peer),
- Kitos (pvz pipes and filters).



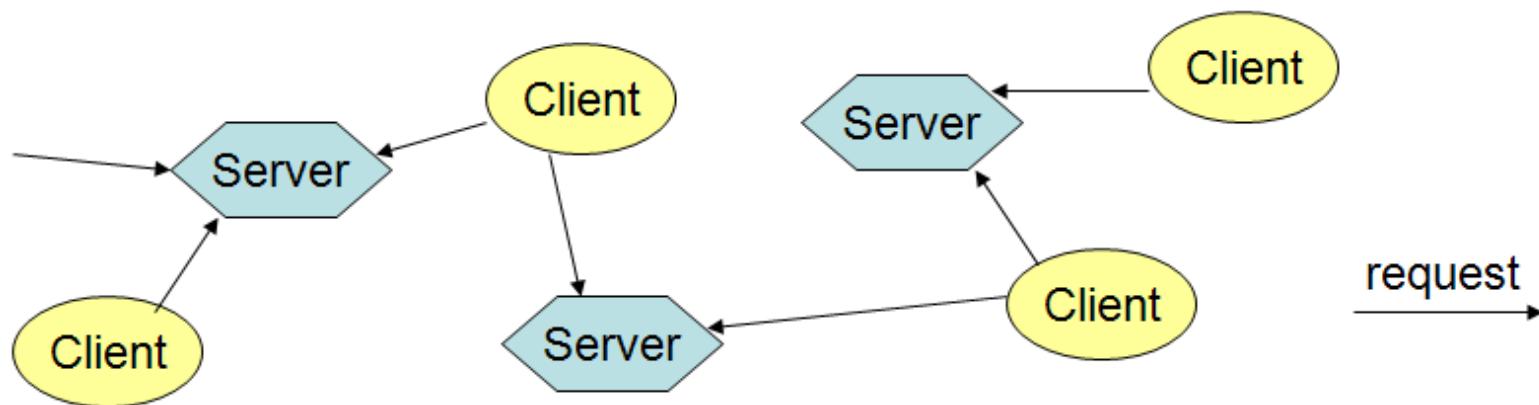
## KLIENTO-SERVERIO ARCHITEKTŪRA

- Kiekvienas kliento-serverio architektūros komponentas turi arba kliento, arba serverio rolę:
  - Klientas - komponentas kuris sukūria užklausas. Klientai yra aktyvus ir inicijuoja tranzakcijas.
  - Serveris - komponentas kuris patenkina užklausas. Serveriai yra pasyvus ir reguoja į kliento užklausas.
  - Serveris gali aptarnauti daugiau nei vieną klientą.
- Žiniatinklis yra kliento-serverio sistema.



# Kliento-serverio architektūra

- Interneto naršyklės elgiasi kaip klientai ir sukuria užklausas žiniatinklio serveriams.
- Žiniatinklio serveriai atsako į užklausas ir / arba atlieka skaičiavimus.



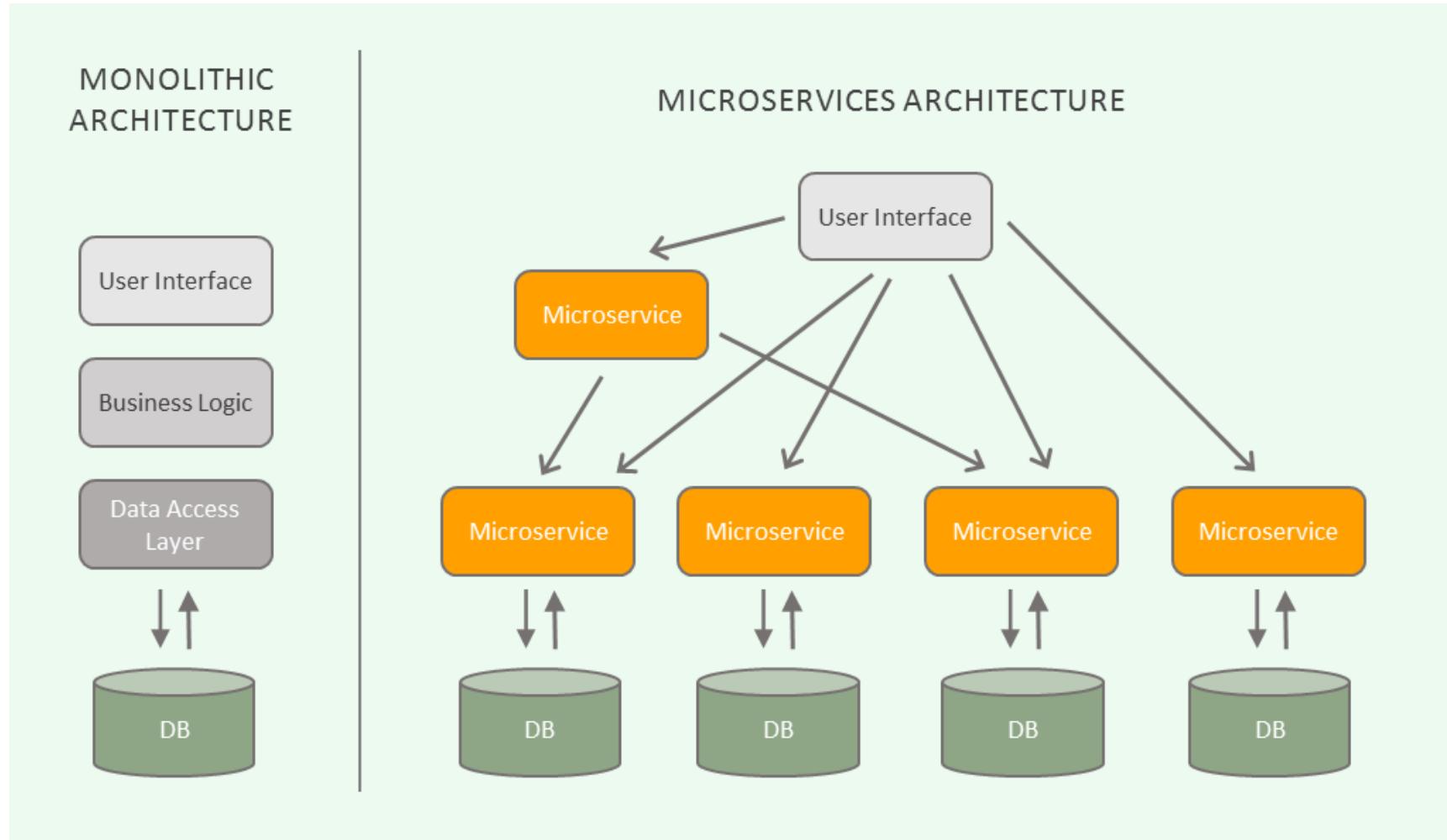
# MIKROSERVIAI

*In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. -- James Lewis and Martin Fowler*

- pradėjo populiarėti nuo 2014
  - <https://www.martinfowler.com/microservices/>
  - <https://martinfowler.com/articles/microservices.html>



# MIKROSERVISAI IR MONOLITINĖ SISTEMA



## CENTRALIZUOTAS / PASKIRSTYTAS APDOROJIMAS

- Kliento-serverio architektūrą galima laikyti viduriu tarp:
  - Centralizuoto apdorojimo - skaičiavimai atliekami naudojant centrinę platformą, kuri pasiekiamas per “kvailus” terminalus.
  - Paskirstyto apdorojimo - skaičiavimai atliekami taip pat ir ant naudotojo platformos.



## STORAS / PLONAS KLIENTAS

- Storas klientas:
  - Dalis aplikacijos veikia kliento pusėje.
  - Klientas žino kaip struktūrizuoti duomenis ir kur jie yra.
  - Skirtingi klientai gali naudoti aplikaciją skirtingais būdais.
- Plonas klientas:
  - Klientas yra mažiau sudėtingas.
  - Dauguma kodo veikia serverio pusėje.
  - Tinklo saveika tarp kliento ir serverio yra sumažinta.



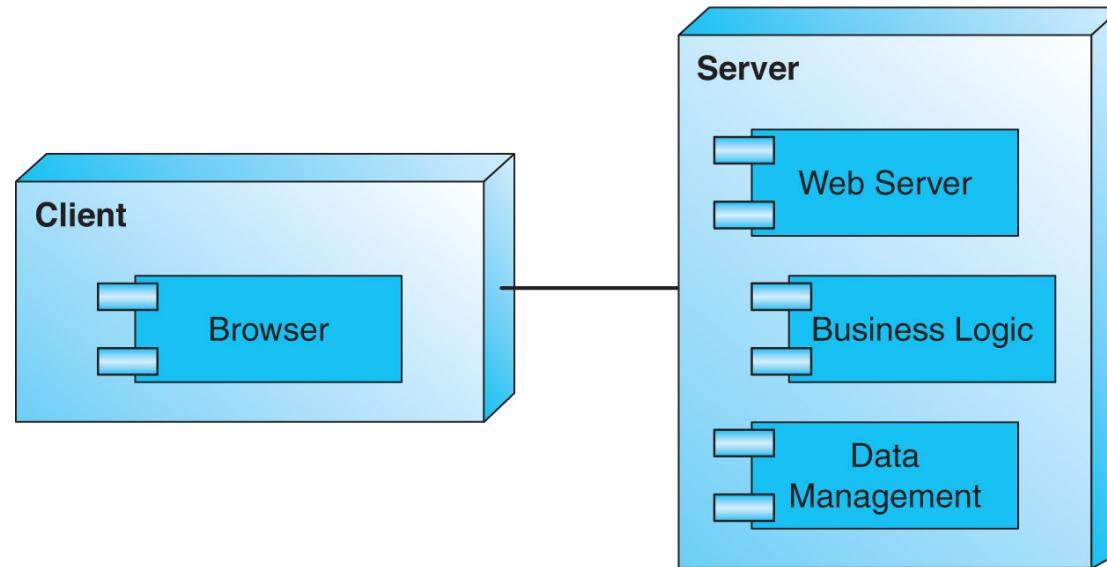
## SLUOKSNIUOTA ARCHITEKTŪRA

- 1 sluoksnio architektūra:
  - monolitinės IT sistemos,
  - naudotojo sąsaja, veiklos logika ir duomenų valdymas yra sujungti į vieną sluoksnį.
- Žiniatinklio aplikacijos įprastai realizuojamos naudojant 2, 3 arba daugėlio (N) sluoksniių architektūrą.
- Kiekvienas sluoksnis turi unikalią atsakomybę.



## 2 SLUOKSNIŲ ARCHITEKTŪRA

- 1 sluoksnis yra naudotojo sasajos sluoksnis.
- 2 sluoksnis yra veiklos logikos ir duomenų valdymo sluoksnis.



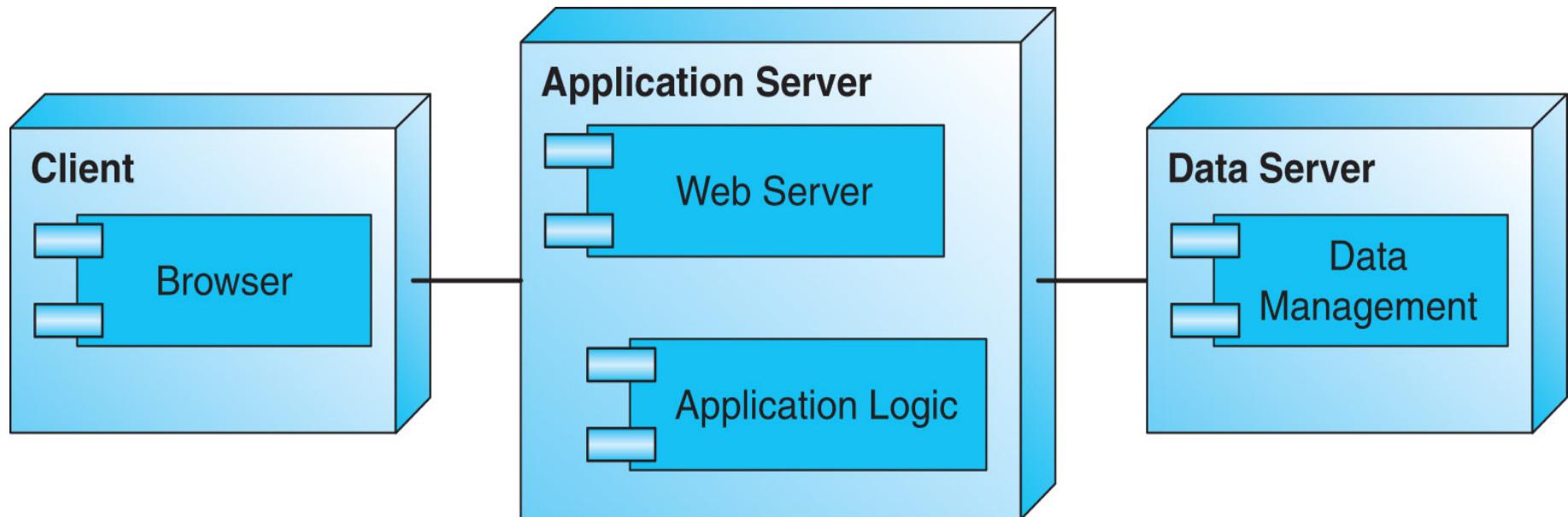
## 2 SLUOKSNIŲ ARCHITEKTŪROS CHARAKTERISTIKOS

- Privalumai: nebrangi
- Trūkumai:
  - komponentų tarpusavio priklausomybė
  - nėra dubliavimo (angl. redundancy)
  - ribotas išplečiamumas (angl scalability)
- Tipinė aplikacija:
  - 10-100 naudotojų
  - mažos įmonės arba organizacijos



## 3 SLUOKSNIŲ ARCHITEKTŪRA

3 sluoksnis paprastai perima iš 2 sluoksnio duomenų valdymo atsakomybę.



## 3 SLUOKSNIŲ ARCHITEKTŪROS CHARAKTERISTIKOS

- Privalumai:
  - padidintas našumas dėl specializuotos aparatinės įrangos
  - sumažinta komponentų tarpusavio priklausomybė
  - padidintos išplečiamumo galimybės
- Trūkumai: nėra dubliavimo (angl. redundancy)
- Tipinė aplikacija:
  - 100-1000 naudotojų
  - vidutinės įmonės arba regiono lygio organizacijos



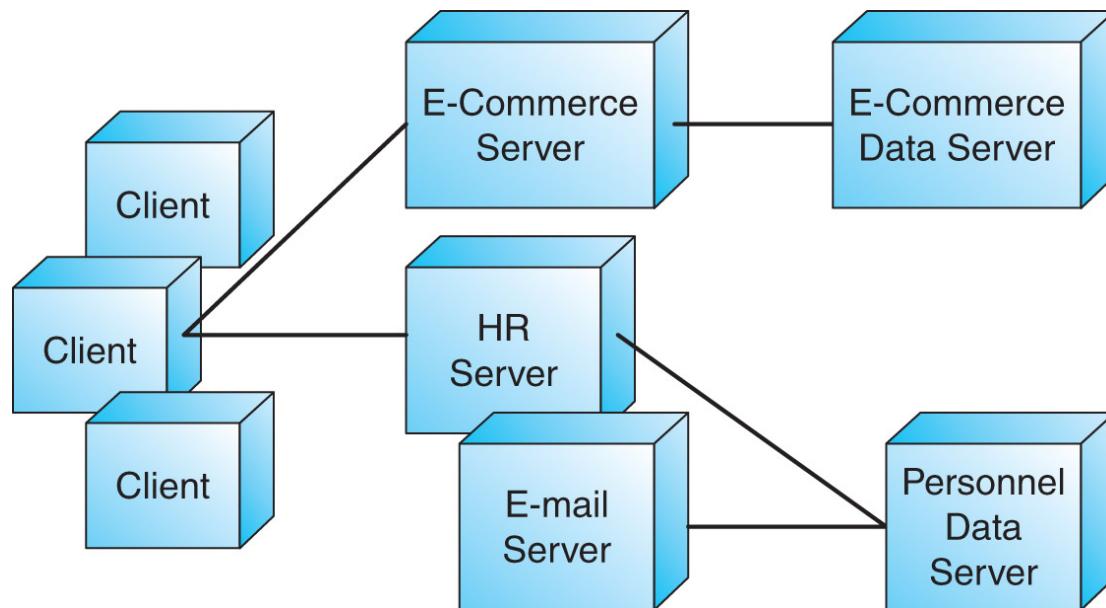
## **DAUGELIO SLUOKSNIŲ ARCHITEKTŪRA**

- Daugėlio (N) sluoksnių architektūra išplečia 3 sluoksnių architekturą vienu ar keleta galimų būdų:
  - sluoksnio funkcionalumo replikavimas
  - specializuota funkcija sluoksnuje
  - portalo paslaugos, orientuotos į žiniatinklio apkrovos valdymą



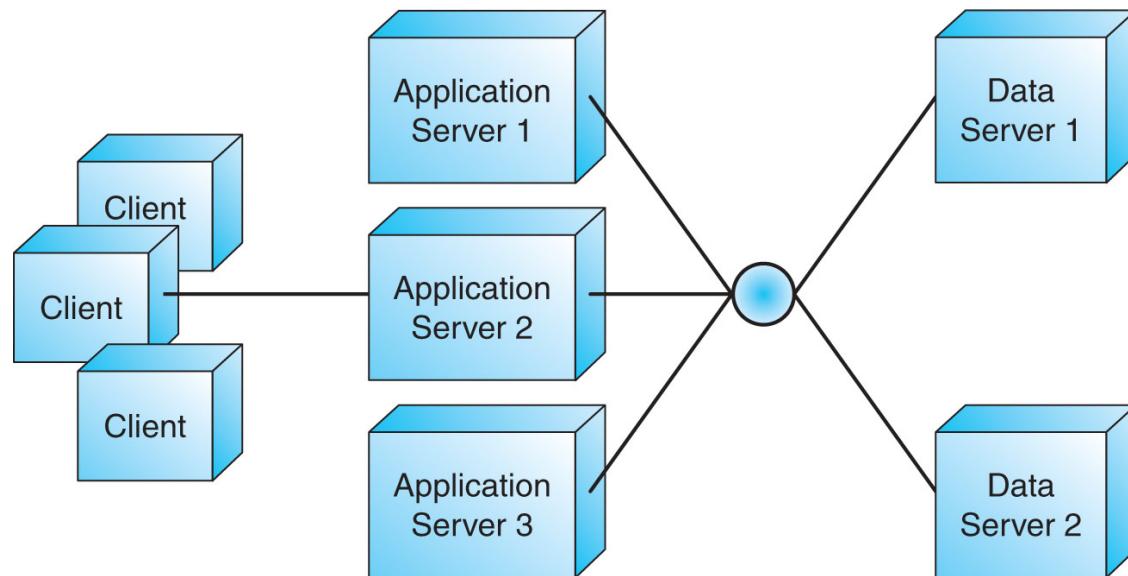
# REPLIKAVIMAS

Aplikacijų ir duomenų serveriai yra replikuoti ir bendrai dalinasi apkrova.



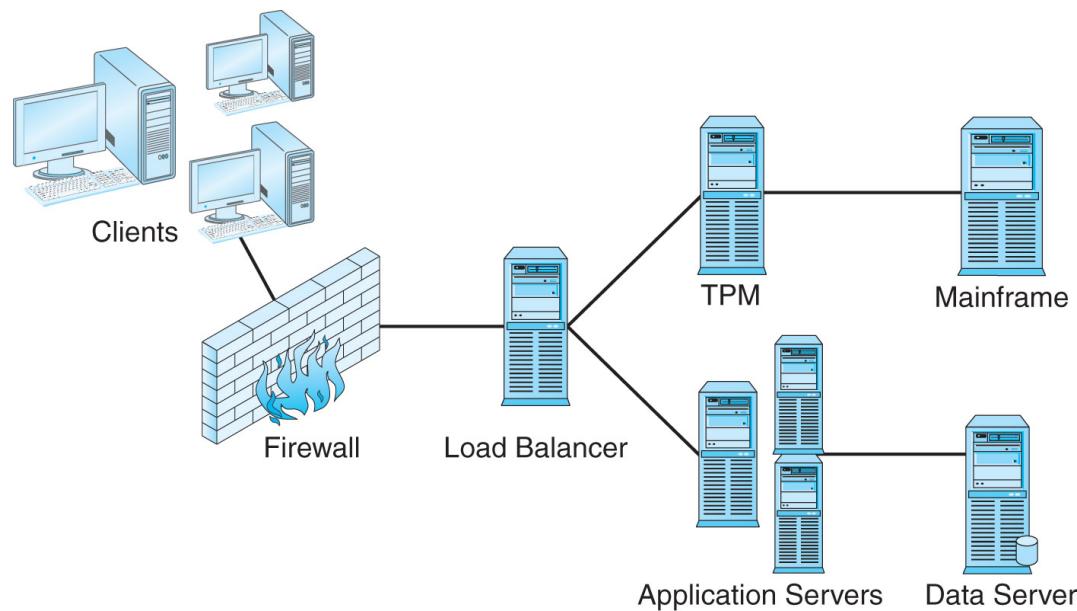
## SPECIALIZACIJA

Serveriai yra specializuoti ir pagal specifinę funkciją, apdoroja jiems paskirtą krūvį.



# PORTALO PASLAUGOS

Portalo serveriai apdoroja įeinantį srautą, sumažindami aplikacijų serverio apkrovą.

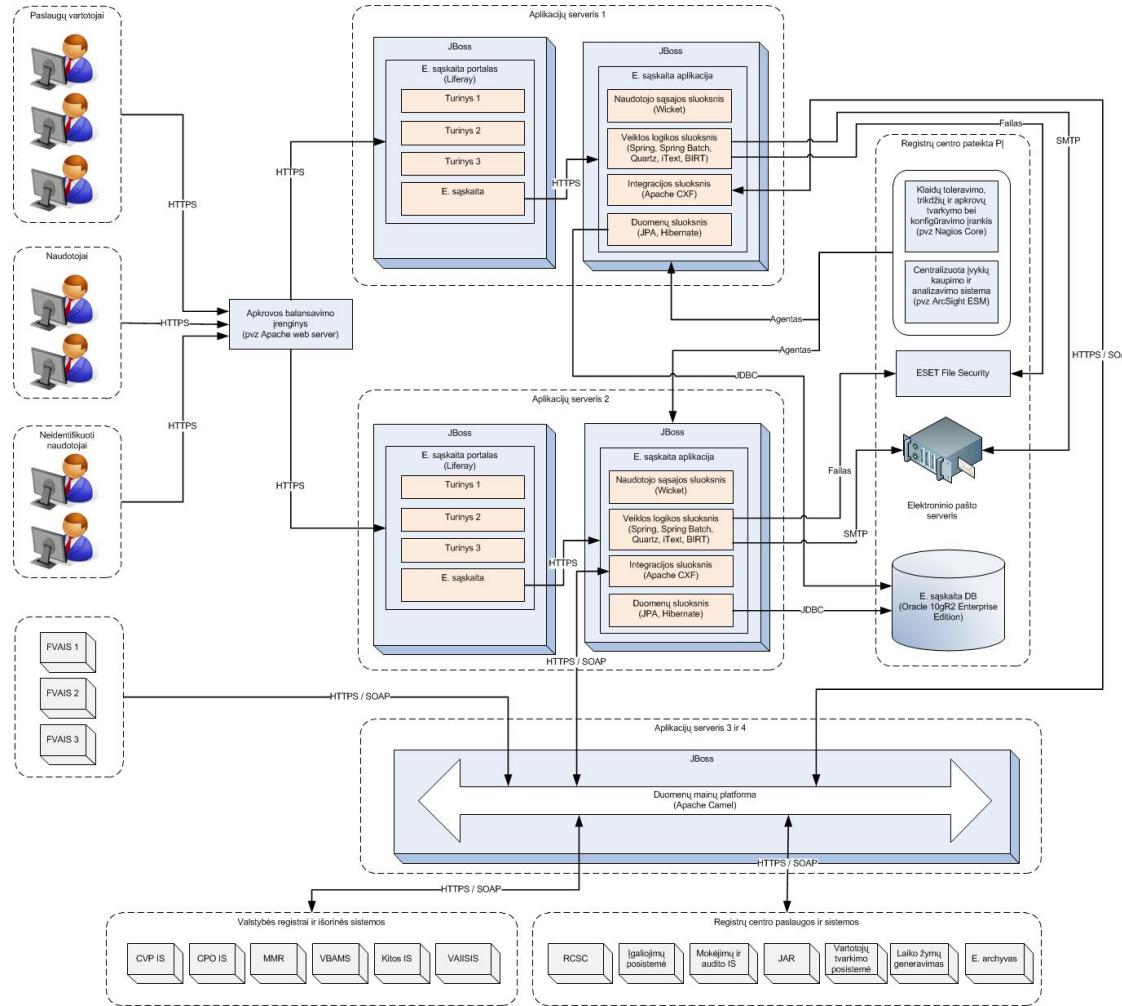


# N SLUOKSNIŲ ARCHITEKTŪROS CHARAKTERISTIKOS

- Privalumai:
  - atsieti programinės įrangos komponentai
  - lankstus komponentų pridėjimas / pašalinimas atsižvelgiant į apkrovą
  - išplečiamumas
  - dubliavimas
- Trūkumai: didesni priežiūros kaštai
- Tipinė aplikacija:
  - 1000+ naudotojų
  - didelės įmonės arba organizacijos



# N SLUOKSNIŲ ARCHITEKTŪROS PAVYZDYS



## KOKIĄ APLIKACIJĄ KURSIME

- Daugiasluoksnę verslo aplikaciją:
  - naudotojo sąsajos sluoksnis - React biblioteka-karkasas
  - veiklos logikos sluoksnis - Spring karkasas
  - duomenų valdymo sluoksnis - JPA (Java Persistence API)
- Aplikacijos į mikro servisus neskaidysime
  - tačiau analogiškai būtų kuriami ir mikro servisi



## UŽDUOTIS 2 - PABAIGTI APLIKACIJĄ

- Pabaigti parduotuvės aplikaciją
  - dalis funkcionalumo čia  
<https://itpro2017.herokuapp.com/>
  - REST API čia  
<https://itpro2017.herokuapp.com/swagger-ui.html#/>



## UŽDUOTIS 2 - PABAIGTI APLIKACIJĄ

- UI turi būti galima:
  - pirmame puslapyje matyti navigacijos eilutę ir produktų sąrašą
  - turi būti galima įvesti naudotojo vardą ir aplikacija su juo turi dirbti
  - produktų administravimo lange pridėti, redaguoti, ištrinti produktą
  - įdėti produktą į nurodyto naudotojo krepšelį
  - peržiūrėti krepšelį
  - ištrinti produktą iš krepšelio



# KITOJE PASKAITOJE

Tomcat. Maven. Java. Spring



# IŠ PRAEITOS PASKAITOS

- kaip išvengti history objekto perdavimo per props?
  - react-router 4.4.0-beta+ bus `__RouterContext`
  - o dabar galima pavyzdžiui `withRouter`

```
import { withRouter } from 'react-router';
<..>
export default withRouter(AppContainer)
// arba jeigu komponentas tame pačiame modulyje
var AppContainer = withRouter((props) => {
  return (<div>
    <button onClick={() => props.history.push("products")}; >/>
    {props.children}
```

- į props objektą `history/match/location` gaus tik šis komponentas bet kuriame komponentų medžio lygyje



# IŠ PRAEITOS PASKAITOS

- arrow funkcijos metodas klasėje gauna objektą `this`

```
class C extends Component {  
    handleChange = (event) => {  
        this.setState({value: event.target.value});  
    }  
}
```

- be arrow funkcijos reiktų naudoti `bind()`:

```
class C extends Component {  
    constructor(props) {  
        super(props);  
        this.handleChange = this.handleChange.bind(this);  
    }  
    handleChange(event) {  
        this.setState({value: event.target.value});  
    }  
}
```



# IŠ PRAEITOS PASKAITOS

- formose <input type="text">, <textarea> ir <select> visi gali turėti value={this.state...}
  - arba keli pasirinkimai <select multiple={true} value={[ 'B' , 'C' ]}>
- vienos funkcijos pvz. visiems elementams formoje:

```
handleChange = (event) => {
  const target = event.target;
  const value = target.type==='checkbox'?target.checked:target.value;
  const name = target.name;
  this.setState({ [name]: value });
}
// formos laukai turi turėti name, pvz. <input name="<kaip state>"
```



# IŠ PRAEITOS PASKAITOS

- jeigu linux'uose:
  - neatsinaujina po pakeitimų react aplikacija
  - nepasistartuoja su npm start ir iškrenda klaida, susijusi su WATCH
  - kažkuriam laikui pradėjus neatsinaujina react vaizdas naršyklėje
- galimai pasiekėte linux watch (failų stebėjimų) limitą, o jį padidinti galima taip:

```
echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p
```



**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

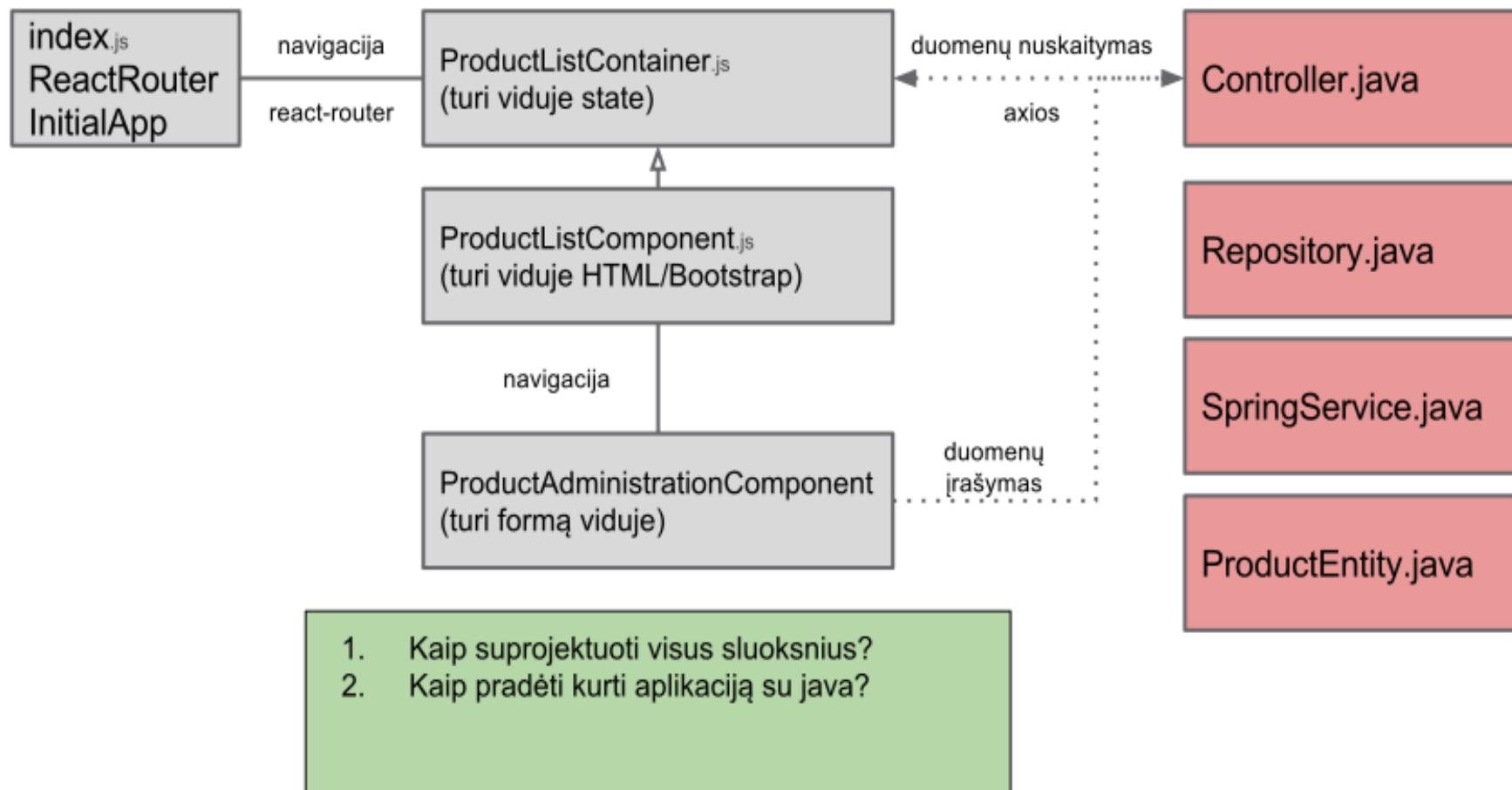
# TOMCAT. MAVEN. SPRING BOOT APLIKACIJA

Andrius Stašauskas

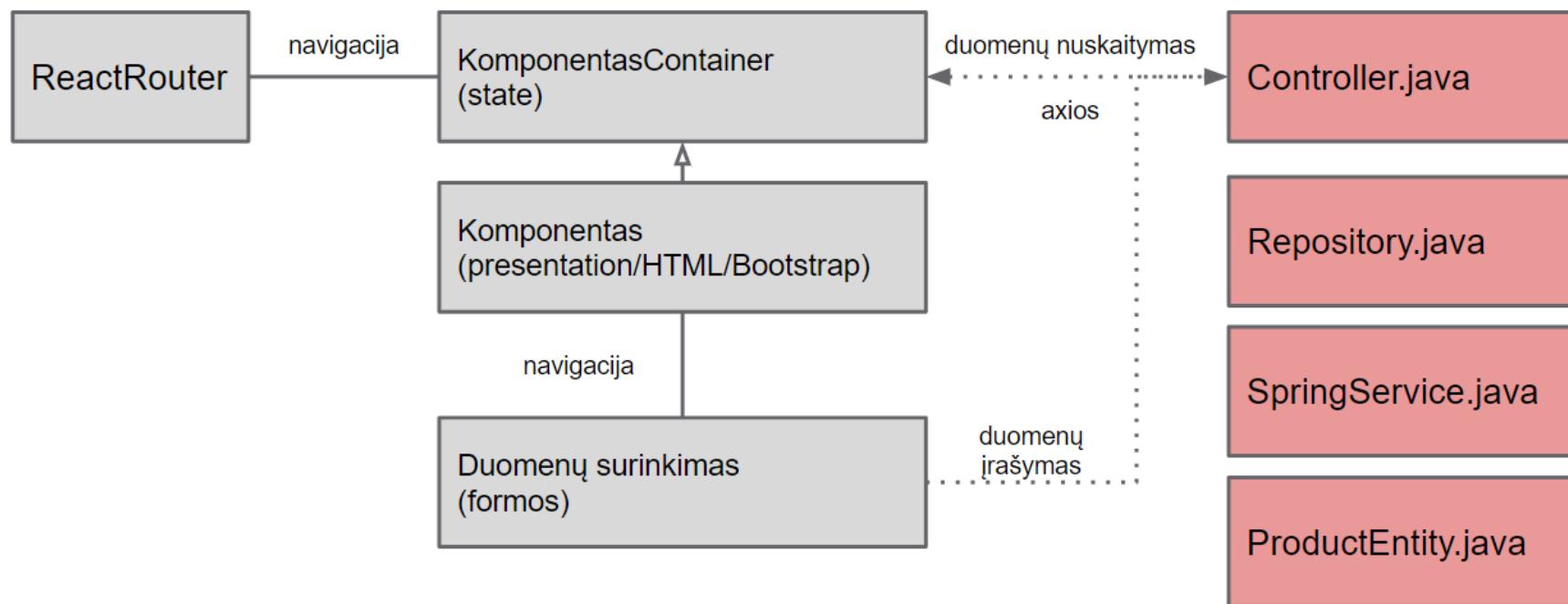
andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

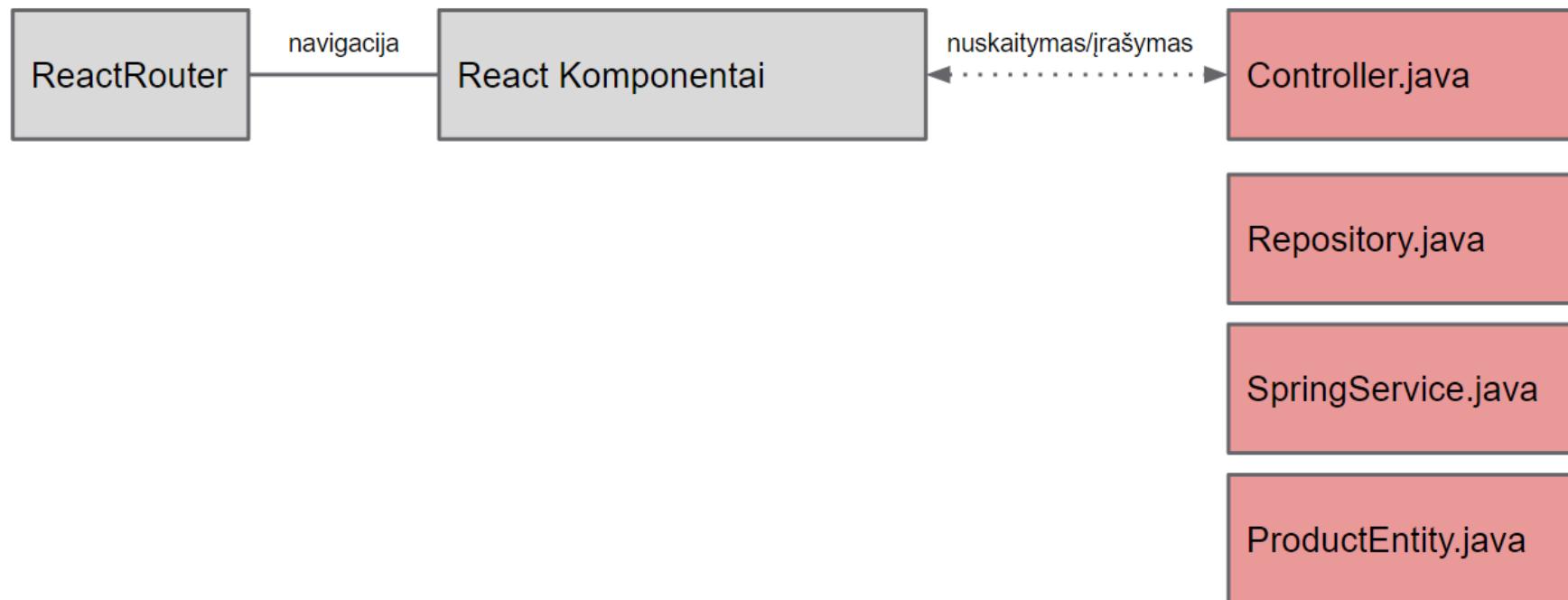
# KĄ JAU MOKAME IR KO DAR NE



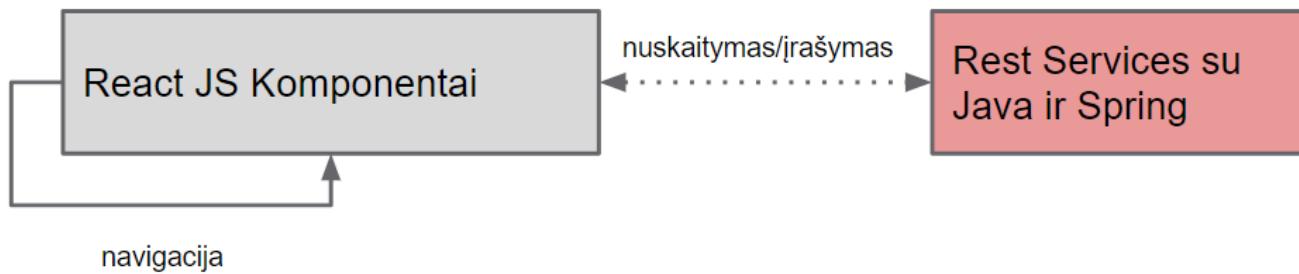
# KĄ JAU MOKAME IR KO DAR NE



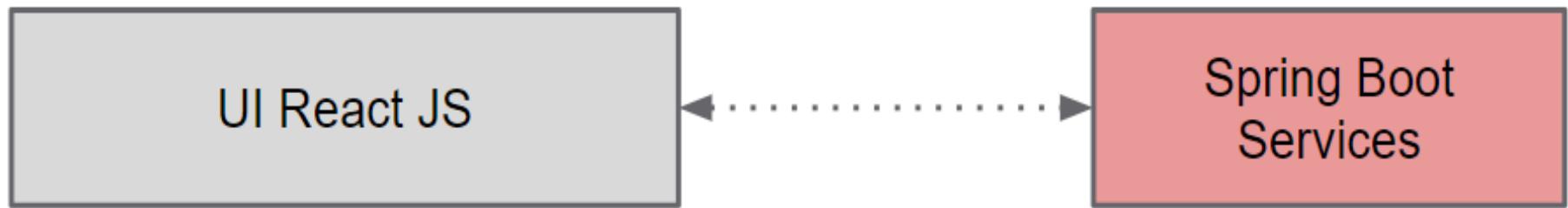
# KĄ JAU MOKAME IR KO DAR NE



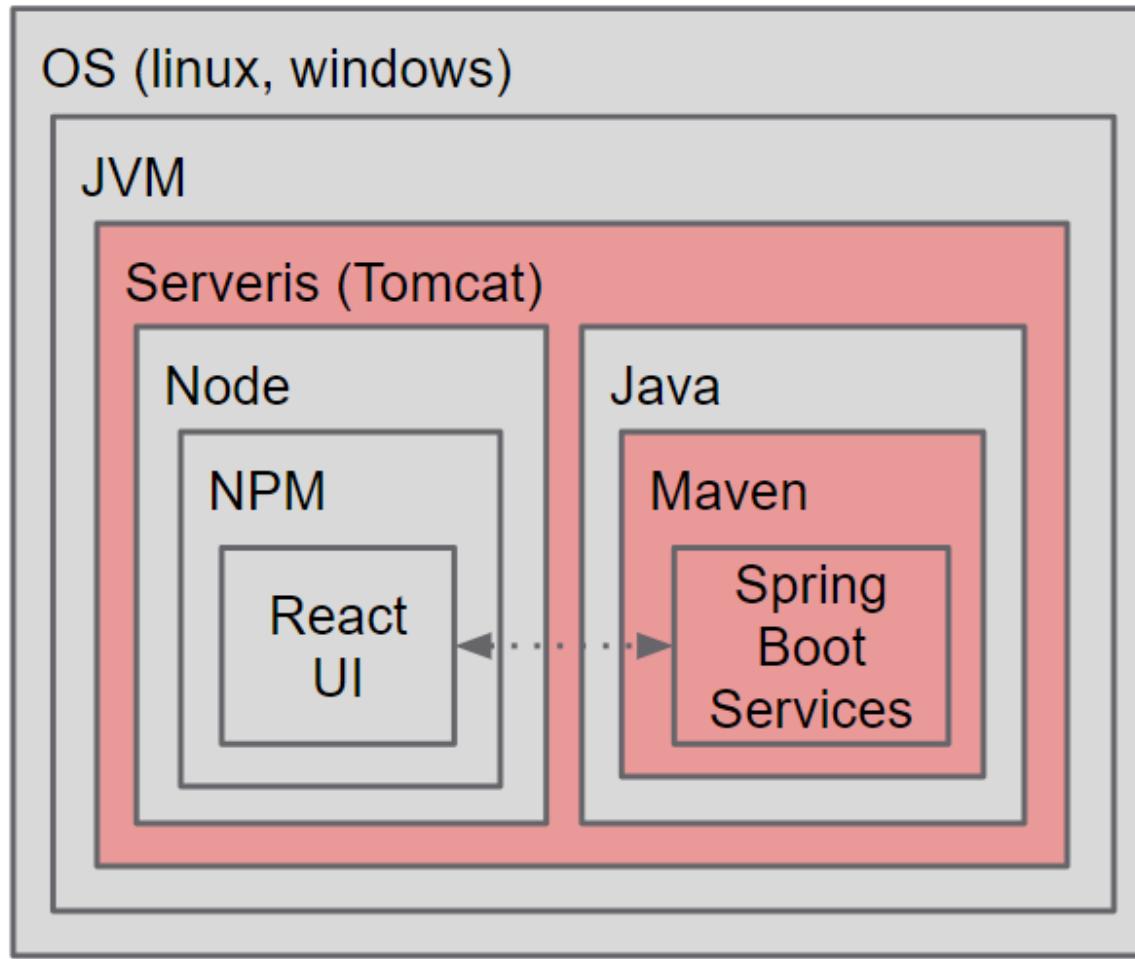
# KĄ JAU MOKAME IR KO DAR NE



# KĄ JAU MOKAME IR KO DAR NE



# KĄ JAU MOKAME IR KO DAR NE



# TURINYS

- Tomcat
  - WAR failai
- Maven
  - POM failai
  - Archetipai
- kaip sukurti Spring Boot aplikaciją
  - kaip prie jos prijungti React



# TOMCAT

## TOMCAT

- **Apache Tomcat** - tai žiniatinklio konteineris, kuris įgalina aplikacijų, sukurtų Java Servlet arba JavaServer Pages (JSP) pagrindu, veikimą. Dauguma moderniųjų žiniatinklio aplikacijų kūrimo karkasų yra įgyvendinti Java Servlet pagrindu: JavaServer Faces, Struts, Spring.
- Šis serveris gali būti naudojamas ir kaip HTTP serveris.



# APACHE TOMCAT KATALOGŲ STRUKTŪRA

## Katalogas      Aprašymas

---

/bin                Skriptai

---

/conf              Konfigūracija

---

/lib                Bibliotekos

---

/logs              Įvykių žurnalai

---

/temp             Laikų failų direktorija

---

/webapp          Aplikacijų direktorija

---

/work              Darbinė direktorija



# UŽDUOTIS 1 - PARUOŠIMAS DARBUI

```
$ wget http://apache.mirror.vu.lt/apache/tomcat/\  
> tomcat-8/v8.5.35/bin/apache-tomcat-8.5.35.tar.gz  
$ tar xvzf apache-tomcat-8.5.35.tar.gz  
$ cd apache-tomcat-8.5.35/bin  
$ ./startup.sh  
$ ./shutdown.sh - išjungimas
```

- Atidaryti: <http://localhost:8080>
- Tomcat serveris startuoja HTTP konektorių, kurio portas yra 8080 (portas leidžia skirtingoms aplikacijoms toje pačioje mašinoje dirbtis drauge vienu metu).



## UŽDUOTIS 1 - APLIKACIJŲ TVARKYKLĖ

- Apache Tomcat serveryje yra sudiegtą aplikacijų tvarkyklę, kuri leidžia atlikti tam tikrus aplikacijų administravimo darbus. Norint ją pasinaudoti, pirmiausia reikia atlikti saugumo nustatymus
- Faile /conf/tomcat-users.xml:

```
<user username="tomcat" password="tomcat" roles="manager-gui"/>
```

- Vartotojui tomcat, kurio slaptažodis tomcat, yra suteikta rolė manager-gui.



# APLIKACIJŲ TVARKYKLĖ

localhost:8080/manager/html

The Apache Software Foundation http://www.apache.org/

TM



## Tomcat Web Application Manager

Message: OK

**Manager**

List Applications    HTML Manager Help    Manager Help    Server Status

**Applications**

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

**Deploy**

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:



## APLIKACIJŲ TVARKYKLĖ

- Naudojantis šia tvarkykle galime atlikti tokias funkcijas:
  - Sustabdyti aplikaciją
  - Iš naujo paleisti aplikaciją
  - Panaikinti aplikaciją
  - Pridėti naują aplikaciją
  - Sunaikinti aktyvias sesijas
  - Pamatyti serverio parametrus



## UŽDUOTIS 2 - WAR PALEIDIMAS

- Atsisiukskite war failą
  - <https://tomcat.apache.org/tomcat-6.0-doc/appdev/sample/sample.war>
  - <http://stasauskas.lt/itpro2018/6-sample.war>
- Nueikite į <http://localhost:8080/manager/html> ir paleiskite (deploy) java aplikaciją sample.war
  - "Select WAR file to upload" -> "Deploy"
  - išbandykite, ar veikia, sustabdykite ir ištrinkite (undeploy)



# MAVEN

# MAVEN TURINYS

- Kas yra Maven
- Katalogų struktūra
- Darinio gyvavimo ciklas (angl. lifecycle)
- Projekto aprašas (angl. descriptor)
- Priklausomybės (angl. dependencies)
- Papildiniai (angl. plugins)
- Savybės (angl. properties)
- Profiliai (angl. profile)
- Archetipas



## KAS YRA MAVEN

- Darinio (angl. build) sukūrimo įrankis
- Standartizuota darinio sukūrimo infrastruktūra
- Priklausomybių (angl. dependency) valdymo įrankis
- Kokybės įrankis
- Atviro kodo Apache projektas
- Maven skirtas Java'ai taip, kaip NPM skirtas Node'ui



## PRIEŠ MAVEN (IKI 2003)

- javac komandos naudojimas projektų kompiliavimui
- Nuo IDE (Integrated development Environment) priklausomas projektų kompiliavimas
- GNU Make įrankis
- Ant (Another Neat Tool) įrankis



## MAVEN RAIDA

- Maven 1 (2003)
- Maven 2 (2005) - nesuderinamas su Maven 1.
- Maven 3 (2010) - suderinamas su Maven 2, stabilesnis, turintis daugiau funkcionalumo.



# MAVEN KONFIGŪRACIJA

- Windows nustatymai
  - %MAVEN\_HOME%\conf\settings.xml
  - Lokalios repositoriujos vieta: %UserProfile%\.m2\
- Linux nustatymai
  - /usr/local/maven/conf/settings.xml
  - Lokalios repositoriujos vieta: ~/ .m2/
- Nurodyti kitus Maven nustatymus:
  - mvn --settings=[PATH\_TO\_SETTINGS\_FILE]
- Nurodyti kitą lokalios repositoriujos vietą:
  - mvn -Dmaven.repo.local=/path/to/local/repo



# UŽDUOTIS 3 - MAVEN ĮDIEGIMAS

```
$ sudo apt-get install maven  
$ mvn -version  
Apache Maven 3.5.2
```



## MAVEN FILOSOFIJA

- Susitarimas per konfigūraciją (mažiau konfigūracijos)
- Lengvas darinio sukūrimo procesas
- Geriausių praktikų šablonai
- Nuoseklus darinio sukūrimas



# STANDARTINĖ MAVEN PROJEKTO STRUKTŪRA

Katalogas	Aprašymas
src	išeities kodas
src/main	pagrindinio artefakto išeities kodas
src/main/java	java išeities kodas
src/main/resources	nekompiliuojami resursai
src/main/webapp	žiniatinklio aplikacijos resursai
src/test	testavimui skirtas išeities kodas
src/test/java	testavimo java išeities kodas
src/test/resources	nekompiliuojami testavimo resursai
target	Maven darbinis katalogas
pom.xml	aprašo byla



## DARINIO GYVAVIMO CIKLAI

- maven paremtas darinio gyvavimo ciklais, skirtais artefaktų sukūrimui (pvz. jar byla, war byla) ir paskirstymui
- yra trys įtaisyti (angl. build-in) darinių gyvavimo ciklai:
  - Default - projekto darinio pagaminimas ir įdiegimas
  - Clean - projekto išvalymas
  - Site - projekto svetainės sugeneravimas
- kiekvienas iš išvardintų ciklų yra sudarytas iš skirtingo rinkinio fazių
- fazė reprezentuoja konkretų gyvavimo ciklo etapą



# PAGRINDINĖS “DEFAULT” FAZĖS

Fazė	Aprašymas
validate	patikrina ar projektas korektiškas ir visa reikiama informacija yra pasiekiamai
generate-sources	generuoja išeities kodą įtraukimui į kompiliavimą
generate-resources	generuoja resursus įtraukimui į paketą
compile	kompiliuoja projekto išeities kodą
test	testuoja sukompiliuotą kodą naudojant nurodytą testavimo karkasą
package	supakuoja sukompiliuotą kodą į nurodytą paskirstymo formatą (pvz. jar)
integration-test	pagal poreikį, įdiegia paketus į aplinką, kurioje vykdomi integraciniai testai
verify	vykdo patikrinimus ar paketai atitinka kokybės kriterijus
install	įdiegia paketus į lokalią repozitoriją, tam, kad kiti lokalus projektai galėtų naudoti priklausomybės ryšiu
deploy	galutinės paketų kopijos įdiegiamos į nutolusią repozitoriją, tam, kad pasidalinti su kitais PĮ kurėjais ir projektais



## “CLEAN” FAZĖS

### Fazė Aprašymas

---

pre-clean vykdo procesus, reikalingus prieš realų projekto valymą

---

clean pašalina visas bylas, kurios buvo sugeneruotos ankstesnio darinio sukūrimo metu

---

post-clean vykdo procesus, skirtus projekto valymo užbaigimui



## “SITE” FAZĖS

Fazė	Aprašymas
pre-site	vykdo procesus, reikalingus pieš realų projekto sveitainės generavimą
site	generuoja projekto sveitainės dokumentaciją
post-site	vykdo procesus, skirtus užbaigti sveitainės generavimui ir pasiruošti sveitainės įdiegimui
site-deploy	įdiegia sugeneruota sveitainės dokumentaciją į nurodytą žiniatinklio serverį



# MAVEN TIKSLŲ PAVYZDŽIAI

- Maven darinio sukūrimas yra vykdomas nurodant gyvavimo ciklo tikslus (angl. goals):

Komanda	Aprašymas
<code>mvn install</code>	jvykdo generate*, compile, test, package, integration-test ir install fazes
<code>mvn clean</code>	jvykdo tik clean gyvavimo ciklą
<code>mvn clean compile</code>	išvalo projektą ir jvykdo generate* ir compile fazes
<code>mvn compile install</code>	jvykdo generate*, compile, test, integration-test, package ir install fazes
<code>mvn test clean</code>	jvykdo generate*, compile, test fazes ir tada išvalo projektą



## POM - PROJEKTO APRAŠAS

- POM (angl. Project Object Model) saugomas byloje pom.xml
  - analogiškai NPM/Node saugomas package.json
- Saugo projekto metaduomenis:
  - vardas ir versija
  - supakavimo tipas
  - įrankių nuorodos (CI, SCM ir pan.)
  - priklausomybės
  - papildiniai
  - profiliai - alternatyvios darinio sukūrimo konfigūracijos



## POM - PROJEKTO APRAŠAS

- Naudojama XML kalba
- Vienas POM == vienas artefaktas
- POM ryšiai:
  - paveldėjimas
  - agregavimas



## POM - PROJEKTO VARDAS (GAV)

- Maven projektas unikaliai identifikuojamas naudojant:
  - groupId - sutartas projekto grupavimo identifikatorius (be tarpų/kablelių)
  - artifactId - projekto vardas (be tarpų/kablelių)
  - version - projekto versija
- GAV sintaksė yra groupId:artifactId:version

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.lds.training</groupId>
    <artifactId>maven-training</artifactId>
    <version>1.0</version>
</project>
```



# POM - SUPAKAVIMAS

- Darinio tipas identifikuojamas packaging elemente
- Nurodo Maven kaip pagaminti projekto darinį
- Supakavimo tipai:
  - pom, jar, war, ear, custom (pritaikytas)
  - tipas pagal nutilėjimą yra jar

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-training</artifactId>
    <groupId>org.lds.training</groupId>
    <version>1.0</version>
    <packaging>jar</packaging>
</project>
```



## POM - PROJEKTO PAVELDĘJIMAS

POM bylos gali paveldėti šią konfigūraciją: groupId, versiją, projekto konfigūraciją, priklausomybes, papildinius, profilius ir t.t.

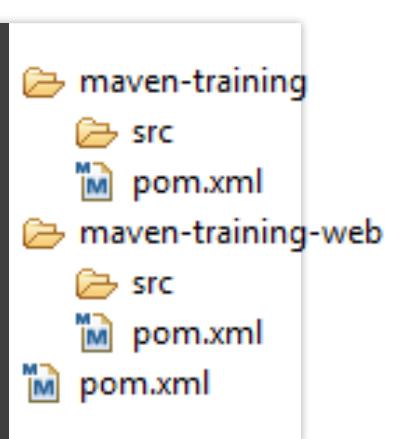
```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <parent>
        <artifactId>maven-training-parent</artifactId>
        <groupId>it.akademija.training</groupId>
        <version>1.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-training</artifactId>
    <packaging>jar</packaging>
</project>
```



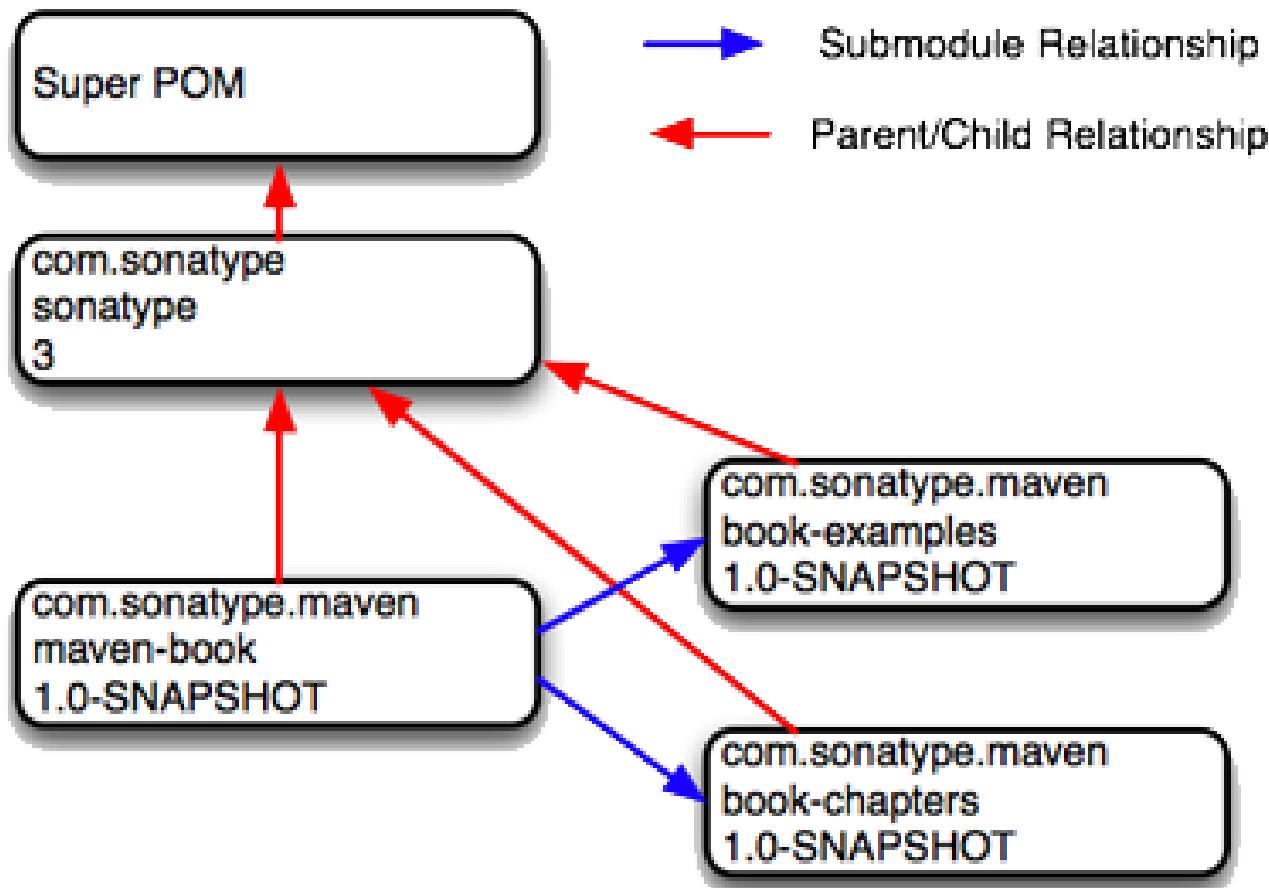
# POM - MODULIAI

- Maven turi daugelio modulių palaikymą
- Kiekvienas Maven projektas sukuria 1 pagrindinj artefaktą
- Tėvo (angl. parent) POM naudojamas modulių grupavimui

```
<project>
    ...
    <packaging>pom</packaging>
    <modules>
        <module>maven-training</module>
        <module>maven-training-web</module>
    </modules>
</project>
```



# SUPER POM IR PAVELDĘJIMO PAVYZDYS



## MAVEN PRIKLAUSOMYBĖS

- Maven pakeitė Java priklausomybių valdymą - nebereikia saugoti bibliotekų išeities kodo valdymo sistemoje (SCM) ar panašiai
- Pasiūlyta Maven repozitorijos (angl. repository) sąvoka. Sukurta Maven Central bendruomenės repozitorija
- Pasiūlyta tranzityvios priklausomybės (angl. transitive dependency) sąvoka
- Dažnai įtraukiami išeities kodo ir javadoc artefaktai



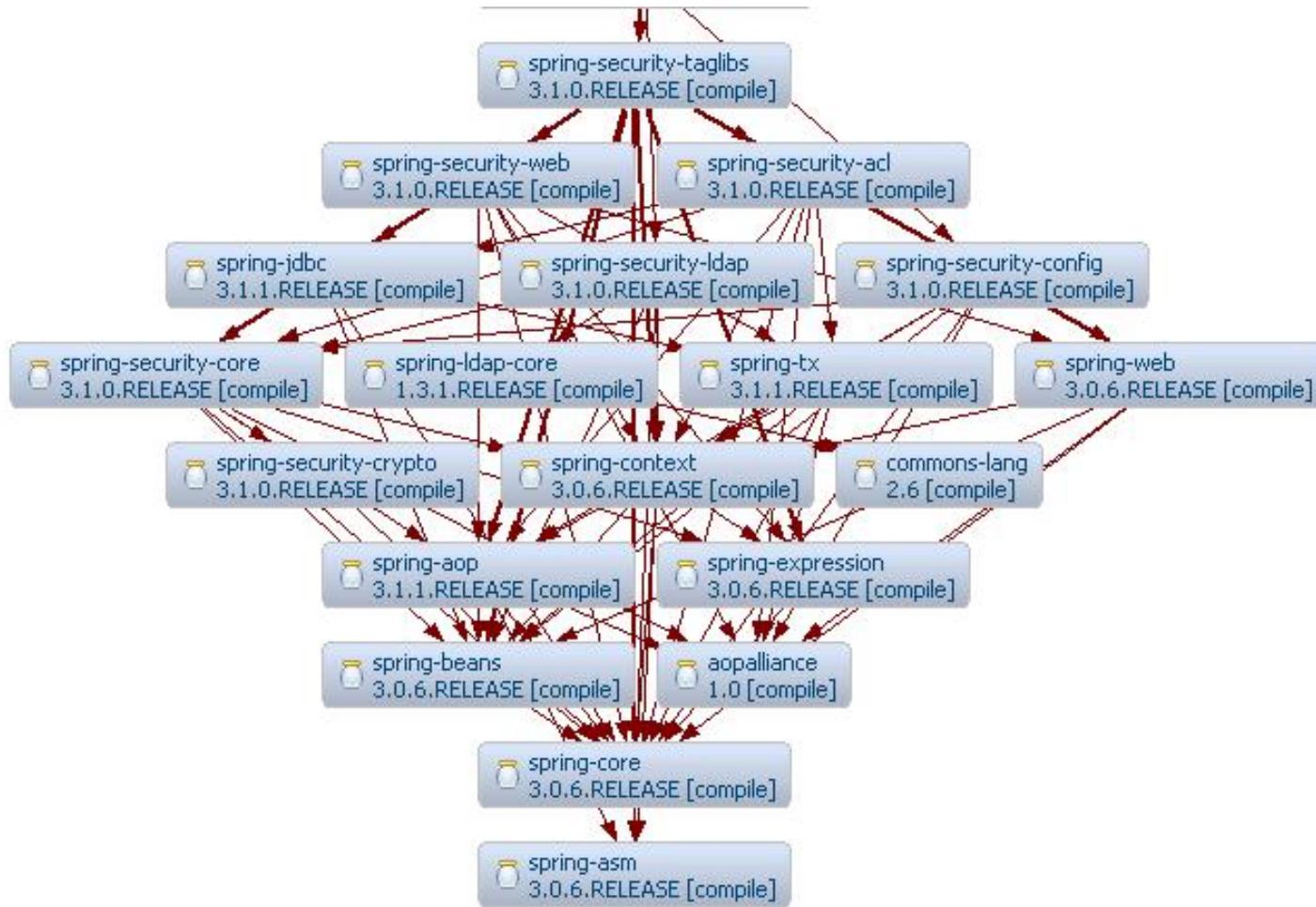
# PRIDĒTI PRIKLAUSOMYBĘ

- Priklausomybės aprašą sudaro:
  - GAV (groupId, artifactId, version)
  - Galiojimo sritis (angl. scope) - compile, test, provided.  
Pagal nutylėjimą naudojama compile
  - Tipas - jar, pom, war, ear, zip ir t.t. Pagal nutylėjimą naudojamas jar
- pom.xml:

```
<project>
...
<dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId> servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
</project>
```



# SPRING PRIKLAUSOMYBIŲ PAVYZDYS



## MAVEN REPOZITORIJA

- Priklausomybės yra parsiunčiamos iš repozitorijų naudojant http protokolą
- Parsiustos priklausomybės yra išsaugomos lokalioje repozitorijoje (pvz. \${user.home}/.m2/repository)
- Repozitorijoje naudojama {groupId}/{artifactId}/{version}/{artifactId}-{version}.jar katalogų struktūra, o groupId ‘.’ yra pakeičiamas ‘/’
- Maven Central yra pagrindinė Maven bendruomenės repozitorija <http://repo1.maven.org/maven2>



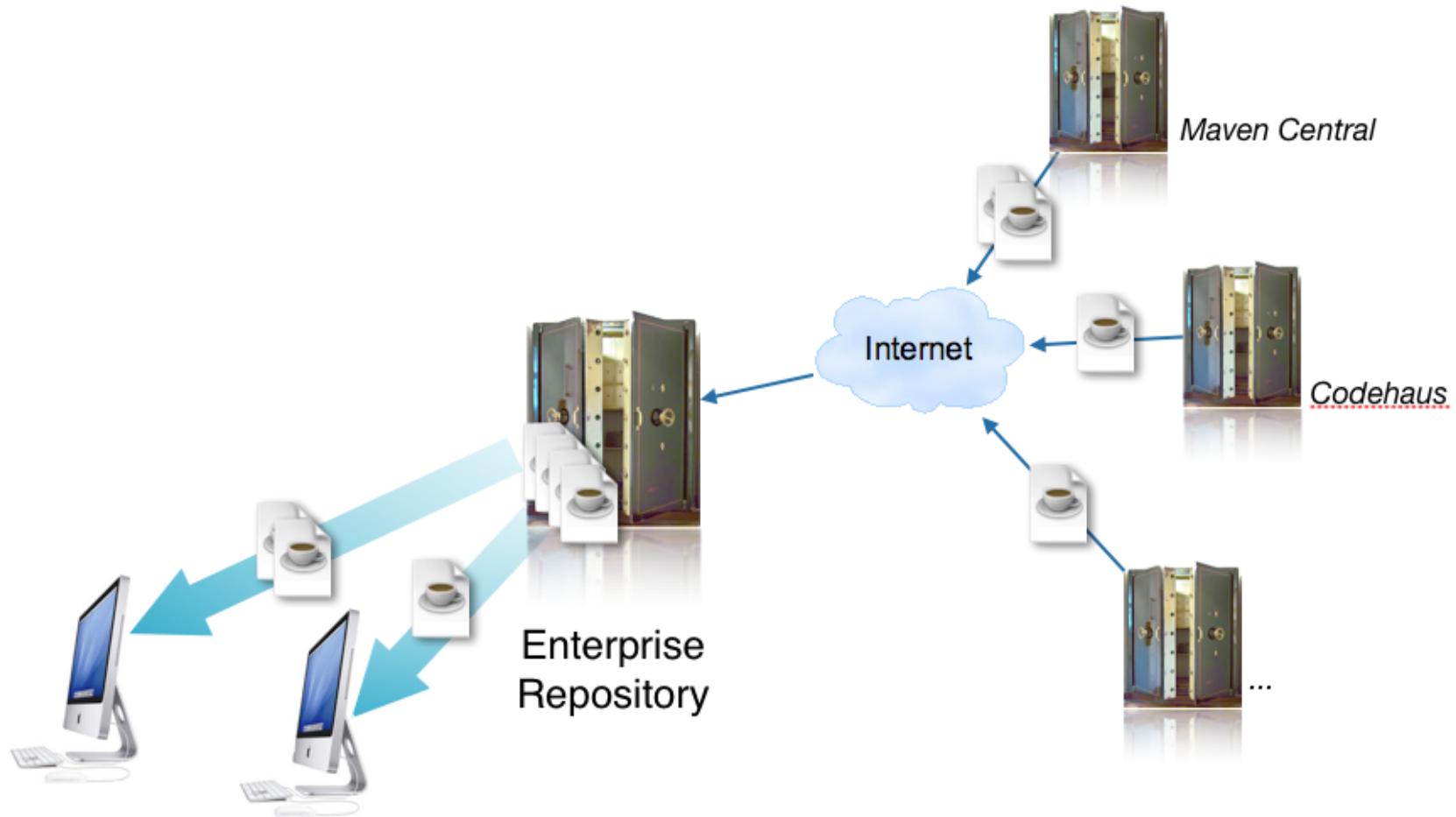
# PRIDĒTI REPOZITORIJĀ

- Repozitorijos aprašomos POM
- Repozitorijos gali būti paveldētos iš tēvo POM
- Momentinių kopijų (angl. snapshot) parsiuntimas gali būti kontroliuojamas
- pom.xml

```
<project>
...
<repositories>
    <repository>
        <id>lds-main</id>
        <name>LDS Main Repo</name>
        <url>http://code.eisgroup.com/nexus/content/groups/main-repo</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
</project>
```



# MAVEN REPOZITORIJŲ PAVYZDYS



## TRANZITYVIOS PRIKLAUSOMYBĖS

- Tranzityvi priklausomybė yra tokia priklausomybė, kuri turi būti įtraukta, kai priklausomybę deklaruojantis projektas yra pats kito projekto priklausomybė:
  - ProjectA priklauso nuo ProjectB
  - Jei ProjectC priklauso nuo ProjectA, tai ProjectB yra automatiškai įtraukiama
- Tik compile ir runtime galiojimo sritys yra tranzityvios
- Tranzityvios galiojimo sritys yra valdomos naudojant:
  - Pašalinimus (angl. exclusions)
  - Neprivalomą (angl. optional) deklaraciją



# PRIKLAUSOMYBĖS PAŠALINIMAS

- Tranzityvi priklausomybė pašalinama naudojant exclusions elementą:

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>3.0.5.RELEASE</version>
      <exclusions>
        <exclusion>
          <groupId>commons-logging</groupId>
          <artifactId>commons-logging</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>
</project>
```



# NEPRIVALOMA PRIKLAUSOMYBĖ

- Neskleidžia tranzityviai prilausomybės:
  - ProjectA turi neprivalomą prilausomybę nuo ProjectB
  - Jei ProjectC priklauso nuo ProjectA tai ProjectB nebus automatiškai ištraukiamas

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>3.0.5.RELEASE</version>
      <optional>true</optional>
    </dependency>
  </dependencies>
</project>
```



# PRIKLAUSOMYBIŲ VALDYMAS 1

- Java neleidžia naudoti kelių versijų vienu metu
- Ką daryti jei versijos persikerta?
  - Leisti Maven nuspести, kuria versiją naudoti - sunkiau nuspėjamas rezultatas
  - Valdyti versijas rankiniu būdu
- Priklausomybių versijos valdomos naudojant dependencyManagement elementą
- Kiti panaudojimai:
  - Leisti tévo POM valdyti versijas
  - Suvienodinti pašalinimus



# PRIKLAUSOMYBIŲ VALDYMAS 2

- pom.xml priklausomybių be versijos pavyzdys

```
<project>
...
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.0.5.RELEASE</version>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        </dependency>  <!-- Nenurodoma versija! -->
    </dependencies>
</project>
```



## MAVEN PAPILDINIAI

- Išplečia Maven funkcionalumą
- Identifikacijai naudojamas GAV (groupId, artifactId, version)
- Papildinio naudojimo būdai:
  - Prikabinti prie darinio kūrimo gyvavimo ciklo
  - Iškvesti autonomiškai (angl. standalone)



## PRIKABINIMAS PRIE GYVAVIMO CIKLO

- Leidžia papildinij įvykdyti kaip Maven darinio kūrimo dalį
- Papildinio aprašymo elementai naudojami vykdymo konfigūravimui:
  - Phase
  - Goal
  - Configuration



# MAVEN PAPILDINIO PAVYZDYS 1

- pom.xml yra prikabintas papildinys maven-enforcer-plugin

```
<project>
  ...
  <build><plugins><plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-enforcer-plugin</artifactId>
    <version>1.0</version>
    <configuration> ... </configuration>
    <executions>
      <execution>
        <id>execute</id>
        <phase>validate</phase>
        <goals><goal>enforce</goal></goals>
          <configuration> ... </configuration>
        </execution>
      </executions>
    </plugin></plugins></build>
  </project>
```



## PAPILDINIO VALDYMAS

- Leidžia sukonfigūruoti sekančius papildinio elementus, jo nevykdant:
  - Version
  - Configuration
  - Executions
- Vykdymui naudojamas:
  - įprastas papildinio įrašas
  - papildinio autonominė komanda



# PAPILDINIO VALDYMO PAVYZDYS

- pom.xml papildinio konfigūracija maven-enforcer-plugin

```
<project><build>
<pluginManagement>
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-enforcer-plugin</artifactId>
    <version>1.0</version>
    <configuration>
      ...
      <ignoreCache>true</ignoreCache>
    </configuration>
  </plugin>
</plugins>
</pluginManagement>
<plugins><plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-enforcer-plugin</artifactId>
</plugin></plugins>
</build></project>
```



## PAPILDINIO PAVELDĘJIMAS

- Papildinys paveldi pluginManagement konfigūraciją
- Papildinys ir pluginManagement gali paveldėti konfigūraciją iš tévo POM
- Taip pat leidžiama pakeisti paveldétą konfigūraciją



## PAPILDINIO AUTONOMINIS ĮVYKDYMAS

- Papildinys gali būti išviečiamas naudojant komandinę eilutę:
  - `GroupId:ArtifactId:Version:Goal`
  - Naudojama `pluginManagement` konfigūracija
  - Konfigūracija gali būti nurodyta saybėmis

```
$ mvn org.apache.maven.plugins:maven-enforcer-plugin:1.0:enforce
```

Jei papildinys sukonfigūruotas POM arba `settings.xml` tai iškvietimas gali būti sutrumpintas:

```
$ mvn enforcer:enforce
```



## MAVEN SAVYBĖS

- Savybės tai tarsi klijai, kurie suriša konfigūraciją
- Savybes galima nurodyti sekančiose vietose:
  - elementas POM byloje
  - Sisteminės savybės
  - POM struktūra
- Savybių reikšmę galima panaudoti įvairiausiose vietose:
  - Versijos konfigūravimui
  - Papildinio konfigūravimui
  - Resursų filtravimui
- Savybėse nurodomos tik primityvios reikšmės



# POM SAVYBĖS

- pom.xml nurodytos savybės ir jų panaudojimas

```
<project>
...
<properties>
    <skipEnforcer>true</skipEnforcer>
    <enforcerVersion>1.0</enforcerVersion>
</properties>
<build><plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-enforcer-plugin</artifactId>
        <version>${enforcerVersion}</version>
        <configuration>
            <skip>${skipEnforcer}</skip>
        </configuration>
    </plugin>
</plugins></build>
</project>
```



## SISTEMINĖS SAVYBĖS

Nurodomos komandinėje eilutėje panaudojus “-D”:

```
$ mvn clean install -Dmaven.test.skip=true -DskipTests=true
```

Sisteminės savybės turi viršenybę prieš POM savybės



# POM STRUKTŪROS SAVYBĖS

- Savybės gali būti paveldėtos iš POM struktūros
- POM elementai yra savybių raktai:
  - Išraiška \${project.version}

```
<project><version/></project>
```

- Išraiška \${project.artifactId}

```
<project><artifactId/></project>
```

- Išraiška \${project.build.sourceDirectory}

```
<project><build><sourceDirectory/></build></project>
```



## POM STRUKTŪROS SAVYBĖS

- Specialios savybės:
  - \${basedir} - einamojo projekto katalogas
  - \${maven.build.timestamp} - darinio kūrimo pradžios laikas



# SAVYBIŲ PAVELDĖJIMAS IR PERKROVIMAS

```
<project> <!-- čia yra tėvinis projektas -->
    ...
    <properties>
        <skipTests>true</skipTests>
        <skipEnforcer>${skipTests}</skipEnforcer>
    </properties>
</project>

<project>
    <parent>
        ... <!-- tėvinio projekto GAV -->
    </parent>
    <properties>
        <skipTests>false</skipTests>
    </properties>
</project>
```



# RESURSU FILTRAVIMAS

- Projekto resursai gali naudoti savybes
- Resursai filtruojami process-resources fazėje
- Filtravimas gali buti išjungtas nurodytiems resursų katalogams

```
<project>
    ...
    <properties>
        <someProperty>SomeValue</someProperty>
    </properties>
</project>
```

Tekstinė byla  
/src/main/resources kataloge:  
\${someProperty}

Filtravimas

Tekstinė byla  
/src/main/resources kataloge:  
SomeValue



## MAVEN PROFILIAI

- Leidžia aktyvuoti rinkinj alternatyvių konfigūracijų
- Gali būti naudojami nurodyti:
  - savybes
  - priklausomybes
  - papildinius
  - kita
- Paveldi ir išplėčia bazinę konfigūraciją



# PROFILIO PAVYZDYS 1

```
<project>
    ...
    <profiles>
        <profile>
            <id>enforcer</id>
            <activation/>
            <properties>
                <enforcer.skip>false</enforcer.skip>
            </properties>
        </profile>
    </profiles>
</project>
```



# PROFILIO PAVYZDYS 2

```
$ mvn exec:exec - vykdo Java programą atskirame procese.  
$ mvn exec:java - vykdo Java programą toje pačioje VM, kaip ir Maven  
$ mvn exec:java -Dexec.mainClass="com.example.Main" \  
> [-Dexec.args="argument1"] ...
```

```
<profiles><profile>  
  <id>run</id>  
  <activation><activeByDefault>true</activeByDefault></activation>  
  <build><plugins><plugin>  
    <groupId>org.codehaus.mojo</groupId>  
    <artifactId>exec-maven-plugin</artifactId>  
    <executions>  
      <execution>  
        <goals><goal>java</goal></goals>  
        <phase>runtime</phase>  
      </execution>  
    </executions>  
    <configuration>  
      <mainClass>ClassWithTheMain</mainClass>  
    </configuration>  
  </plugin></plugins></build>  
</profile></profiles>
```



## PROFILIO AKTYVAVIMAS

- Profilis gali būti aktyvuotas:
  - pagal nutylėjimą
  - tiesiogiai pagal profilio vardą
  - priklausomai nuo savybės
  - priklausomai nuo operacinės sistemos
  - priklausomai nuo bylos egzistavimo



## AKTYVAVIMAS KOMANDINĖJE EILUTĖJE

```
$ mvn clean install -P enforcer
```

Keletas profilio identifikatorių yra atskirami kableliu

```
$ mvn clean install -P enforcer,kitas-profilis
```



## ARCHETIPAS

- Archetipas yra Maven projektų šablonų priemonių komplektas
- Bendresnis archetipo apibrėžimas būtų autentiškas šablonas ar modelis, kurį naudojant yra gaminami kiti tos pačios rūšies dalykai



# MAVEN PROJEKTO SUKŪRIMAS

- Paprasto projekto sukūrimas:

```
$ mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app \
> -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

- Arba galima atsakinėti į klausimus terminale:

```
$ mvn archetype:generate
```

- Eclipse IDE projekto failų sukūrimas:

```
$ mvn eclipse:eclipse
```

- Idea IDE projekto failų sukūrimas:

```
$ mvn idea:idea
```



## NAUDINGOS NUORODOS

- <http://maven.apache.org/users/index.html>
- <http://maven.apache.org/pom.html>
- <http://maven.apache.org/guides/getting-started/index.html>
- <http://search.maven.org/>
- <https://github.com/sonatype/maven-example-en>



# UŽDUOTIS 4 - ARCHETIPAS

```
$ mvn archetype:generate -DgroupId=lt.mokymai \
-DartifactId=SecondMavenProject -Dpackage=lt.mokymai.maven \
-Dversion=1.0-SNAPSHOT \
-DarchetypeArtifactId=maven-archetype-quickstart
$ cd SecondMavenProject
```

```
└ pom.xml
  └ src
    └ main
      └ java
        └ lt
          └ mokymai
            └ maven
              └ App.java
    └ test
      └ java
        └ lt
          └ mokymai
            └ maven
              └ AppTest.java
```

# UŽDUOTIS 4 - ECLIPSE

- Sukurti Eclipse projekto failus:

```
$ mvn eclipse:eclipse // idea:idea
```

- Peržiūrėti SecondMavenProject projekto katalogą:

```
$ ls -a
.  ..  .classpath  pom.xml  .project  src
```

- Projektą importuoti į Eclipse
  - senesnėse eclipse: Window > Preferences -> Java > Build Path > Classpath Variables -> New M2\_REPO = /home/username/.m2/repository
  - kodo atitikimas standartui: Window > Preferences -> Java Compiler -> Compliance -> 1.8

## UŽDUOTIS 4 - KOMPILIAVIMAS IR VYKDYMAS

```
$ mvn clean compile
```

- Panaudoti exec:java papildinį lt.mokymai.maven.App klasės įvykdymui:

```
$ mvn exec:java -Dexec.mainClass="lt.mokymai.maven.App"
[INFO] --- exec-maven-plugin:1.3.2:java (default-cli) @ SecondMavenProject ---
[WARNING] Warning: killAfter is now deprecated. Do you need it ? Please comment on MEXEC-
Hello World!
[INFO] -----
[INFO] BUILD SUCCESS
```



## UŽDUOTIS 4 - KOMPILIAVIMAS IR VYKDYMAS

- jeigu nesikompiliuos, galbūt neturite jdk 8 virtualioje mašinoje

```
$ java -version
openjdk version "10.0.2" 2018-07-17
$ sudo apt-get install openjdk-8-jdk
$ sudo update-alternatives --list java
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
$ sudo update-alternatives --set java \
    /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
$ java -version
openjdk version "1.8.0_162"
```



## UŽDUOTIS 4 - TESTAVIMAS

```
$ mvn clean test
-----
 T E S T S
-----
Running lt.mokymai.maven.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.018
Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```



## UŽDUOTIS 4 - CLEAN IR COMPILE

```
$ mvn clean  
$ ls -a  
. .. .classpath pom.xml .project .settings src
```

- Pastaba. Pagrindiniame projekto kataloge nėra target katalogo.

```
$ mvn clean compile  
$ ls -a  
. .. .classpath pom.xml .project .settings src target  
$  
$ ls -a target/  
. .. classes
```

- Pastaba. Pagrindiniame projekto kataloge sukurtas target katalogas.



## UŽDUOTIS 4 - TEST

```
$ mvn clean test  
$ ls -a  
. .. .classpath pom.xml .project .settings src target  
  
$ ls -a target/  
. .. classes surefire surefire-reports test-classes
```



## UŽDUOTIS 4 - PACKAGE

```
$ mvn clean package
$ ls -a
. .. .classpath pom.xml .project .settings src target
$ ls -a target/
. .. classes maven-archiver SecondMavenProject-1.0-SNAPSHOT.jar
surefire surefire-reports test-classes
$ ls ~/.m2/repository/
antlr asm backport-util-concurrent biz classworlds commons-cli
commons-collections commons-io commons-lang dom4j jdom jline junit net org oro xml-api
```

- Pastaba: target kataloge sukurtas SecondMavenProject-1.0-SNAPSHOT.jar maven projekto artefaktas, tačiau jis nėra perkeltas į lokalią maven repozitoriją (nėra katalogo ~/.m2/repository/lt/mokymai/)



# UŽDUOTIS 4 - INSTALL

```
$ mvn clean install
$ ls -a target/
. .. classes maven-archiver SecondMavenProject-1.0-SNAPSHOT.jar
surefire surefire-reports test-classes
$ ls ~/.m2/repository/
antlr lt backport-util-concurrent classworlds commons-collections
commons-lang jdom junit net oro asm biz commons-cli commons-io dom4j jline org xml-api
$ ls ~/.m2/repository/lt/mokymai/SecondMavenProject/
1.0-SNAPSHOT maven-metadata-local.xml
```

- Pastaba: target kataloge sukurtas SecondMavenProject-1.0-SNAPSHOT.jar maven projekto artefaktas ir jis yra perkeltas į lokalią maven repozitoriją.



# UŽDUOTIS 5 - MAVEN MULTI PROJEKTAI

- Parsiusti pavyzdinį maven projekta

```
$ wget http://books.sonatype.com/mvnex-book/mvnex-examples.zip
```

- Išarchyvuoti ir pasirinkti projekto katalogą:

```
$ unzip mvnex-examples.zip  
$ cd mvnexbook-examples-1.0/ch-multi-spring
```

- Pagaminti darinį (angl. build):

```
$ mvn clean install
```

- Peržiūrėti projekto aprašus (pom.xml bylas)
- Importuoti projektus (mvnexbook-examples-1.0/ch-multi-spring) į Eclipse.



## UŽDUOTIS 6 - JAVA KLASĖS KAIP SERVERIS

- Sukurti naują Maven projektą
  - galima ir su maven-archetype-quickstart
- Perkelti Java kurso praktikos klasses į projektą arba sukurti naujas klasses
- Pagaminti projekto darinį
- Įvykdyti pagrindinę Javą klasę main Maven priemonėmis
- Įvykdyti pagrindinę Javą klasę iš Eclipse aplinkos

Video pvz: <http://youtu.be/8hvtZxAlNyw>



## UŽDUOTIS 6 - JAVA KLASĖS KAIP SERVERIS

- Sukurti antrą maven projektą tokiu pačiu principu ir jį panaudoti anksčiau kurtame
  - pridėti antrojo projekto pom.xml GAV aprašą prie pirmojo projekto dependencies
- dabar pirmasis projektas naudoja antrajį
  - perkeliame visas java klasses iš pirmojo į antrajį
- Paleidžiame mvn clean install antrajam
- Tuomet paleidžiame pirmajam
- Dabar pabandome paleisti Main klasę pirmajame
  - nors pirmasis pats neturi klasės, bet tranzityviai gauna jas iš antrojo ir main metodas pasileidžia



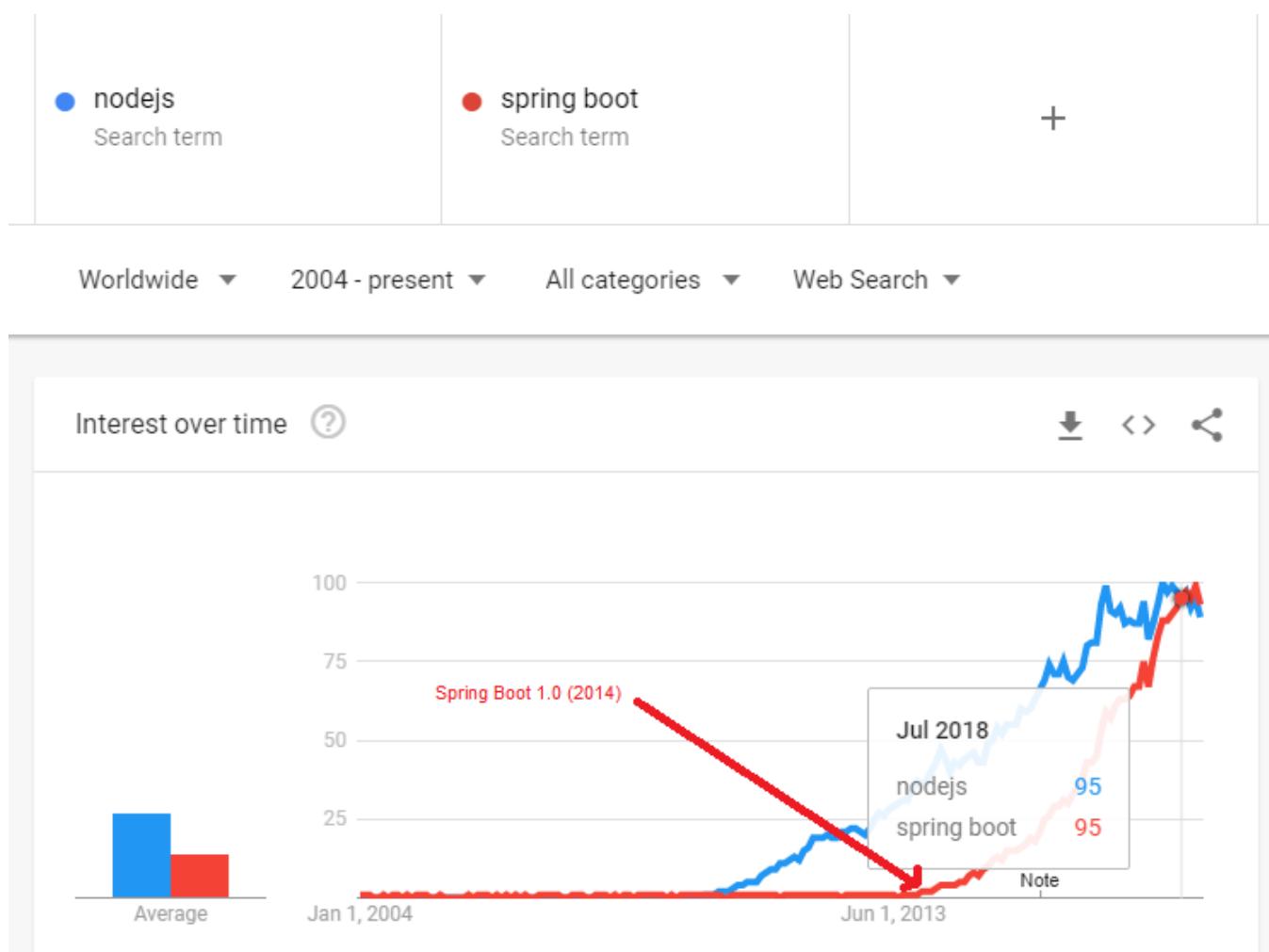
# SPRING BOOT APLIKACIJA

# SPRING BOOT APLIKACIJA

- kodėl Spring Boot?
- kas yra Servlet
  - java klasės
  - technologija
  - specifikacija
- kas yra Spring
  - java klasės
  - biblioteka
- kas yra Spring Boot
  - java klasės
  - Spring biblioteka



# REST KARKASŲ KOVOS



# REST KARKASŲ KOVOS

- Kodėl naudojamas Spring Boot ?
  - Node.js per anksti: didelės kompanijos nori stabilumo
  - jokių didelių privalumų: ką gali Node.js, tą gali ir java
  - ekosistema: java išvystyta, daug bibliotekų, palaiko VISAS platformas, DB, enterprise lygio palaikymas
    - jms/webservisai/rest/big data
  - Java turi statinius tipus
  - Node.js servisai yra vienoje gijoje (thread)
    - t.y. nepalaiko multi-threading
  - Javascript einamasis palaikymas yra košmaras
  - Javascript sunkiai dokumentuojamas

# SERVLET

- Pagal JavaDoc aprašymą:

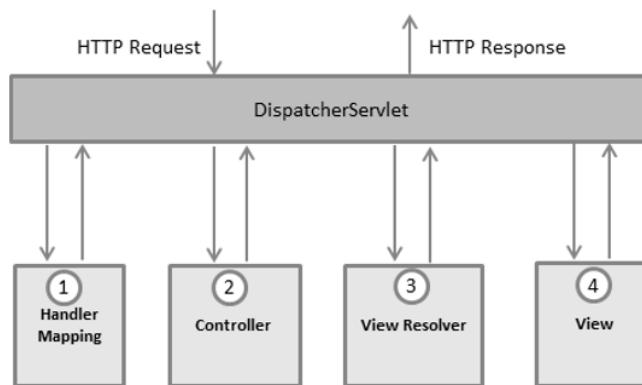
*Servletas - tai maža Java programa, kuri veikia žiniatinklio serveryje. Servletai gauna ir apdoroja užklausas, gautas iš žiniatinklio klientų, dažniausiai naudojant HTTP protokolą”.*

- Tai yra klasė, kuri realizuoja javax.servlet.Servlet interfeisą



# SERVLET

- skirtas išplėsti serverio galimybes
- gali būti sukonfigūruoti keli servletai skirtiniems keliams
  - /kelias1 -> Servlet1
  - /kelias2 -> Servlet2
- Spring Boot servlet'as sugeba apdoroti HTTP užklausas, suprasti REST tipo užklausas ir atiduoti statinj turinj



## UŽDUOTIS 7 - SPRING BOOT ARCHETIPAS

- kol kas kūrėme praktiškai tuščią projektą, su pačia paprasčiausia aplikacija, kuri nedirba per tinklą
  - java klasė su main metodu
- toliau sukursime beveik pilnai paruoštą naudoti pavyzdinę Spring Boot aplikaciją

```
$ mvn archetype:generate -DgroupId=it.akademija \
-DartifactId=hello-world-calc \
-DarchetypeGroupId=am.ik.archetype \
-DarchetypeArtifactId=spring-boot-blank-archetype \
-DarchetypeVersion=1.0.6 -DinteractiveMode=false
```



# UŽDUOTIS 7 - SPRING BOOT ARCHETIPAS

- Maven nustatymai koduotei UTF-8 kodui bei generuotiemis failams ir Java 8

```
<properties>
    <!-- Kodas užkoduotas universalia koduote UTF-8 -->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <!-- Kodas skirtas Java 8 -->
    <java.version>1.8</java.version>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

- dėti į pom.xml



# UŽDUOTIS 7 - SPRING BOOT ARCHETIPAS

- Paruošiame naudojimui IDE

```
$ mvn eclipse:eclipse
```

- senesnėse eclipse:
  - Window > Preferences -> Java > Build Path > Classpath Variables -> New M2\_REPO =  
/home/<username>/.m2/repository
- kodo atitikimas standartui
  - Window > Preferences -> Java Compiler -> Compliance -> 1.8



## UŽDUOTIS 7 - SPRING BOOT ARCHETIPAS

- Paruošiame darinį (build+tests)

```
$ mvn clean install  
$ mvn test
```

- Paleidžiame Spring Boot aplikaciją

```
$ mvn spring-boot:run -Drun.jvmArguments=''-Dserver.port=8081'
```

- Pastaba: jeigu prieš tai paleidome Tomcat, 8080 portas jau užimtas, todėl naudojame 8081 arba bet kurį laisvą



## UŽDUOTIS 7 - SPRING BOOT ARCHETIPAS

- Patikriname ar veikia naršyklėje
  - <http://localhost:8081/calc?left=1&right=2>
  - pastaba: serveris ir yra viena aplikacija
- Tokios JAR aplikacijos negalime įdėti į Tomcat serverį
  - reikalingas WAR failas
- Tai gal tiesiog pakeisti/įrašyti pom.xml packaging?

```
<packaging>war</packaging>
```

- nesuveiks.. reikia perkonfigūruoti Spring Boot



# UŽDUOTIS 7 - SPRING BOOT ARCHETIPAS

- pom.xml pridedame priklausomybę

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

- App.java perdarome taip, kad extendint'ų  
SpringBootServletInitializer Spring Boot servlet'a

```
@SpringBootApplication
public class App extends SpringBootServletInitializer {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
    @Override
    protected SpringApplicationBuilder configure(
        SpringApplicationBuilder builder) {
        return builder.sources(App.class);
    }
}
```



## UŽDUOTIS 7 - SPRING BOOT ARCHETIPAS

- Paleidus mvn clean install target'e atsiras .war
  - įdėkite jį į anksčiau atsisiųstą tomcat serverį
  - ar vis dar veikia /calc kalkuliatorius?
- Paprasčiau ir greičiau galima paleisti įdėtinį tomcat7

```
$ mvn org.apache.tomcat.maven:tomcat7-maven-plugin:2.2:run-war \
> -Dmaven.tomcat.port=8081
```

- spring-boot:run aplikaciją padaro serveriu, o tomcat7:run-war aplikaciją paleidžia serveryje kaip vieną iš aplikacijų
- todėl prieiname ne ...8081/calc , o ...8081/<appName>/calc



## UŽDUOTIS 8 - REACT PRIJUNGIMAS

- Pirmiausia reikia paruošti React aplikaciją. Tam, kad veiktų ir Spring Boot, ir Tomcat, turime į package.json pridėti

```
"homepage": "./"
```

- Įprastai čia turėtų būti tikros svetainės adresas, pvz.  
<http://svetaine.lt/kelias/iki/jos>



## UŽDUOTIS 8 - REACT PRIJUNGIMAS

- Spring Boot aplikacijoje pervadiname
  - src/main/resources/templates katalogą į
  - src/main/resources/public ir jo turinį ištriname
- Katalogo build turinį iš React aplikacijos tiesiog nukopijuojame į Spring Boot aplikacijos src/main/resources/public katalogą



## UŽDUOTIS 8 - REACT PRIJUNGIMAS

- Kol kas kelią iki '/' blokuoja java kodas, todėl iš failo `src/main/java/it/akademija/HelloController.java` turime surasti ir ištrinti šį kodą:

```
@RequestMapping("/")
String hello() {
    return "Hello World!";
}
```

- Jį ištrynus, React aplikacija veiks po keliu /, bet tuo pačiu veiks ir /calc servisas
- Tuomet neveiks HelloControllerTest testas: jį reikia ištrinti, arba laikinai netestuoti su mvn -DskipTests

## UŽDUOTIS 8 - REACT PRIJUNGIMAS

- Sukonfigūruojame Spring Boot pom.xml, kad statiniai resursai atsinaujintų neperkrovus puslapio

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <addResources>true</addResources>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- Pastaba: tai veiks tik su Spring Boot, bet ne Tomcat
- Paleidimui naudokite tas pačias jau išmoktas komandas



# KITOJE PASKAITOJE

Spring. Maven priklausomybių migracija



# IŠ PRAEITOS PASKAITOS

- jeigu java serveryje įkėlus projektą rodomas tuščias baltas langas, bent ant kito port'o aplikacija veikia
  - pasiūlymas užkomentuoti index.js service worker import'ą bei paskutine eilutę, kur service worker atlieka unregister
  - aplikacijos kūrimo etape užsikešuoja ir neteisingai iš cache nuskaito



# IŠ PRAEITOS PASKAITOS

- jeigu neteisingai suderinote aplinką, tuomet dar nors ir java -version rodo 8 java versiją, tačiau:

```
$ mvn -version
> Apache Maven 3.5.2
> Java version: 10.0.2, vendor: Oracle Corporation
> Java home: /usr/lib/jvm/java-10-oracle
```

- turite pakeisti java ir maven'ui. Ubuntu sistemoje versiją pakeisti galima j .bashrc isidėjus:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
```



# IŠ PRAEITOS PASKAITOS

- praeitą kartą generavote keletą aplikacijų
- toliau dirbsite su dviem projektais
  - su įprastu quickstart maven projektu
  - vis atnaujinsite Spring Boot projektą
- iš pradžių spring prijungsite ir atliksite užduotis iš quickstart šablono pasidarytoje aplikacijoje/projekte





**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

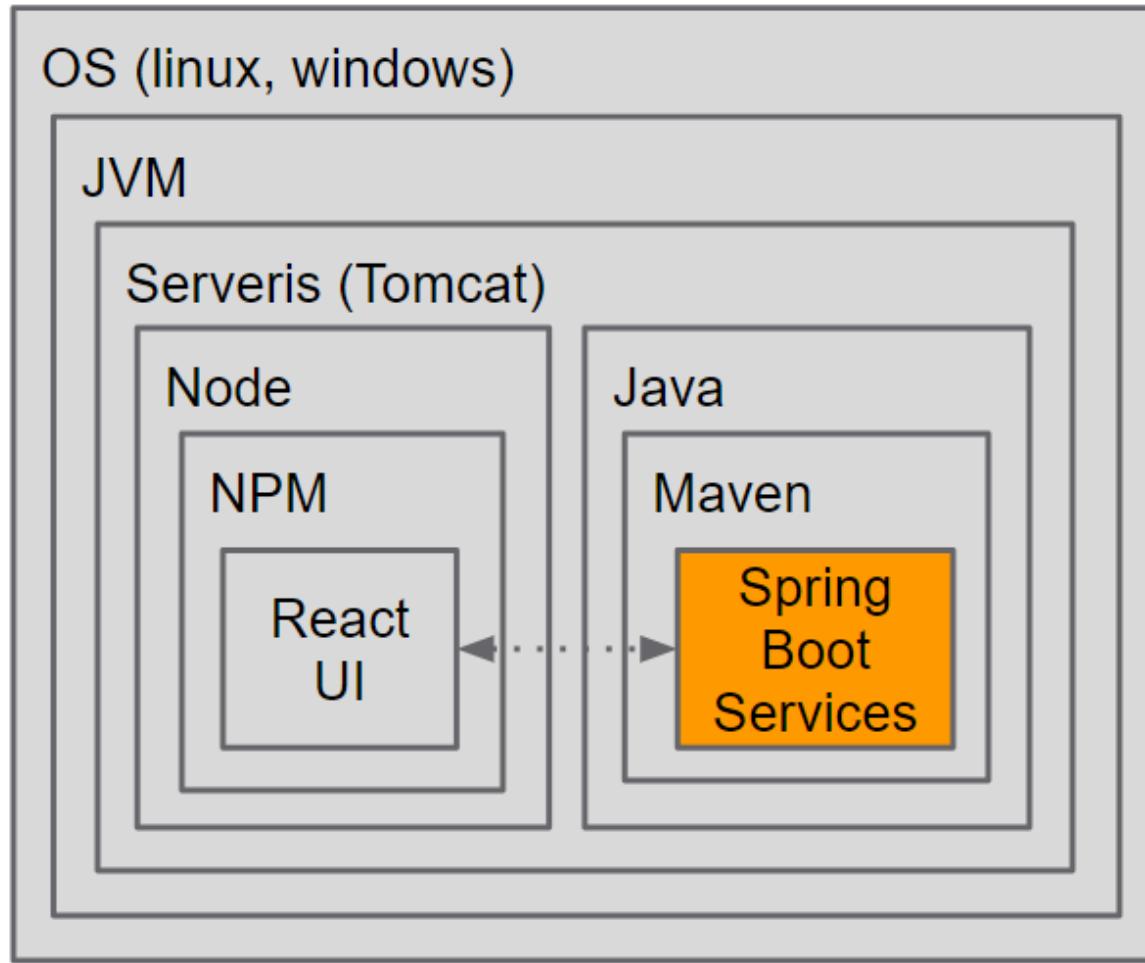
# KAS YRA SPRING. KONFIGŪRACIJA PER XML. MAVEN MIGRACIJA

Andrius Stašauskas

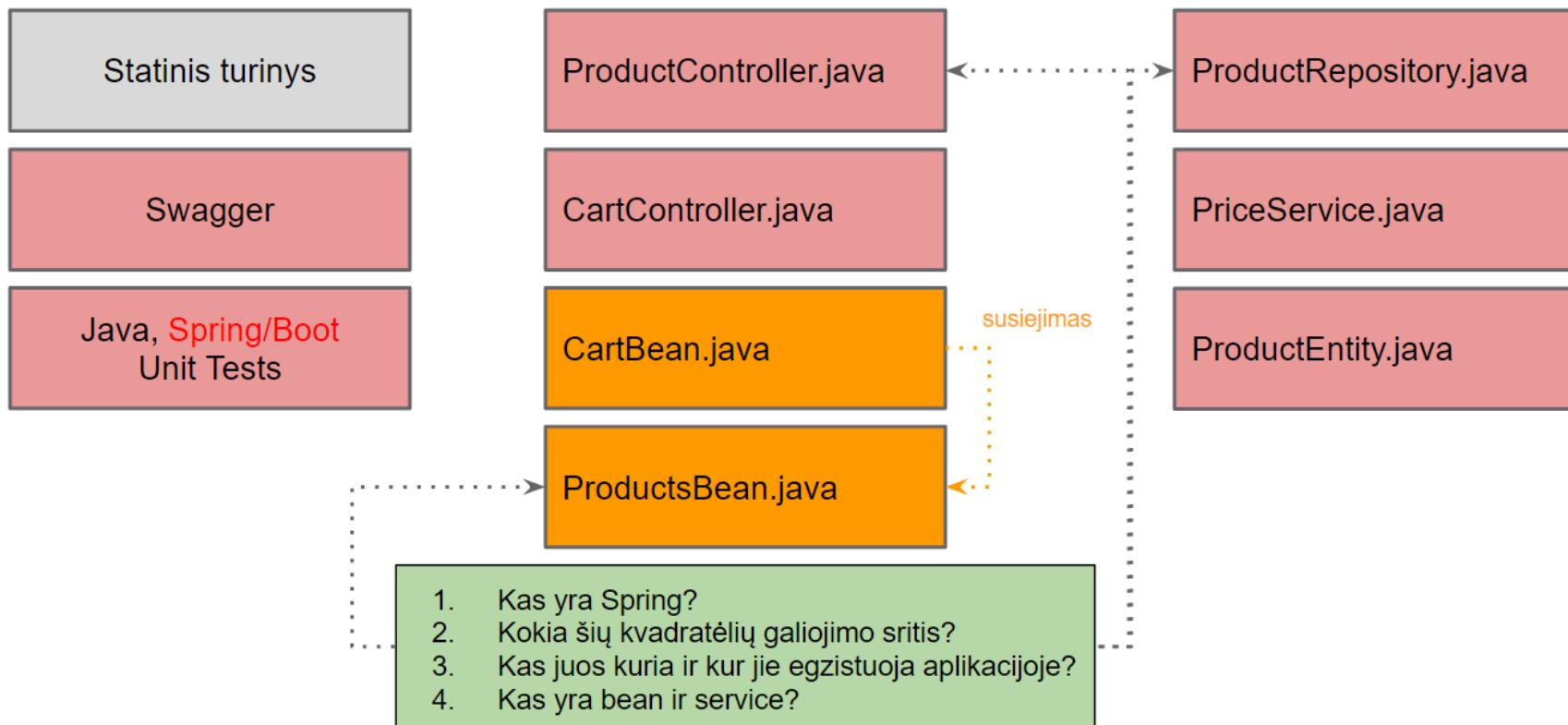
andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# KĄ JAU MOKAME IR KO DAR NE



# KĄ JAU MOKAME IR KO DAR NE



# TURINYS

- Kas yra Spring
  - IoC
  - DI
  - Container ir bean
- Spring konfigūracija:
  - XML
  - priklausomybės
- Maven projekto migracija
  - Spring Boot migracija



## KAS YRA SPRING?

- Vienas populiausiu atviro kodo programinės įrangos kūrimo karkasų Java platformai
- Spring yra lengvasvoris ir pagrindiniai karkaso komponentai užima apie 2MB
- Pagrindinės Spring karkaso savybės gali būti naudojamos kuriant bet kokią Javą aplikaciją, tačiau papildomi karkaso moduliai ir išplėtimai leidžia kurti žiniatinklio aplikacijas naudojant Java EE platformą



## SPRING TIKSLAI

- Supaprastinti Java EE PĮ kūrimo procesą.
- Spręsti problemas, kurių nepadengė Java EE.
- Integracija su populiariausiomis technologijomis.
- Pateikti modulinę architektūrą
  - galima pasirinkti ką naudoti, o ko ne



# MAVEN SPRING PRIKLAUSOMYBIŲ PAVYZDŽIAI

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.3.13.RELEASE</version>
</dependency>
```

- Stabilios Spring Context versijos prilausomybė

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>1.5.9.RELEASE</version>
</dependency>
```

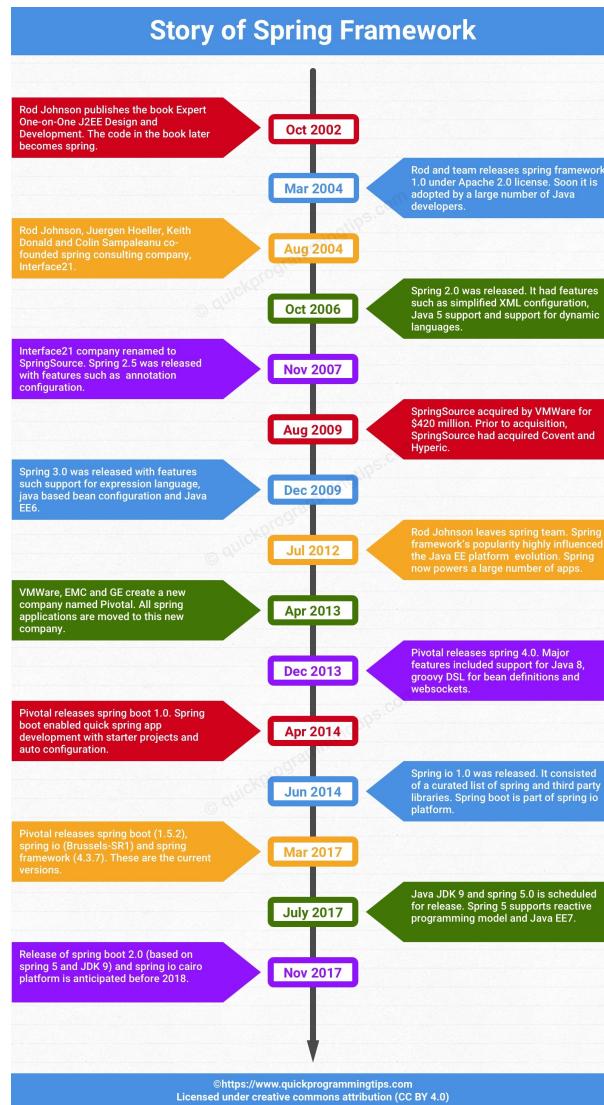
- Spring Boot



# SPRING RAIDA

- Spring 1.0 (2001) - priklausomybių injekcija (DI - dependency injection), AOP, žiniatinklio karkasas.
- Spring 2.0 (2006) - Išplečiama konfigūracija (angl. extensible config), bean galiojimo sritis, dinaminių kalbų palaikymas (pvz. Groovy, Ruby), nauja žymių (angl. tag) biblioteka.
- Spring 2.5 (2007) - valdymas anotacijomis (angl. annotation-driven), automatinis bean suradimas (angl. discovery), naujas žiniatinklio karkasas, JUnit 4 integracija.
- Spring 3.0 (2009) - REST, SpEL, deklaratyvi validacija, ETag palaikymas, konfigūracija Java pagrindu (angl. Java-based).
- Spring 4.0 (2013) - Java 8, groovy DSL for beans and websockets
- Spring Boot 1.0 (2014) - quick Spring app development
  - Spring Boot 1.5.2 (2017)
  - Spring 4.3.7 (2017) - Java 6-8
- Spring 5.0 (2017-2018) - Java 8-10, reactive/functional programming, Java EE7
  - Spring Boot 2.0.6 (2018-10-16) Min Tomcat is 8.5.x
  - Spring 5.0.10 (2018-10-15)
- Spring 5.1 (2018-09-21) - Java 8-12 - Java 18.3 (10; no support), 18.9 (11; long support)
  - Spring Boot 2.1 (2018-10-30)





## PAPILDOMI SPRING PROJEKTAI

- Spring Web Flow
- Spring-WS
- Spring Boot
- Spring Web
- Spring Security
- Spring Batch, Spring Integration, Spring LDAP, Spring IDE / STS
- Spring Rich Client, Spring BeanDoc, BlazeDS Integration, Spring-DM, dmServer, Bundlor, tcServer



# SPRING ARCHITEKTŪRA



Spring Framework Runtime

## Data Access/Integration

JDBC

ORM

OXM

JMS

Transactions

## Web

WebSocket

Servlet

Web

Portlet

AOP

Aspects

Instrumentation

Messaging

## Core Container

Beans

Core

Context

SpEL

Test



AKADEMIJA.LT  
INFOBALT IR TECH CITY

## ŽINIATINKLIS

- **Web** pagrindinis žiniatinklio funkcionalumas. Bylų nusiuntimas ir IoC konteinerio inicializavimas naudojant servlet listener ir web aplikacijų kontekstas.
- **Web-Servlet** modulis suteikia Spring MVC (modelis, vaizdas, kontroleris) implementaciją, skirtą žiniatinklio aplikacijoms.
- **WebSocket** modulis realizuoja integraciją su WebSocket ir SockJS, taip pat STOMP
- **Web-Portlet** modulis suteikia MVC implementaciją, skirtą portalo komponentams (angl. portlet).



## DUOMENŲ PRIEIGA / INTEGRACIJA

- **JDBC** (Java Database Connectivity) šablonų abstrakcijos sluoksnis, skirtą pakeisti tiesioginį JDBC naudojimą.
- **ORM** (Object Relational Mapping) integracija su populiariausiais objektų į realiacijnę DB susiejimo karkasais (pvz. JPA, Hibernate, JDO, iBatis).
- **OXM** Object/XML susiejimo abstrakcijos sluoksnis, skirtis JAXB/Castor/XMLBeans/JiBX/XStream palaikymui.
- **JMS** (Java Messaging Service) - siųsti ir gauti žinutes.
- Tranzakcijų modulis realizuoja deklaratyvų ir programinį tranzakcijų valdymą.



## TESTAVIMAS

- Testavimo modulis turi integraciją su testavimo karkasais:
  - JUnit,
  - TestNG.
- Suteikia galimybę užkrauti testavimui skirtą aplikacijos kontekstą (ApplicationContext).
- Turi testavimui naudingus netikrus (angl. mock) objektus.



## KITI MODULIAI

- **AOP** modulis suteikia aspektais orientuoto programavimo implementaciją, skirtą apibrėžti metodų perėmėjus (angl. method-interceptors), įterpimo taškus (angl. pointcuts).
- **Aspects** modulis realizuoja integraciją su AspectJ aspektais paremtu programavimo karkasu.
- **Instrumentation** modulis suteikia klasių instrumentavimo ir klasių užkrovimo palaikymą specifiniams aplikacijų serveriams.
- **Messaging** modulis realizuoja abstrakcijas žinučių pagrindu veikiančioms aplikacijoms.



## PAGRINDINIS KONTEINERIS

- **Core** modulis realizuoja kertinį Spring karkaso funkcionalumą:
  - kontrolės inversija (IoC - Inversion of Control),
  - priklausomybių injekcija (DI - Dependency Injection).
- **Bean** modulis suteikia BeanFactory fabriko šablono (angl Factory Pattern) implementaciją, skirtą Java objektų sukūrimui.



## PAGRINDINIS KONTEINERIS

- **Context** modulis, naudodamas Core ir Bean modulius, realizuoja objektų aprašymo ir konfigūravimo funkcionalumą. **ApplicationContext** sasaja yra centrinis Context modulio elementas.
- **SpEL** Spring Expression Language (išraiškų kalbų) modulis suteikia galingą užklausų ir objektų grafo manipuliacimo, vykdymo metu, funkcionalumą.



## KAS YRA IOC?

- Įprastoje programoje objektų gyvavimo ciklą kontroliuoja parašytas programinis kodas.
- IoC paremtoje sistemoje objektų gyvavimo ciklą valdo programinis konteineris.
- Jums reikia sukurti tik patį pirminį objektą BeanFactory, o visus kitus objektus, pagal poreikį, sukurs konteineris.



## KAIP REALIZUOJAMAS IOC?

- Konteineris, valdydamas objektų gyvavimo ciklą, taip pat turi valdyti ir ryšius bei priklausomybes tarp objektų. Šiam tikslui naudojamos dvi strategijos:
  - Priklasomybės paieška (angl. Dependency lookup) - komponentas kitus jam reikalingus komponentus susiranda pats.
  - Priklasomybės injekcija (angl. Dependency injection) - konteineris perduoda reikalingus komponentus per:
    - konstruktorių,
    - set\* metodus (JavaBeans properties).
- Spring IoC naudoja priklasomybės injekcijos strategiją.



## INJEKCIJA PER KONSTRUKORIŪ AR SET\* METODUS?

- Injekcija per konstruktorių paprastai naudojama komponento parametrams, kurie yra būtini jo darbui.
- Injekcija per set\* metodus paprastai naudojama, kai komponentas turi parametru reikšmes pagal nutylėjimą arba norima leisti reikšmes perrašyti konteineriui.
- Praktikoje dažniausiai naudojama injekcija per set\* metodus.



## SPRING IOC KONTEINERIAI

- BeanFactory konteineris - paprasčiausias konteineris, realizuojantis bazinej priklausomybių injekcijos palaikymą ir apibrežiamas org.springframework.beans.factory.BeanFactory interfeisu. BeanFactory ir kiti susiję interfeisai (pvz. BeanFactoryAware, InitializingBean, DisposableBean) yra vis dar laikomi Spring karkase dėl atgalinio suderinanumo su daugeliu trečių šalių karkasų.



# SPRING IOC KONTEINERIAI

- ApplicationContext konteineris - dažniausiai naudojamas konteineris, apibrėžiamas org.springframework.context.ApplicationContext interfeisu. Pagrindinės savybės:
  - I18N palaikymas.
  - Įvykių skleidimas (angl. Event Propagation): ContextRefreshedEvent, ContextStartedEvent, ContextStoppedEvent, ContextClosedEvent, RequestHandledEvent.
  - Resursų užkrovimas.



## APPLICATIONCONTEXT IMPLEMENTACIJOS

- `FileSystemXmlApplicationContext` - šis konteineris nuskaito bean aprašymo XML bylas naudodamas į konstruktorių perduotą pilną bylos kelią.
- `ClassPathXmlApplicationContext` - šis konteineris nuskaito bean aprašymo XML bylas iš aplikacijos CLASSPATH.
- `WebXmlApplicationContext` - šis konteineris nuskaito bean aprašymo XML bylas iš žiniatinklio aplikacijos.



# SPING IOC PAVYZDYS - BEAN

```
package com.tutorialspoint;
public class HelloWorld {
    private String message;
    public void setMessage(String message) {
        this.message = message;
    }
    public void getMessage() {
        System.out.println("Your Message : " + message);
    }
}
```



# SPING IOC PAVYZDYS - BEANS.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
        <property name="message" value="Hello World!"/>
    </bean>
</beans>
```



# SPING IOC PAVYZDYS - BEANFACTORY

```
package com.tutorialspoint;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
public class MainApp {
    public static void main(String[] args) {
        XmlBeanFactory factory = new XmlBeanFactory
            (new ClassPathResource("Beans.xml"));
        HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
        obj.getMessage();
    }
}
```



# SPRING IOC PAVYZDYS - APPLICATIONCONTEXT

- vieno iš galimų kontekstų FileSystemXmlApplicationContext pavyzdys

```
package com.tutorialspoint;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new FileSystemXmlApplicationContext
            ("C:/Users/ZARA/workspace/HelloSpring/src/Beans.xml");
        HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
        obj.getMessage();
    }
}
```

# UŽDUOTIS 1 - SUKURTI PROJEKTO STRUKTŪRĄ

- Pagrindiniame projektų kataloge sukurti naują **FirstSpringProject** panaudojant Maven archetipą:

```
$ mvn archetype:generate -DgroupId=lt.itmokymai.spring \
> -DartifactId=FirstSpringProject \
> -DarchetypeArtifactId=maven-archetype-quickstart \
> -DinteractiveMode=false
```

```
# Pasirinkti projekto katalogą:
$ cd FirstSpringProject/
# Sukurti src/main/resources katalogą:
$ mkdir src/main/resources
```

- Sukurti Eclipse projekto konfigūraciją:

```
$ mvn eclipse:eclipse
```



# UŽDUOTIS 1 - POM.XML PRIDĒTI SPRING PRIKLAUSOMYBĘ

- Taigi Spring Boot archetipą kol kas palikome nuošalyje ir pasidarėme naują quickstart paremtą **FirstSpringProject**
- Pridékime į jo pom.xml Spring

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.10.RELEASE</version>
</dependency>
```



# UŽDUOTIS 1 - POM.XML PRIDĒTI EXEC:JAVA PAPILDINĮ

```
<build><plugins><plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.3.2</version>
  <executions>
    <execution>
      <goals>
        <goal>java</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <mainClass>lt.itmokymai.spring.App</mainClass>
  </configuration>
</plugin></plugins></build>
```



# UŽDUOTIS 1 - PRIDĒTI APLIKACIJŲ KONTEKSTO BYLĄ

- Sukurti pradinę src/main/resources/ **application-context.xml** bylą:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Bean definitions goes there -->

</beans>
```



# UŽDUOTIS 1 - SUKURTI BEAN SERVICEA

- Sukurti ServiceA klasę lt.itmokymai.spring pakete:

```
package lt.itmokymai.spring;
public class ServiceA {
    private String message;
    public String getResult() { return getMessage(); }
    public String getMessage() { return message; }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

- Užregistruoti ServiceA bean application-context.xml byloje:

```
<bean id="serviceABean" class="lt.itmokymai.spring.ServiceA">
    <property name="message" value="ServiceA message" />
</bean>
```



# UŽDUOTIS 1 - PAKEISTI LT.ITMOKYMAI.SPRING.APP KLASĘ

- App.main() metode sukurti IoC konteinerį.
- iš konteinerio gauti ServiceA bean.
- atspausdinti ServiceA.getResult() rezultatą

```
package lt.itmokymai.spring;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class App {
    public static void main( String[] args ) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
            "application-context.xml");
        ServiceA serviceA = (ServiceA) context.getBean("serviceABean");
        System.out.println(serviceA.getResult());
        ((ConfigurableApplicationContext) context).close();
    }
}
```



# UŽDUOTIS 1 - ĮVYKDYTI LT.ITMOKYMAI.SPRING.APP KLASĘ

- Pagaminti projekto darinį:

```
$ mvn clean package
```

- Įvykdyti lt.itmokymai.spring.App klasę naudojant exec:java papildinį:

```
$ mvn exec:java
```

- Įvykdyti lt.itmokymai.spring.App klasę Eclipse priemonėmis.



# SPRING BEAN (PUPOS)

## KAS YRA SPRING BEAN?

- Objektai, kurie sudaro aplikacijos pagrindą ir kurių gyvavimo ciklas (inicIALIZAVIMAS, surinkimas ir pan.) yra valdomas Spring IoC konteinerio.
- Paprastai tai Java klasė, realizuojanti tam tikrą interfeisą ir JavaBean specifikaciją.
- Bean sukūrimui konteineris naudoja konfigūracijos metaduomenis:
  - XML paremta konfigūracija,
  - anotacijomis paremta konfigūracija,
  - Java paremta konfigūracija.



## SPRING BEAN APRAŠAS

- `class` - privalomas atributas nurodantis bean sukūrimui naudojama Java klasę.
- `name` - šis atributas nurodo unikalų bean identifikatorių. XML konfigūracijoje galima naudoti `id` ir / arba `name` atributus bean identifikatorių nurodymui.
- `scope` - šis atributas nurodo bean galiojimo sritį.
- `constructor-arg` - naudojamas priklausomybių injekcijai į bean konstruktorių.
- `properties` - naudojamas priklausomybių injekcijai į `set*` metodus.



## SPRING BEAN APRAŠAS

- `autowire` - naudojamas automatinei priklausomybių injekcijai.
- `lazy-init` - nurodo konteineriui sukurti bean pagal poreikį, o ne konteinerio paleidimo metu.
- `init-method` - grįžtamasis iškvietimas po to kai konteineris priskyrė visas privalomas bean savybes.
- `destroy-method` - grįžtamasis iškvietimas po to, kai bean valdantis konteineris yra sunaikimamas.



# SPRING BEAN APRAŠYMO PAVYZDYS 1

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <!-- A simple bean definition -->
    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- A bean definition with lazy init set on -->
    <bean id="..." class="..." lazy-init="true">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
</beans>
```



# SPRING BEAN APRAŠYMO PAVYZDYS 2

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <!-- A bean definition with initialization method -->
    <bean id="..." class="..." init-method="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- A bean definition with destruction method -->
    <bean id="..." class="..." destroy-method="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- more bean definitions go here -->
</beans>
```



# SPRING BEAN GALIOJIMO SRITIS (SCOPE)

- singleton - konteineris sukurs tik vieną bean esybę (naudojama pagal nutylėjimą).
- prototype - IoC kiekvieną kartą kurs naują bean esybę.
- request - bean galiojimo sritis yra HTTP užklausa. Galimas tik su žiniatinklio ApplicationContext.
- session - bean galiojimo sritis yra HTTP sesija. Galimas tik su žiniatinklio ApplicationContext.
- application - bean galiojimo sritis yra aplikacija. Galimas tik su žiniatinklio ApplicationContext.
- websocket - bean galiojimo sritis yra websocket. Galimas tik su žiniatinklio ApplicationContext.



# SPRING BEAN GALIOJIMO SRITIS

- Pavyzdys:

```
<!-- A bean definition with singleton scope -->
<bean id="..." class="..." scope="singleton">
    <!-- collaborators and configuration for this bean go here -->
</bean>
```



# SPRING BEAN GALIOJIMO SRITIS

- Norint panaduoti mažesnės galiojimo srities bean didesnėje, reikia naudoti aop:proxy:

```
<!-- HTTP sesijos bean -->
<bean id="userPreferences"
      class="com.something.UserPreferences" scope="session">
    <!-- nurodo IoC is userPreferences padaryti proxy bean -->
    <aop:scoped-proxy/>
</bean>

<!-- singleton bean kuris gauna proxy bean [userPreferences] -->
<bean id="userService" class="com.something.SimpleUserService">
    <property name="userPreferences" ref="userPreferences"/>
</bean>
```



# SPRING BEAN GYVAVIMO CIKLAS - INICIALIZAVIMAS

- Spring bean inicializavimo grįztamasis iškvietimas naudojant InitializingBean interfeisą:

```
import org.springframework.beans.factory.InitializingBean;
public class MyBean implements InitializingBean {
    public void afterPropertiesSet() { /* do some initialization work */
}
```

- arba init-method attributą XML Spring bean konfigūracijoje:

```
<bean id="myBean" class="pvz.MyBean" init-method="init"/>
public class MyBean {
    public void init() { /* do some initialization work */ }
}
```



# SPRING BEAN GYVAVIMO CIKLAS - SUNAIKINIMAS

- Spring bean sunaikinimo grįztamasis iškvietimas naudojant **DisposableBean** interfeisą:

```
import org.springframework.beans.factory.DisposableBean;
public class MyBean implements DisposableBean {
    public void destroy() { /* do some destruction work */ }
}
```

- arba **destroy-method** attributą XML Spring bean konfigūracijoje:

```
<bean id="myBean" class="pvz.MyBean" destroy-method="destroy"/>
public class MyBean {
    public void destroy() { /* do some destruction work */ }
}
```



## SPRING BEAN GYVAVIMO CIKLAS - DEFAULT

- Spring bean pagal nutylėjimą inicializavimo ir sunaikinimo grįztamieji iškvietimai gali būti nurodomi XML konfigūracijos default-init-method and default-destroy-method atributais:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
       default-init-method="init" default-destroy-method="destroy">
    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
</beans>
```



## SPRING BEAN BAIGIAMOJI DOROKLĖ

- Spring bean baigiamosios doroklės (angl. post processor) grj̄tamasis iškvietimas yra nurodomas BeanPostProcessor interfeisu.
- BeanPostProcessor intefeisas deklaruojant postProcessBeforeInitialization or postProcessAfterInitialization metodus, skirtus bean inicializavimo papildymui.
- Galima užregistruoti daugiau nei vieną BeanPostProcessor.



## SPRING BEAN BAIGIAMOJI DOROKLĖ

- BeanPostProcessor implementuoja the Ordered interfeisą, tam kad galima būtų pakeisti baigiamųjų doroklių iškvietimo seką.
- ApplicationContext automatiškai randa visus bean kurie implementuoja BeanPostProcessor intefesis ir užregistruoja konteineryje.
- Konteineris sukurtą bean objektą perduoda BeanPostProcessortolimesniam inicializavimui.



# BAIGIAMOJI DOROKLĖ PAVYZDYS - BEAN

```
package lt.itmokymai.spring;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.beans.BeansException;
public class InitHelloWorld implements BeanPostProcessor {
    public Object postProcessBeforeInitialization(Object bean,
        String beanName) throws BeansException {
        System.out.println("BeforeInitialization : " + beanName);
        return bean; // you can return any other object as well
    }
    public Object postProcessAfterInitialization(Object bean,
        String beanName) throws BeansException {
        System.out.println("AfterInitialization : " + beanName);
        return bean; // you can return any other object as well
    }
}
```



# BAIGIAMOJI DOROKLĖ PAVYZDYS - XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="helloWorld" class="com.tutorialspoint.HelloWorld"
          init-method="init" destroy-method="destroy">
        <property name="message" value="Hello World!"/>
    </bean>
    <bean class="com.tutorialspoint.InitHelloWorld" />
</beans>
```



## UŽDUOTIS 2

- prieš bean kuriantis, išvesti bean vardą
- bean susikūrus, pranešti apie tai, kad bean sukurtas
  - pranešime turi figūruoti bean varda
- po bean sunaikinimo išvesti tekstą, kad bean susinaikino
  - pranešime turi figūruoti bean varda



# SPRING PRIKLAUSOMYBĖS



AKADEMIJA.LT  
INFOBALT IR TECH CITY

# SPRING BEAN APRAŠYMO PAVELDĖJIMAS

- Bean gali turėti įvairios konfigūracijos: konstruktorių argumentai, atributai, inicializavimo metodai ir kt.
- Vaiko (angl. child) bean paveldi konfigūracijos duomenis iš tėvo (angl. parent) bean aprašymo ir gali perkrauti reikiamas reikšmes ir / arba pridėti naujas.
- Bean aprašymo paveldėjimas yra nesusijęs su Java klasių paveldėjimu ir bean aprašymą galima naudoti, kaip šabloną kitų bean aprašymui.
- XML paremtoje konfigūracijoje vaiko bean indikuoja parent atributas, nurodantis tėvo bean identifikatorių (bean name arba id atributo reikšmę).



# BEAN APRAŠYMO PAVELDĖJIMO PAVYZDYS

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
        <property name="message1" value="Hello World!"/>
        <property name="message2" value="Hello Second World!"/> <!-- inherits from parent -->
    </bean>
    <bean id="helloIndia" class="com.tutorialspoint.HelloIndia"
          parent="helloWorld">
        <property name="message1" value="Hello India!"/> <!-- override -->
        <property name="message3" value="Namaste India!"/> <!-- add -->
    </bean>
</beans>
```



# SPRING BEAN PRIKLAUSOMYBIŲ INJEKCIJA

- Per konstruktorių - priklausomybių injekcija, kai konteineris iškviečia klasės konstruktorių su argumentais, atstovaujančius kitos klasės priklausomybę.
- Per set\* - konteineris iškvietęs klasės konstruktorių be parametru, po to kviečia set\* metodus, atstovaujančius kitų klasių priklausomybes.
- Vienu metu galima naudoti abu priklausomybių injekcijos būdus. Rekomenduojama privalomoms priklausomybėms naudoti injekciją per konstruktorių, o neprivalomoms injekciją per set\* metodus.



# PRIKLAUSOMYBIŲ INJEKCIJA PER KONSTRUKTORIŪ 1

```
public class SpellChecker {  
    public SpellChecker() {  
        System.out.println("Inside SpellChecker constructor." );  
    }  
    public void checkSpelling() {  
        System.out.println("Inside checkSpelling." );  
    }  
}  
public class TextEditor {  
    private SpellChecker spellChecker;  
    public TextEditor(SpellChecker spellChecker) {  
        System.out.println("Inside TextEditor constructor." );  
        this.spellChecker = spellChecker;  
    }  
    public void spellCheck() { spellChecker.checkSpelling(); }  
}
```



# PRIKLAUSOMYBIŲ INJEKCIJA PER KONSTRUKTORIŪ 2

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <!-- Definition for textEditor bean -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <constructor-arg ref="spellChecker"/>
    </bean>
    <!-- Definition for spellChecker bean -->
    <bean id="spellChecker" class="com.tutorialspoint.SpellChecker"/>
</beans>
```



# PRIKLAUSOMYBIŲ INJEKCIJA PER SET\* 1

```
public class TextEditor {  
    private SpellChecker spellChecker;  
    // a setter method to inject the dependency.  
    public void setSpellChecker(SpellChecker spellChecker) {  
        System.out.println("Inside setSpellChecker." );  
        this.spellChecker = spellChecker;  
    }  
    // a getter method to return spellChecker  
    public SpellChecker getSpellChecker() {  
        return spellChecker;  
    }  
    public void spellCheck() {  
        spellChecker.checkSpelling();  
    }  
}
```



# PRIKLAUSOMYBIŲ INJEKCIJA PER SET\* 2

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <!-- Definition for textEditor bean -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <property name="spellChecker" ref="spellChecker"/>
    </bean>
    <!-- Definition for spellChecker bean -->
    <bean id="spellChecker" class="com.tutorialspoint.SpellChecker"/>
</beans>
```



# VIDINIO SPRING BEAN DEKLARACIJA

- Kaip Java klasėje galima deklaruoti vidinę klasę, taip pat vidiniai bean yra deklaruojami kito bean `<property/>` arba `<constructor-arg/>` deklaracijos apimtyje:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="outerBean" class="...">
        <property name="target">
            <bean id="innerBean" class="..."/>
        </property>
    </bean>
</beans>
```



## UŽDUOTIS 3 - SUKURTI BEAN SERVICEB

- Sukurti ServiceB klasę lt.itmokymai.spring pakete ir realizuoti ServiceA priklausomybę naudojant set\* priklausomybės injekciją:

```
package lt.itmokymai.spring;
public class ServiceB {
    private ServiceA serviceA;
    public void setServiceA(ServiceA serviceA) {
        this.serviceA = serviceA;
    }
    public String getResult () {
        return "ServiceB result:" + serviceA.getResult ();
    }
}
```



## UŽDUOTIS 3 - SUKURTI BEAN SERVICEB

- src/main/resources/application-context.xml užregistruoti ServiceB

```
<bean id="serviceBBean" class="lt.itmokymai.spring.ServiceB">
    <property name="serviceA" ref="serviceABean" />
</bean>
```

- lt.itmokymai.spring.App.main() atspausdinti ServiceB.getResult() metodo rezultatą
- Įvykdyti lt.itmokymai.spring.App klasę.



## UŽDUOTIS 3 - PAKEISTI BEAN SERVICEB

- Pakeisti `lt.itmokymai.spring.ServiceB` klasę, kad naudotų `ServiceA` priklausomybės injekciją per konstruktorių.
- Pakeisti `ServiceB` bean deklaraciją `src/main/resources/application-context.xml` byloje.



## UŽDUOTIS 3 - SUKURTI BEAN SERVICEC

- Sukurti lt.itmokymai.spring.ServiceC klasę, kuri paveldėtų lt.itmokymai.spring.ServiceA klasę.
- Pridėti ServiceC bean deklaraciją  
src/main/resources/application-context.xml  
byloje taip, kad paveldėtų ServiceA bean message atributo priskirtą reikšmę.



## UŽDUOTIS 3 - SUKURTI BEAN SERVICEC

- Perkrauti ServiceC.getResult() metodą:

```
public String getResult() {  
    return "ServiceC result:"+getMessage();  
}
```

- lt.itmokymai.spring.App.main() atspausdinti ServiceC.getResult() metodo rezultatą į komandinę eilutę.



# SPRING BEAN KOLEKCIJOS

# SPRING BEAN KOLEKCIJŲ INJEKCIJA

- Spring bean XML aprašas leidžia deklaruoti kolekcijos tipo atributus:
  - `<list>` - skirtas apibrėžti primitivių tipų arba bean sąrašą. Leidžia pasikartojančias reikšmes.
  - `<set>` - skirtas apibrėžti primitivių tipų arba bean aibę, be pasikartojančių reikšmių.
  - `<map>` - skirtas rako - reikšmės kolekcijai, kai raktas ir reikšmė gali būti bet koks tipas.
  - `<props>` - skirtas rako - reikšmės kolekcijai, kai raktas ir reikšmė yra String tipo.



# KOLEKCIJŲ INJEKCIJOS PAVYZDYS - KLASĖ

```
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
public class Customer
{
    private List<Object> lists;
    private Set<Object> sets;
    private Map<Object, Object> maps;
    private Properties pros;
    //...
}
```



# KOLEKCIJŲ INJEKCIJOS PAVYZDYS - LIST

```
<bean id="..." class="Customer">
    <property name="lists">
        <list>
            <value>1</value>
            <ref bean="PersonBean" />
            <bean class="com.mkyong.common.Person">
                <property name="name" value="mkyongList" />
                <property name="address" value="address" />
                <property name="age" value="28" />
            </bean>
        </list>
    </property>
</bean>
```



# KOLEKCIJŲ INJEKCIJOS PAVYZDYS - SET

```
<bean id="..." class="Customer">
    <property name="sets">
        <set>
            <value>1</value>
            <ref bean="PersonBean" />
            <bean class="com.mkyong.common.Person">
                <property name="name" value="mkyongSet" />
                <property name="address" value="address" />
                <property name="age" value="28" />
            </bean>
        </set>
    </property>
</bean>
```



# KOLEKCIJŲ INJEKCIJOS PAVYZDYS - MAP

```
<bean id="..." class="Customer">
    <property name="maps">
        <map>
            <entry key="Key 1" value="1" />
            <entry key="Key 2" value-ref="PersonBean" />
            <entry key="Key 3">
                <bean class="com.mkyong.common.Person">
                    <property name="name" value="mkyongMap" />
                    <property name="address" value="address" />
                    <property name="age" value="28" />
                </bean>
            </entry>
        </map>
    </property>
</bean>
```



# KOLEKCIJŲ INJEKCIJOS PAVYZDYS - PROPS

```
<bean id="..." class="Customer">
    <property name="props">
        <props>
            <prop key="admin">admin@nospam.com</prop>
            <prop key="support">support@nospam.com</prop>
        </props>
    </property>
</bean>
```



## UŽDUOTIS 4

- Aprašyti spring bean kolekciją, kurioje būtų produktų sąrašas (kaip React)
- Kiekvienas produktas taip pat savaime turi būti bean
- Produktas (produkto klasė) turi turėti title, image ir kitus parametrus (kaip React)
- ServiceC klasėje sukurti produktų sąrašą
- Nuskaityti tokią kolekciją iš App main klasės ir išvesti į ekraną (konsolę) produktų pavadinimus



# SPRING BEAN AUTOMATINIS SURIŠIMAS

Autowire



## SPRING BEAN AUTOMATINIS SURIŠIMAS

- Spring konteineris gali automatiškai surišti (angl. autowire) bendradarbiaujančius bean, nenaudodamas tiesiogiai <constructor-arg> ir / arba <property> elementuose nurodytos priklausomybių injekcijos.
- Automatinio surišimo būdai:
  - no - Naudojamas pagal nutylėjimą. Nurodo kad automatinis surišimas nebus atliekamas ir surišimas turi būti deklaruotas tiesiogiai nurodant priklausomybių injekcijas.



## SPRING BEAN AUTOMATINIS SURIŠIMAS

- `byName` - surišimas pagal savybės vardą. Spring konteineris bean attributų vardus, kaip identifikatorius, naudos kitų užregistruotų bean paieškai ir priklausomybės injekcijai.
- `byType` - surišimas pagal bean klasės tipą. Spring konteineris bean attributų tipus naudos kitų to paties tipo užregistruotų bean paieškai ir priklausomybės injekcijai. Konteineris meta klaidą jei randamas daugiau nei vienas atributo tipo bean.



## SPRING BEAN AUTOMATINIS SURIŠIMAS

- constructor - panašus į byType, tačiau naudojamas tik konstruktoriaus parametrams. Konteineris meta klaidą jei randamas daugiau nei vienas parametro tipo bean
- pavyzdžiui, žymime bean, kad Spring pats suieškotų priklausomybes pagal tipą:

```
<bean id="..." class="..." autowire="byType">
```

- tačiau norint naudoti kelis režimus, reikia naudoti anotacijas



## UŽDUOTIS 5 - PRIKLAUSOMYBĖS

- ServiceB naudoja ServiceA. Todėl pašalinkite xml konfigūracijoje nuorodas į ServiceA
  - tiek constructor-arg parametrus, tiek property
- Nurodykite xml konfigūracijoje ServiceB parametrumą autowire
  - pabandykite parinkti skirtinges reikšmes
- atkreipkite dėmesį kad constructor ir byType gali ieškoti tipo, o to tipo bean gali egzistuoti keli, ir jis nežinos, kuri pasirinkti



# SPRING BOOT PRIKLAUSOMYBIŲ MIGRACIJA

## UŽDUOTIS 6 - PRIKLAUSOMYBIŲ MIGRACIJA

- Trumpam grįžkime prie savo Spring Boot Starter archetipu paremtu projekto
- Norėtumėme naudoti ne 4.1.8, o 5.0.10 Spring versiją
- Spring Boot kartu atsineša (transitive) ir Spring [Core], todėl reikia migruoti Spring Boot į naujesnę versiją
- migruoti lengviausia po truputį per minor versijas, o ne iškart į naujausią
- pom.xml pakeisti versiją:

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
</parent>
```



## UŽDUOTIS 6 - PRIKLAUSOMYBIŲ MIGRACIJA

- Kadangi pasikeitė Spring Boot API, tai ir DataSource Spy nebeveikia
- Iš tikrujų, reikia ne tik pom.xml pakeisti versiją, bet ir pridėti dependency:

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
</parent>
...
<dependency>
    <groupId>com.integralblue</groupId>
    <artifactId>log4jdbc-spring-boot-starter</artifactId>
    <version>1.0.2</version>
</dependency>
```



# UŽDUOTIS 6 - PRIKLAUSOMYBIŲ MIGRACIJA

- application.properties pridėti:

```
logging.level.jdbc.sqlonly=DEBUG
```

- AppConfig.java ištrinti visą klasės turinį

```
@Configuration  
public class AppConfig { }
```

- Pakeitus pom.xml, reikia pergeneruot mvn eclipse:eclipse ir Eclipse projektą atnaujinti Refresh
- Eclipse galima atsidaryti pom.xml ir Dependencies Hierarchy turi būti nauja spring-core versija
- Paleisti aplikaciją

## UŽDUOTIS 6 - PAKETŲ MIGRACIJA

- App.java reikia migruoti SpringBootServletInitializer, nes jis buvo perkeltas į naują package ir yra pažymėtas kaip @Deprecated
- Taip žymima minor versijoje tai, kas bus pašalinta pasikeitus major versijai. Taigi šj import

```
import org.springframework.boot.context.web.SpringBootServletInitializer;
```

- migruojame (pakeičiame) į

```
import org.springframework.boot.web.support.SpringBootServletInitializer;
```

- Paleisti aplikaciją ir pasitikrinti, kad ji veikia tiek tomcat, tiek spring-boot



## UŽDUOTIS 7 - MIGRACIJA 1.4.2 -> 1.4.7

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.7.RELEASE</version>
</parent>
```

- mvn clean install, mvn eclipse:eclipse
- Eclipse atsidaryti pom.xml ir Dependencies Hierarchy turi būti 4.3.9 spring-core versija
- Paleisti aplikaciją ir pasitikrinti, kad ji veikia tiek tomcat, tiek spring-boot
  - t.y. /calc servisas turi veikti



# UŽDUOTIS 7 - MIGRACIJA 1.4.7 -> 1.5.0

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.0.RELEASE</version>
</parent>
```

- pasitikrinti, kad **spring-core** versija 4.3.6
- HelloControllerTest.java ištrinti (-) / pridėti (+) eilutes

```
-import org.springframework.boot.test.IntegrationTest;
-import org.springframework.boot.test.SpringApplicationConfiguration;
-import org.springframework.test.context.web.WebAppConfiguration;
+import org.springframework.boot.test.context.SpringBootTest;
-@SpringApplicationConfiguration(classes = App.class)
-@WebAppConfiguration
-@IntegrationTest({"server.port:0",
-    "spring.datasource.url:jdbc:h2:mem:hello-world-calc;DB_CLOSE_ON_EXIT=FALSE"})
+@SpringBootTest(classes = App.class, webEnvironment = SpringApplication.WebEnvironment.RANDOM_PORT)
```



## UŽDUOTIS 7 - MIGRACIJA 1.5.0 -> 1.5.17

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.17.RELEASE</version>
</parent>
```

- pasitikrinti, kad **spring-core** versija 4.3.20



# UŽDUOTIS 7 - MIGRACIJA 1.5.17 -> 2.0.0

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.RELEASE</version>
</parent>
...
<!-- pasalinti spring loaded -->
-<dependencies>
-  <dependency>
-    <groupId>org.springframework</groupId>
-    <artifactId>springloaded</artifactId>
-    <version>${spring-loaded.version}</version>
-  </dependency>
-</dependencies>
```

- pasitikrinti, kad spring-core versija 5.0.4



# UŽDUOTIS 7 - MIGRACIJA 1.5.17 -> 2.0.0

- App.java pakeisti servlet import

```
-import org.springframework.boot.web.support.SpringBootServletInitializer;  
+import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;
```

- nuo šios versijos keičiasi ir aplikacijos paleidimo komandos, nes serveris keičiasi iš tomcat7 į tomcat8
- paleisti spring-boot

```
mvn clean install spring-boot:run -Dspring-boot.run.arguments==server.port=8081
```

- paleisti įdėtinį tomcat8

```
$ mvn clean install org.codehaus.cargo:cargo-maven2-plugin:1.7.0:run \  
> -Dcargo.maven.containerId=tomcat8x -Dcargo.servlet.port=8081 \  
> -Dcargo.maven.containerUrl=http://repo1.maven.org/maven2/org/apache/tomcat/tomcat/8.5.35/tomcat-8.5.35.zip
```



## UŽDUOTIS 7 - MIGRACIJA 2.0.0 -> 2.0.6

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.6.RELEASE</version>
</parent>
```

- pasitikrinti, kad **spring-core** versija 5.0.10



# KITOJE PASKAITOJE

Spring konfigūracija anotacijomis. Spring Junit testai. Spring Boot. Rest Servisi. Swagger. Spring Boot Junit testai



# IŠ PRAEITOS PASKAITOS

- eliminuokite tai, kur daugiausiai sugaištate laiko:
  - stabdo IDE - keiskite į kitą
  - stabdo maven projektai - naudokite java projektus (eclipse:eclipse)
  - stabdo virtuali mašina - persidarykite iš naujo arba konfigūruokite windows
  - ilgai kraunasi virtuali mašina - saugokite jos state vietoj power off
  - ilgai ieškote kur kas yra - pasidarykite vieną katalogą ir ten laikykite visus projektus
  - projektų pavadinimai nesuprantami - pasikeiskite
  - katalogų pavadinimai nesuprantami - pasikeiskite
  - ilgai trunka react-create-app arba maven archetipo kūrimas - kopijuokite esamą projektą į naują katalogą, persivadinkite, bet negaminkite visko iš naujo
  - nematote elementarių klaidų - **reikia pailsėti**





# AKADEMIJA.IT

INFOBALT IR TECH CITY

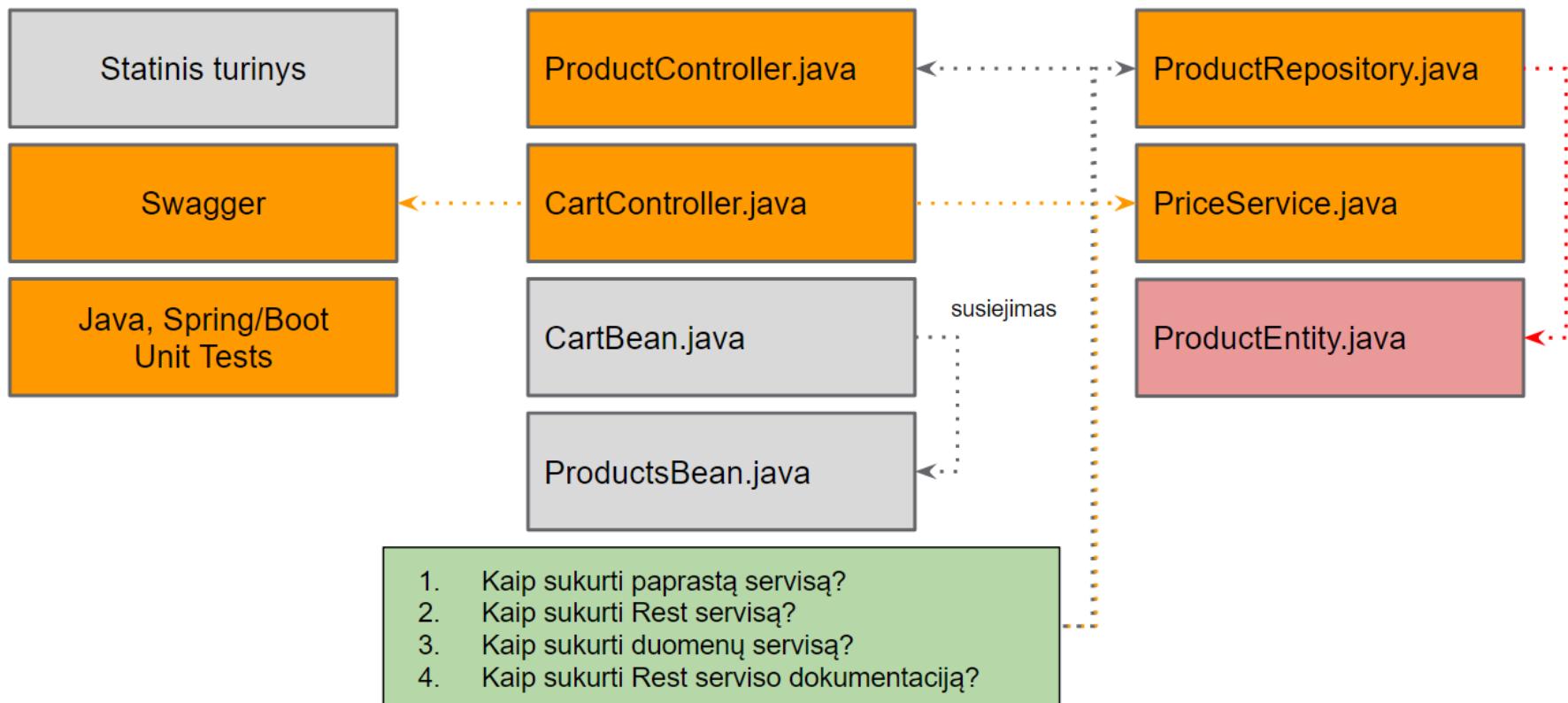
## SPRING ANOTACIJOS. JUNIT. SPRING BOOT. REST SERVISAI. SWAGGER

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# KĄ JAU MOKAME IR KO DAR NE



# TURINYS

- Spring konfigūracija:
  - anotacijos priklausomybėms
  - anotacijos bean kūrimui
- Spring Junit testuose
- Spring Boot
- Rest Servisai
- Swagger
- Integraciniai testai
  - Spring Boot Junit testai



# **ANOTACIJOMIS PAREMTA KONFIGŪRACIJA**



**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

## **ANOTACIJOMIS PAREMTA KONFIGŪRACIJA**

- Pradedant nuo Spring 2.5 priklausomybių injekcijas galima konfigūruoti anotacijomis.
- Vietoj XML konfigūracijos yra anotuojama bean klasė ir / arba jos atributai ir metodai.
- Anotacijų injekcija yra atliekama prieš XML injekciją, o tai suteikia galimybę perkrauti konfigūraciją.



# BEAN SURIŠIMAS ANOTACIJOMIS

- Anotacijomis paremtas surišimas nėra įjungtas pagal nutylėjimą, jį reikia aktyvuoti XML konfigūracijos byloje:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <context:annotation-config/>
    <!-- bean definitions go here -->
</beans>
```

- Tai tik surišimas, bet ne Bean kūrimas

## SUSIEJIMO ANOTACIJOS

- @Required anotacija taikoma bean set\* metodams.
- @Autowired anotacija gali būti taikoma atributamas, set\* metodams ir konstruktoriui.
- @Qualifier anotacija kartu su @Autowired patikslina su kuriuo konkrečiai bean susieti priklausomybę.



## SUSIEJIMO ANOTACIJOS

- Spring palaiko sekančias JSR-250 anotacijas:
  - `@Resource(name="beanName")` - alternatyva `@Autowired` ir `@Qualifier("beanName")` anotacijų kombinacija.
  - `@PostConstruct` - alternatyva bean aprašymo `init-method` XML attributui.
  - `@PreDestroy` - alternatyva bean aprašymo `destroy-method` XML attributui.



# SUSIEJIMO ANOTACIJŲ PAVYZDYS

```
public class Student {  
    private Integer age;  
    private String name;  
    @Required  
    public void setAge(Integer age) { this.age = age; }  
    public Integer getAge() { return age; }  
    @Required  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getName() {  
        return name;  
    }  
}
```



# SUSIEJIMO ANOTACIJŲ PAVYZDYS

```
public class Profile {  
    @Autowired  
    @Qualifier("student1")  
    private Student student;  
    private Student student2;  
    public Profile(){  
        System.out.println("Inside Profile constructor.");  
    }  
    public void printAge() {  
        System.out.println("Age : " + student.getAge());  
    }  
    public Profile(@Qualifier("studentas2") Student student2) {  
        this.student2 = student2;  
    }  
}
```



# SUSIEJIMO ANOTACIJŲ PAVYZDYS

```
public class Student {  
    private Integer age;  
    private String name;  
    @Autowired(required=false)  
    public void setAge(Integer age) { this.age = age; }  
    public Integer getAge() { return age; }  
    @Autowired  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getName() {  
        return name;  
    }  
}
```



## UŽDUOTIS 1 - ANOTACIJOS

- Tęsiame nuo FirstSpringProject Quickstart projekto, kurį sukūrėme prieitą paskaitą
- Igaliinti konfigūraciją anotacijomis
- Nuskaityti produktų bean kolekciją per anotacijas į java klasę. Patarimai:
  - xml faile galima sukurti daug produktų bean su ta pačia klasė net ir be pavadinimo
  - autowire metodui setProducts
- Pasinaudojant PostConstruct ir PreDestroy atspaudinti kolekciją į ekrana



# **BEAN SUKŪRIMAS ANOTACIJOMIS**

# BEAN SUKŪRIMAS ANOTACIJOMIS

- bean sukūrimą per anotacijas galima iš konfigūracijos klasės

```
@Configuration  
@ComponentScan("lt.itmokymai.spring")  
public class AppConfig {  
    ...  
}
```

arba XML byloje

```
<context:component-scan base-package="lt.itmokymai.spring"/>
```

- kuriamas objektas privalo turėti tuščią konstruktorių
  - tokį ir turi, kol neaprašome kitokio



## ĮGALINTI BEAN KŪRIMĄ

- @ComponentScan įgalina šias anotacijas:
  - @Bean @Component @Repository @Service @Controller @Configuration
  - taip pat ir @Autowired bei @Qualifier
- Spring Boot naudoja @SpringBootApplication anotaciją, kuri extend'ina @ComponentScan
  - basePackage nurodomas - paimamas tos klasės package, ant kurios naudojama anotacija



# PAGRINDINĖS ANOTACIJOS

- **@Component** - tėvinė anotacija kitoms, bazinis bean
  - rasta klasę skenuojant classpath taps bean
- **@Repository** - skirta DAO (DB) servisui
  - sukonfigūravus, JPA, Hibernate ir kt. exception'us springas paverstų DataAccessException
- **@Service** - servisų sluoksnis biznio logikai
- **@Controller** - pažymi MVC valdiklį
  - leidžia viduje naudoti @RequestMapping
- **@Configuration** - žymi konfigūracijos komponentą
  - viduje leidžia kurti bean per metodus su @Bean



## APIMTYS

- Apimtj (scope) galima nustatyti anotacija @Scope
- Kartais Spring nesupranta, kaip nustatyti apimtj, pvz. bean ne visda persikuria pagal reikalavimus. Tam pakeisti yra du variantai:

```
@Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET_CLASS)
```

- arba pasiimti bean tiesiogiai kviečiant Spring

```
@Autowired ApplicationContext ctx;  
@RequestMapping(value = "/calc", method = RequestMethod.GET)  
public void calc() throws Exception {  
    ((TestClass) ctx.getBean(TestClass.class)).doSomething(); }
```



## APIMTYS

- Kad būtų paprasčiau ir nereiktu patiems apsirašyti proxyMode, yra sukurtos ir atskiros anotacijos
  - @RequestScope
  - @SessionScope
  - @ApplicationScope
- Tačiau prototype scope tokios anotacijos neturi (teisinga naudoti request)
- Apimčių pavadinimus galima gauti iš ConfigurableBeanFactory ir WebApplicationContext



# PAGRINDINĖS ANOTACIJOS

- bean iš egzistuojančių klasių galima sukurti konfigūracijos klasėje

```
@Configuration  
public class PapildomiBean {  
    @Bean  
    public ServiceA papildomasServisasA() {  
        return new ServiceA();  
    }  
    @Bean  
    public ServiceA darVienasServisasA() {  
        return new ServiceA();  
    }  
}
```

- taip bean galima sukurti iš bet kokios konfigūracinės klasės



# PAGRINDINĖS ANOTACIJOS

## Autowiring By Name

```
@Configuration  
public class Config {  
  
    @Bean(autowire = Autowire.BY_NAME)  
    public ClientBean clientBean () {  
        return new ClientBean();  
    }  
    Explicitly defining a name for a bean  
    @Bean(name = "someOtherName")  
    public ServiceBean serviceBean1 () {  
        return new ServiceBean();  
    }  
  
    @Bean  
    public ServiceBean serviceBean2 () {  
        return new ServiceBean();  
    }  
}
```

```
public class ClientBean {  
    @Autowired  
    private ServiceBean someOtherName;  
}
```

```
public class ServiceBean{  
}
```

LogicBig.COM



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# PAGRINDINĖS ANOTACIJOS

Spring doesn't know which instance to inject because both beans qualify,  
so it will throw: **NoUniqueBeanDefinitionException**

```
@Configuration  
public class AppConfig {  
  
    @Bean  
    public TheBean getBean1() {  
        return new TheBean();  
    }  
  
    @Bean  
    public TheBean getBean2() {  
        return new TheBean();  
    }  
  
    ....  
}
```

```
public class TheClientBean{  
    @Autowired  
    private TheBean theBean;  
  
    .....  
}
```

LogicBig.com



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# PAGRINDINĖS ANOTACIJOS

The fix

Using `@Bean(name = .... )` in Java-config and  
`@Qualifier(...)` in bean class, will resolve the conflict.  
Now only 'bean1' qualifies.

```
@Configuration  
public class AppConfig{  
  
    @Bean(name = "bean1")  
    public TheBean getBean1() {  
        return new TheBean();  
    }  
  
    @Bean(name = "bean2")  
    public TheBean getBean2() {  
        return new TheBean();  
    }  
    ....  
}
```

```
public class TheClientBean{  
  
    @Qualifier("bean1")  
    @Autowired  
    private TheBean theBean;  
  
    .....  
}
```

LogicBig.com



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# PAGRINDINĖS ANOTACIJOS

The fix

Using `@Qualifier(...)` at both places

```
@Configuration  
public class AppConfig{  
    @Qualifier("bean1")  
    @Bean  
    public TheBean getBean() {  
        return new TheBean();  
    }  
  
    @Bean  
    public TheBean getBean2() {  
        return new TheBean();  
    }  
    ....  
}
```

```
public class TheClientBean{  
    @Qualifier("bean1")  
    @Autowired  
    private TheBean theBean;  
    .....  
}
```

LogicBig.com



AKADEMIJA.IT  
INFOBALT IR TECH CITY

## UŽDUOTIS 2 - PRIKLAUSOMYBIŲ ANOTACIJOS

- Migruoti (perdaryti) ServiceA, ServiceB ir ServiceC priklausomybių pavyzdjį, kad jis naudotų tik anotacijas
- Užkomentuokite po vieną xml faile esantį bean - migruoti bus paprasčiau
  - galų gale xml faile turi likti tik component-scan žymė
  - produktus kurkite @Bean konfigūracijos bean'se
- Skaitykite VISŪ iškritusių exception'ų VISAS žinutes
  - Spring visada detaliai aprašo, kas negerai ir kur negerai



## UŽDUOTIS 2 - PRIKLAUSOMYBIŲ ANOTACIJOS

- Sukurti bean Spausdintuvas su anotacija @Service ir scope singleton
- Įsidėti Spausdintuvas priklausomybę į ServiceC
- Kodą, kuris išveda produktų pavadinimus į ekraną, iškelti į naujajį servisą
- Iš ServiceC kvesti naujajį servisą, jam paduodant produktų sąrašą, kurį naujasis servisas turi atspausdinti ekrane
  - bus du pakvietimai: PostConstruct ir PreDestroy pažymėtuose metoduose



# SPRING JUNIT TESTUOSE

# SPRING IR JUNIT PAGRINDINIAI ELEMENTAI

- JUnit 4.5+ versijai naudokite `SpringJUnit4ClassRunner` klasę.
- Aplikacijų konteksto palaikymas: `@ContextConfiguration`.
- Priklasomybių injekcija: `@Autowired`, `@Qualifier`, `@Inject` ir t.t.
- Transakcijų palaikymas: `@Transactional`, `@BeforeTransaction`, `@AfterTransaction`, `@TransactionConfiguration`, `@Rollback`.



# SPRING IR JUNIT PAVYZDYS

- Pvz. su Spring, Junit, Autowired, SpringJUnit4ClassRunner, ContextConfiguration locations

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/simple-job-launcher-context.xml",
"/jobs/skipSampleJob.xml" })
public class SkipSampleFunctionalTests {
    @Autowired
    private JobLauncherTestUtils jobLauncherTestUtils;
    private SimpleJdbcTemplate simpleJdbcTemplate;
    @Autowired
    public void setDataSource(DataSource dataSource) {
        this.simpleJdbcTemplate = new SimpleJdbcTemplate(dataSource);
    }
    @Test
    public void testJob() throws Exception {
        simpleJdbcTemplate.update("delete from CUSTOMER");
        for (int i = 1; i <= 10; i++) {
            simpleJdbcTemplate.update("insert into CUSTOMER values (?, 0, ?, 100000)",
                i, "customer" + i);
        }
        JobExecution jobExecution = jobLauncherTestUtils.launchJob().getStatus();
        Assert.assertEquals("COMPLETED", jobExecution.getExitStatus());
    }
}
```



# SPRING TEST PRIKLAUSOMYBĖS

- Spring testavimui reikalingas spring-test artefaktas

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.0.10.RELEASE</version>
    <scope>test</scope>
</dependency>
```

- taip pat reikia migruoti junit versiją iš 3 į 4

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```



# JUNIT TESTŪ VARIANTAI

- Unit testai testuoja vienos klasės funkcionalumą
  - jie skirti atrasti bazinio funkcionalumo klaidas
- Integraciniai - testuoja daugiau nei 1 klasės funkcionalumą; kodo vienetų integraciją tarpusavyje
  - Spring integruoja klasses tarpusavyje, todėl Spring Junit testai yra labiau integraciniai negu unit testai
  - Integraciniai testai dažnai konfigūruojami pom.xml profilyje, su juo leidžiami mvn verify fazėje
- E2E (end-to-end) - testuoja verslo scenarijus taip, kaip verslas/klientas juos naudos realybėje
  - pvz., Rest servisai bei puslapis naršyklėje



## UŽDUOTIS 3 - SPRING JUNIT

- Igaliinti Spring Junit teste ir patikrinti, kad klasė ServiceC ar kita teisingai nuskaitė produktų sąrašą
- Kontekstas bus tas pats "/application-context.xml"



## NAUDINGOS NUORODOS

- Pagrindinis puslapis: <http://www.springsource.org/>
- Dokumentacija:
  - <http://www.springsource.org/documentation>
  - <http://spring.io/docs>
- Forumas: <http://forum.springsource.org/>
- <http://www.springbyexample.org/examples/>



# SPRING BOOT

## IŽANGA Į SPRING BOOT

*Spring Boot - tai technologija, palengvinanti aplikacijų kūrimą Spring karkaso pagrindu. Programuotojas be didelių Spring karkaso žinių, gali gana sparčiai pradėti vystymo darbus. Ji veikia pateikdama iš anksto paruošą numatytają konfigūraciją ir leidžia ją paprastai keisti.*



# ĮŽANGA Į SPRING BOOT

- Šio projekto tikslai:
  - Pateikti radikliai greitesnę ir plačiau prieinamą galimybę pradėti darbus su Spring karkasu;
  - Pateikti iš anksto apibrėžtą rinkinį numatytyjų nustatymų, tačiau leisti juos lengvai pakeisti;
  - Pateikti nefunkcinių priemonių rinkinį, kurios yra dažniausiai naudojamos (jdėtiniai serveriai, apsauga, metrikos, konfigūracijos išnešimas)
  - Visiškai jokio automatinio kodo generavimo ir konfigūravimo XML.



## REIKALAVIMAI

- Spring Boot (>= 2.0.6.RELEASE)
- Spring Framework (>= 5.0.10.RELEASE)
- Java 8
- Visos versijos tarpusavyje suderinamos
- Panaudojus nesuderinamas, gali tekti spręsti kurio nors framework'o bug'us



# PROJEKTO PARUOŠIMAS NAUDOJANT MAVEN

- Nudojame archetipą, daug kas jau yra paruošta
- Archetipe sukonfigūruotas Spring Boot web aplikacijoms
  - `spring-boot-starter-parent` ir `spring-boot-starter-web` web aplikacijai
  - `spring-boot-maven-plugin` vykdomajam (executable) jar pagaminti
  - `spring-boot-starter-test` Spring Boot testuoti
- Starter artifakte sukonfigūruotas Spring Boot ir Spring
- Norėdami patys įsidėti Spring Boot į aplikaciją, turėtume visa tai konfigūruoti



# SPRING BOOT HELLO WORLD

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController // rest valdiklis
@EnableAutoConfiguration // aplikacijai
public class HelloWorld {
    @RequestMapping("/") // užklausos
    String home() {
        return "Hello World!";
    }
    public static void main(String[] args) throws Exception {
        SpringApplication.run(HelloWorld.class, args); // aplikacija
    }
}
```



## ANOTACIJA @RESTCONTROLLER

- Tai stereotipinė (angl. stereotype) anotacija. Ji suteikia papildomą informaciją tiek programuotojams, kurie skaito kodą, tiek pačiam Spring karkasui ir nurodo, kad ši klasė atlieka konkretų vaidmenį - šiuo atveju tai web kontroleris. Spring karkasas apdorodamas HTTP užklausas remsis informacija iš tokiomis anotacijomis pažymėtu klasiu.



## ANOTACIJA @REQUESTMAPPING

- Ši anotacija suteikia maršrutizavimo informaciją. Šiuo atveju, ji nurodo Spring karkasui, jog visas HTTP užklausas, kurių kelias yra “/” apdoros metodas “home”.
- @RestController anotacija nurodo, jog String tipo rezultatas turėtų būti išspausdinamas klientui toks koks yra.



## ANOTACIJA @ENABLEAUTOCONFIGURATION

- Ši klasės lygio anotacija nurodo Spring karkasui “atspėti” konfigūraciją, remiantis priklausomybėmis (jar bibliotekos), kurias programuotojas įtraukė į projektą.
- Šiuo atveju ji veikia kartu su main metodu



## MAIN METODAS

- Programos kontrolė deleguojama statiniam klasės `SpringApplication` metodui `run`, nurodant aplikacijos šakninį komponentą. Spring karkasas paleis aplikaciją, t.y. startuos serverį su numatytaisiais parametrais.
- Kadangi mes pasinaudojome archetipu, o Jame aplikacija jau yra sukonfigūruota `App.java` ir vliau jau mūsų sutvarkyta Tomcat'ui, `@EnableAutoConfiguration` bei `main(args)` mums Rest valdiklyje nereikalinga



# SPRING BOOT HELLO WORLD

- Galime sutrumpinti kodą:

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorld {
    @RequestMapping("/")
    String home() {
        return "Hello World!";
    }
}
```



## UŽDUOTIS 4 - PAPRASTAS REST SERVISAS

- Grįžtame prie Spring Boot Starter projekto ir sukuriame klasę RestServisas, pažymime ją @RestController
- Sukuriame metodą getProductsCollection, pažymime jį @RequestMapping su "/productsCollection"
- Nusikopijuojame iš FirstSpringProject klasę Product ir beans'us iš java konfigūracijos klasės
- Įvedame į RestServisas produktų sąrašo priklausomybę ir getProductsCollection grąžiname produktų pavadinimus vienoje eilutėje
- Patikriname, ar servisas veikia ir grąžina visus produktus



## UŽDUOTIS 5 - CONTEXT XML PRIJUNGIMAS

- Prijungsime prie Spring Boot projekto context XML
- Konfigūracijai AppConfig.java reikia pridėti anotaciją:

```
@ImportResource({ "classpath*:application-context.xml" })
```

- /src/main/resources reikia sukurti application-context.xml su <beans ...> žyme
- viduje sukuriame dar keletą <bean...> produktų
- iš naujo build'iname aplikaciją, pastartuojame ir patikriname, ką grąžina "/productsCollection"
  - produktais tiek iš java konfigūracijos, tiek iš XML



# **REST SERVİSÜ KŪRIMAS. CRUD**

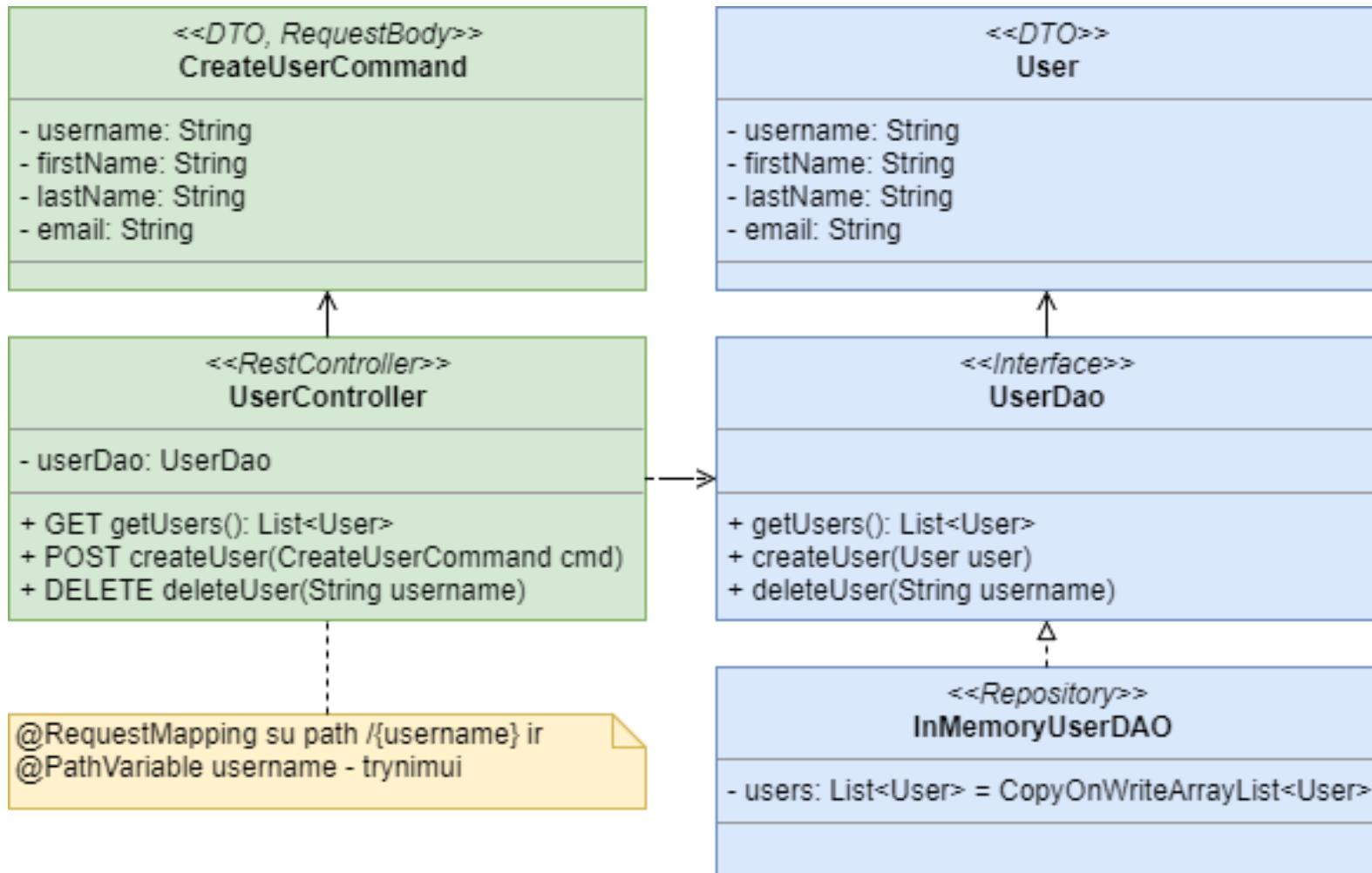


## KURIAME SERVISĄ > SERVISO APRAŠYMAS

- Servisas /api/users :
  - GET - grąžins regiszruotų vartotojų sąrašą
  - POST - sukurs naują regiszruotą vartotoją
  - DELETE - ištrins regiszruotą vartotoją
- Iki pilnos CRUD aplikacijos trūksta tik update, pvz. per HTTP PUT metodą
  - CRUD = Create/Read/Update/Delete
  - bet update galima realizuoti ir pačiame POST



# DIZAINAS



# KURIAME SERVISA > MODELIS > VARTOTOJAS (ANGL. USER)

- DTO objektas, pvz. UserDto (dažnai nenaudojam Dto)
- Data Transfer Object - skirtas duomenų perdavimui

```
package it.akademija.model;
public final class User {
    private String username;
    private String firstName;
    private String lastName;
    private String email;
    public User() {}
    public User(String username, String firstName, String lastName, String email) {
        this.username = username;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    } // toliau - get ir set metodai
    public String getUsername() {
        return username;
    }
    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public String getEmail() { return email; }
}
```



# KURIAME SERVISA<sub>č</sub> > SERVISAS > VARTOTOJŲ SĄRAŠO GAVIMAS

- `@RequestMapping` galime naudoti ir klasėje
- Taip pat galime nurodyti metodą, pvz. GET

```
package it.akademija.controller;
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
    /* Apdoros užklausas: GET /api/users */
    @RequestMapping(method = RequestMethod.GET)
    public List<User> getUsers() {
        return Collections.emptyList();
    }
}
```

- <http://localhost:8081/api/users>

# KURIAME SERVIA> MODELIS > VARTOTOJO REGISTRAVIMAS

- Sukuriame CreateUserCommand skirtą priimti duomennis iš Rest serviso

```
public final class CreateUserCommand {  
    private String username;  
    private String firstName;  
    private String lastName;  
    private String email;  
    // toliau - get ir set metodai  
    public String getUsername() {  
        return username;  
    }  
    public void setUsername(String username) {  
        this.username = username;  
    }  
    public String getFirstName() {  
        return firstName;  
    }  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
    public String getLastName() { return lastName; }  
    public void setLastName(String lastName) { this.lastName = lastName; }  
    public String getEmail() { return email; }  
    public void setEmail(String email) { this.email = email; }  
}
```



# KURIAME SERVISĄ > SERVISAS > VARTOTOJO REGISTRAVIMAS

- `@ResponseStatus` nurodo kokį HTTP kodą grąžinti
- `@RequestBody` nurodo, kokios informacijos tikimės iš serviso kvietėjo

```
package lt.itakademija.controller;
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
    /* Sukurs vartotoją ir grąžins atsakymą su HTTP statusu 201 */
    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    public void createUser(@RequestBody final CreateUserCommand cmd)
        System.out.println(cmd);
    }
}
```



# KURIAME SERVIA>SERVISAS>VARTOTOJO REGISTRAVIMAS

```
HTTP POST http://localhost:8081/api/users
{
    "username": "slapyvardis",
    "email" : "userName@mail.com",
    "firstName": "Vardenis",
    "lastName" : "Pavardenis"
}
```



# KURIAME SERVISĄ > SERVISAS > VARTOTOJO TRYNIMAS

- `@RequestMapping` turi šabloną {} pavadinimu `username`
- `@PathVariable` nurodo paimti informaciją kintamojo pavadinimu iš nuorodos kelio vietas pagal šabloną

```
package lt.itakademija.controller;
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
    /* Apdoros užklausas: DELETE /api/users/<vartotojas> */
    @RequestMapping(path = "/{username}", method = RequestMethod.DELETE)
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteUser( @PathVariable final String username ) {
        System.out.println("Deleting user: " + username);
    }
}
```



```
HTTP DELETE http://localhost:8081/api/users/slapyvardis
```



AKADEMIJA.LT

# DAO SERVISO KŪRIMAS

# USERDAO - USER OBJEKTŲ REPOZITORIJA

- DAO - pattern'as (šablonas)
- Data Access Object - darbo su duomenimis servisas

```
package it.akademija.dao;  
// ...  
/*  
 * DAO - Data Access Object. Darbo su User objektais API.  
 */  
public interface UserDao {  
    List<User> getUsers();  
    void createUser(User user);  
  
    void deleteUser(String username);  
}
```



# INMEMORYUSERDAO - USERDAO REALIZACIJA

```
@Repository
public class InMemoryUserDAO implements UserDAO {
    private final List<User> users = new CopyOnWriteArrayList<>();
    @Override public List<User> getUsers() {
        return Collections.unmodifiableList(users);
    }
    @Override public void createUser(User user) { users.add(user); }
    @Override
    public void deleteUser(String username) {
        for (User user: users) {
            if (username.equals(user.getUsername())) {
                users.remove(user);
                break;
            }
        }
    }
}
```



# KURIAME SERVISA> SERVISAS

```
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
    private final UserDAO userDao; // pridedame DAO

    @Autowired
    public UserController(UserDAO userDao) {
        this.userDao = userDao;
    }
    /* Apdoros užklausas: GET /api/users */
    @RequestMapping(method = RequestMethod.GET)
    public List<User> getUsers() {
        return userDao.getUsers(); // skaitome per DAO
    }
    ...
}
```



## UŽDUOTIS 5 - USER SERVICE

- Pabaikite kurti createUser ir deleteUser motodus UserController klasėje
  - jie turi taip pat kvesti UserDao
- Ar užtenka ir servisas jau veikia?
  - klauskite, kas neaišku



# INTEGRACINIAI TESTAI

## POM.XML > TESTAVIMO BIBLIOTEKOS

- Tam, kad nekiltų klausimo, ar po kūrimo ir vėliau keičiant servisą kas nors tikrai veikia, reikia rašyti testus
- Bibliotekos skirtos testavimui
  - spring-boot-starter-test
  - junit
- Bibliotekas jau turime - jas sukonfigūravo pom.xml faile archetipas



# POM.XML > INTEGRACINIŲ TESTŲ PALEIDIMO PLUGINAS

- Pluginas skirtas paleisti integracinius testus:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <executions>
    <execution>
      <phase>verify</phase>
      <goals>
        <goal>integration-test</goal>
        <goal>verify</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

- Plačiau: <http://maven.apache.org/surefire/maven-failsafe-plugin/>



# KURIAME INTEGRACINI TESTĄ

```
@RunWith(SpringRunner.class) @SpringBootTest(webEnvironment =
    SpringBootTest.WebEnvironment.RANDOM_PORT, classes = { App.class })
public class UserControllerIT {
    private static final String URI = "/api/users";
    @Autowired private TestRestTemplate rest;
    @Test public void createsUserThenRetrievesUserListAndDeletesUser()
        final String username = "testUsername";
        final CreateUserCommand createUser = new CreateUserCommand();
        createUser.setUsername(username);
        rest.postForLocation(URI, createUser);
        List<User> users = rest.getForEntity(URI, List.class).getBody();
        Assert.assertThat(users.size(), CoreMatchers.is(1));
        rest.delete(URI + "/" + username);
        users = rest.getForEntity(URI, List.class).getBody();
        Assert.assertThat(users.size(), CoreMatchers.is(0));
    }
}
```



## STARTUOJAME APLIKACIJĄ IŠ NAUJO

- Vietoj “package” ar “install”, naudojame “verify” tam, kad būtų paleisti integraciniai testai.

```
$ mvn clean verify
```



## UŽDUOTIS 6 - PARDUOTUVĖ

- Galite pradėti kurti parduotuvės servisus
  - CartController
  - ProductsController
  - ProductsDAO
  - ir kt.
- .. kartu su reikalingomis papildomomis klasėmis, tokiomis kaip Cart, Product, User ir pan.
- Kitą paskaitą aptarsime vieną iš būdų, kaip galima sukurti parduotuvės servisus



# SERVISO DOKUMENTACIJOS GENERAVIMAS



## POM.XML > SERVISŲ DOKUMENTAVIMO BIBLIOTEKOS

- į pom.xml reikia įtraukti bibliotekas REST servisu dokumentacijai

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.5.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.5.0</version>
</dependency>
```

# DOKUMENTACIJOS KONFIGŪRACIJA

- į App.java pridėti docker'į

```
@EnableSwagger2
@SpringBootApplication
public class App {
    @Bean public Docket swaggerDocket() {
        return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(apiInfo())
            .select()
            .apis(RequestHandlerSelectors.basePackage("it.akademija"))
            .build();
    }
    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            .title("IT Akademija REST Documentation")
            .version("0.0.1-SNAPSHOT")
            .build();
    }
}
```



# SERVISO DOKUMENTAVIMAS

Pavyzdys, kaip galima dokumentuoti servisą UserController ir jo metodus

```
@RestController
@Api(value = "user")
@RequestMapping(value = "/api/users")
public class UserController {
    @RequestMapping(method = RequestMethod.GET)
    @ApiOperation(value="Get users",notes="Returns registered users")
    public List<User> getUsers() {
        return userDao.getUsers();
    }
    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    @ApiOperation(value="Create user",notes="Creates user with data")
    public void createUser(@ApiParam(value="User Data",required=true)
        @RequestBody final CreateUserCommand cmd) { .. }
}
```



# KAIP TAI ATRODYS

## IT Akademija REST Documentation

hello-controller : Hello Controller

Show/Hide | List Operations | Expand Operations

my-controller : My Controller

Show/Hide | List Operations | Expand Operations

user-controller : User Controller

Show/Hide | List Operations | Expand Operations

GET /api/users

Get users

POST /api/users

Create user

DELETE /api/users/{username}

deleteUser

[ BASE URL: / , API VERSION: 0.0.1-SNAPSHOT ]

user-controller : User Controller

Show/Hide | List Operations | Expand Operations

GET /api/users

Get users

Implementation Notes

Returns registered users

POST /api/users

Create user

Implementation Notes

Creates user with data

Parameters

Parameter	Value	Description	Parameter Type	Data Type
createUserCommand	(required)	User Data	body	Model   Model Schema {} "email": "string",



## UŽDUOTIS 7 - SWAGGER

- Prisidėkite swagger savo aplikacijai
  - dokumentuokite savo servisus @ApiOperation anotacija
  - pasitikrinkite, ar teisingai rodoma jūsų servisu dokumentacija
- Pasiekiame per <http://localhost:8081/swagger-ui.html>



## DAUGIAU INFO APIE SWAGGER

- Swagger'io dokumentacija
  - <http://swagger.io/>
  - <http://swagger.io/specification/>
  - <https://github.com/swagger-api/swagger-core/wiki/Annotations>



# DUOMENŲ VALIDACIJA

## POM.XML > VALIDAVIMO BIBLIOTEKOS

- Validaciją, arba duomenų patikrinimą, atliekame React
- Validuoti galime ir Java kode
- Java turi JSR-380 standartą
  - viena iš implementacijų hibernate-validator
- Mūsų archetipe jis jau yra įtrauktas per Spring Boot Starter
- Plačiau:
  - <http://hibernate.org/validator/documentation/>
  - <https://docs.jboss.org/hibernate/validator/6.0/api/>



# VALIDUOJAME DUOMENIS > HIBERNATE ANOTACIJOS

```
public final class CreateUserCommand {  
    @NotNull  
    @Length(min = 1, max = 30)  
    private String username;  
    @NotNull  
    @Length(min = 1, max = 100)  
    private String firstName;  
    @NotNull  
    @Length(min = 1, max = 100)  
    private String lastName;  
    @NotNull  
    @Length(min = 1, max = 200)  
    @Email  
    private String email;  
}
```



## ĮJUNGIAME VALIDACIJĄ

- Įjungiamo validaciją createUser metodo @RequestBody parametru i panaudodami @Valid anotaciją

```
public void createUser(  
    @ApiParam (value = "User Data", required = true)  
    @Valid  
    @RequestBody  
    final CreateUserCommand cmd) {  
    ...  
}
```



# KĄ MATYSIME?

- Pvz. jeigu blogai įvesime email:

```
{  
    ...  
    "status": 400,  
    "error": "Bad Request",  
    "exception":  
        "org.springframework.web.bind.MethodArgumentNotValidException",  
    ...  
    "defaultMessage": "not a well-formed email address",  
}
```



## UŽDUOTIS 8 - VALIDACIJA

- Pridėkite validaciją klasių, kurios dalyvauja kaip RequestBody controller servisuose, atributams
  - mūsų atveju kol kas tik CreateUserCommand
- Kas atsitinka įvykus klaidai? Pvz. blogai įvedus email arba padavus tuščią vardą



# KITOJE PASKAITOJE

Aplikacijos projektavimas. A-Z



# IŠ PRAEITOS PASKAITOS - CORS IŠJUNGIMAS

- Cross-origin resource sharing
  - Išjungiame visiems serverio URL "/\*\*" (mapping), visiems metodams "\*" (pvz. Options preflight check), patikrinimą taip, kad visi "\*" šaltiniai (origins) būtų leidžiami
  - leidžiame allowCredentials=true siųsti cookies informaciją tarp skirtinį domenų

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**").allowedMethods(" * ")
            .allowedOrigins(" * ").allowCredentials(true);
    }
}
```

- Daugiau informacijos [Wiki CORS](#), Spring CORS konfigūracija



# IDE

- Eclipse
  - Window->Show View: Problems ir Error Log
  - Full text search Ctrl+Shift+L
    - Ieškoti Marketplace [Quick Search For Eclipse](#)
  - PlantUML <http://plantuml.com/eclipse>
    - Help->Install New Software... ir prisidėti <http://hallvard.github.io/plantuml/>
      - taip pat sudo apt-get install graphviz
  - ObjectAID <http://objectaid.com/installation>
    - Help->Install New Software... ir prisidėti <http://www.objectaid.com/update/current>



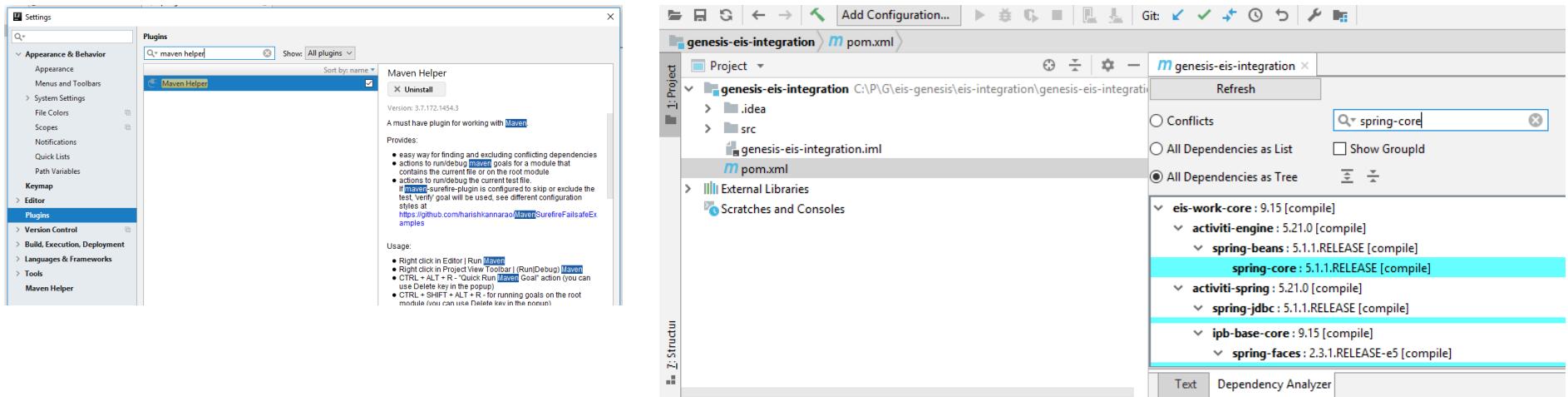
# IDE

- Eclipse shortcuts
  - Ctrl+1 (auto get/set; source generation)
  - Ctrl+Shift+L (full-text search)
  - Ctrl+Shift+F (auto formatting)
  - Ctrl+Shift+O (auto imports)
- Idea
  - Full text search Ctrl+H arba Ctrl+Shift+F
  - Show Diagram ant klasės (bent jau Pro versijoj)
- STS - Spring Tool Suite (preconfigured for Spring Boot)
- Visual Code
  - STS plugin <https://spring.io/tools4>



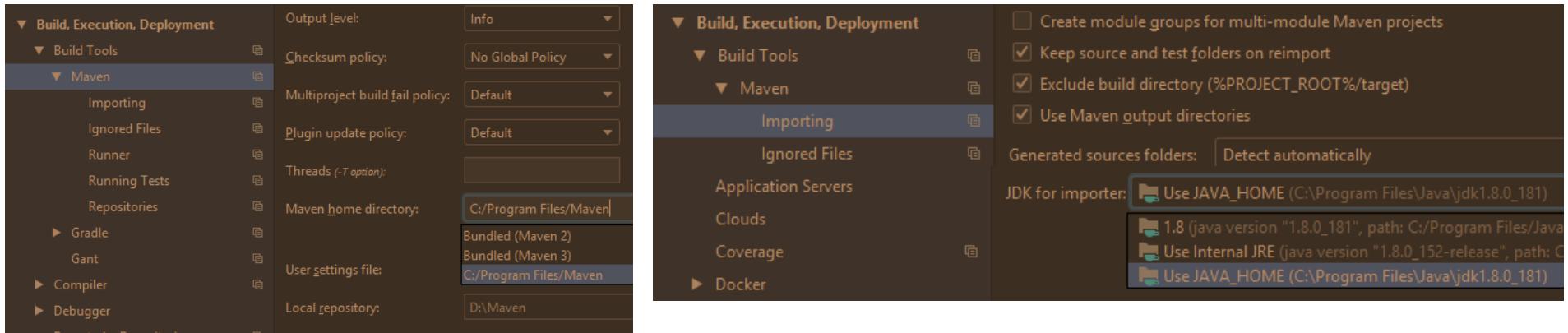
# IŠ PRAEITOS PASKAITOS - IDEA DEPENDENCIES

- kažką panašaus į Eclipse Dependency Hierarchy galite gauti Idea įrašę Maven Helper plugin'ą



# IŠ PRAEITOS PASKAITOS - IDEA CONFIG

- kažką panašaus į Eclipse kad naudotų OS Maven ir JDK



## SKRIPTAI LEISTI APLIKACIJAI

- patarimas būtų pasigreitinti savo darbą, t.y. serverio perleidimą
- npm - pakeitus package.json paleidžiamas npm install ir npm start, o nekeičiant to failo tiesiog npm start
- maven sudėtingiau, nes ilgos eilutės, portas per parametra, įvairūs būdai sukurti darinj, rekomenduoju susikurti skriptus patogesniam paleidimui, pvz.
- ./skriptas.sh su vykdymo teisėmis

```
#!/bin/sh
mvn clean install spring-boot:run -Dspring-boot.run.arguments[--server.port=8081]
mvn clean install org.codehaus.cargo:cargo-maven2-plugin:1.7.0:run \
-Dcargo.maven.containerId=tomcat8x -Dcargo.servlet.port=8081 \
-Dcargo.maven.containerUrl=http://repo1.maven.org/maven2/org/apache/tomcat/tomcat/8.5.35/tomcat-8.5.35.zip
```





# AKADEMIJA.IT

INFOBALT IR TECH CITY

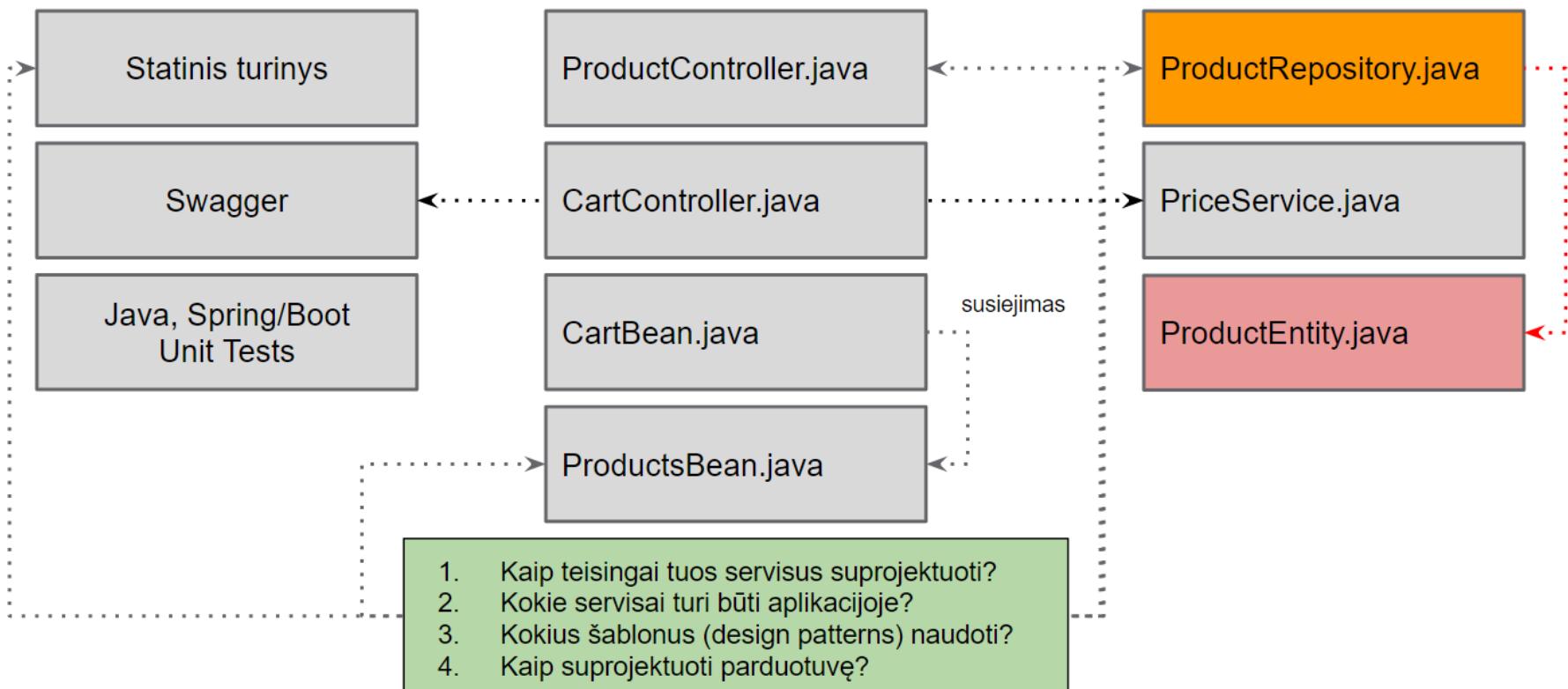
## APLIKACIJOS PROJEKTAVIMAS. PROGRAMAVIMO ISTORIJA

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# KĄ JAU MOKAME IR KO DAR NE



# TURINYS

- Programavimo istorija
- Aplikacijos kūrimas
  - reikalavimai
  - planavimas ir atlikimas
  - projektavimas

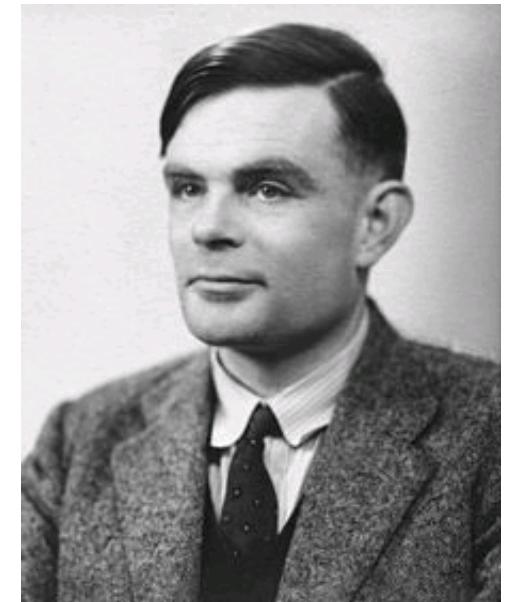


# PROGRAMAVIMO ISTORIJA

# PROGRAMAVIMO TĖVAS

- Iš pradžių buvo duomenys
- Duomenys nebuvo programos, tai buvo tik skaičiai, 0 ir 1, vėliau ir tekstas, bet nieko daugiau.. o programos?
- Matematikas Alan Turing, kuris mokėsi Cambridge, Kings kolegijoje 1931-1934, dar būdamas 22 metų įrodo savo pirmąjį matematinę teoremą
- Vėliau studijuoja universitete Princeton'e (New Jersey), 1936-1937

Alan Turing



1912-1954



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# ĮRAŠOMŲ PROGRAMŲ IŠRADĖJAS

- Turingas 1936 m. parašo mokslinį darbą, aprašydamas “universalią skaičiavimo mašiną” (Turingo mašiną)
  - paaiškina CPU funkcijas
  - manipuliuoja simbolių eilutėmis
  - tinka 100% visiems algoritmams
- John von Neumann 1936 m. Kembridže susipažista su Turingu, kartu jie diskutuoja apie “programas”
- Dirba prie Manhatano atominės bombos projekto, renka taikinius Japonijoje

John von  
Neumann



1903-1957



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# TUO TARPU KITAME ŽEMYNE

- 1936 m. atmestame moksliniame darbe vokietis Konrad Zuse taip pat aprašo programas, tačiau jos nėra Turing-complete (tinka ne visiems algoritmams)
- Zuse 1936 m. sukuria Z1 kompiuterį, skaičiuoja lėktuvų sparnų aerodinamiką
- 1941 m. pristato Z3, kuris teoriškai yra Turing-complete, bet tai išsiaiškinta tik 1998
  - Z3 neturi IF sakinio analogo, todėl nėra praktiškai pritaikomas

Konrad Zuse

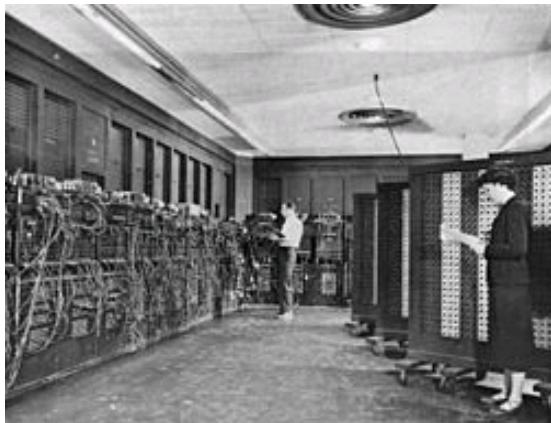


1910-1995



AKADEMIJA.LT  
INFOBALT IR TECH CITY

# PIRMAS EL. KOMPIUTERIS



- J. Presper Eckert and John Mauchly perskaitė Turingo darbą 1943 m. pasiūlo, o 1946 m. ir sukuria pirmą veikiantį elektroninį kompiuterį ENIAC už daugiau nei \$7,1 mil. dolerių 2018 m. kainomis
- Jį galima “perprogramuoti” per kelias savaites naudojant mygtukus ir laidus
  - programos pusiau mechaninės
- Pilnai atitinka Turingo teoretinį kompiuterį
- “Programuoja” šešios moterys
- Pirmoji programa - vandenilio atominei bombai



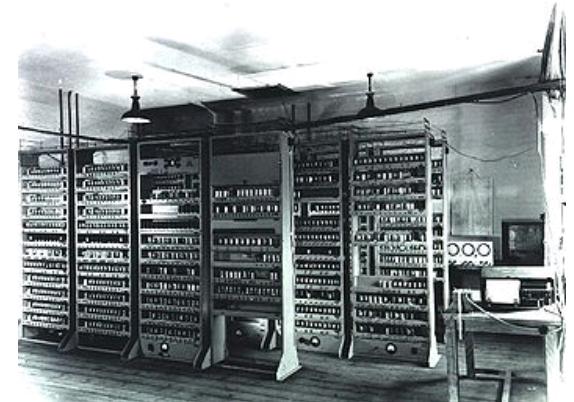
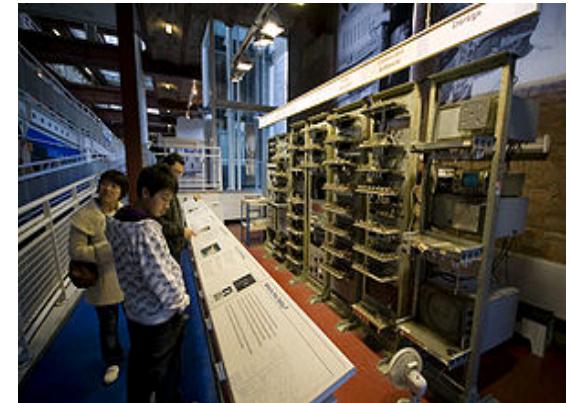
## VON NEUMANNO ARCHITEKTŪRA

- 1944 m. Neumann'as susitinka su ENIAC kūrėjais ir jie kartu sugalvoja ir pasiūlo įrašomas programas (vietoj duomenų)
- 1945 m. Neumann'as visa tai užrašo ir išplatina darbe First Draft, kuriame yra popieriuje suprojektuotas EDVAC kompiuteris
- Tuo pačiu metu Turingas kuria savo darbą apie įrašomas programas (Elektroninį Skaičiuotuvą), bet išleidžia tik 1946 m. ir kadangi Neumann'o darbas išleistas anksčiau, tai ir kompiuterių architektūra su įrašytomis programomis įgauna von Neumanno architektūros pavadinimą



# PIRMOJI PROGRAMA - ASEMBLERIU

- SSEM, arba "Baby", pirmoji paleidžia įrašytą programą birželio 21, 1948, Manchesterio Universitete, Anglijoje
  - S, C1 (000), SUB S (001), Stop (111)
- 1949 balandį EDSAC (Kembridže) pirmasis programas galintis leisti praktiškai pritaikomas kompiuteris
  - H 30, S 6, T 1 (000 001 000)
- komandos (initial orders) yra sutartinės - vieną komandą sudaro vienas simbolis, programą galima įrašyti ir paleisti
- EDSAC simulatorius:  
<http://www.dcs.warwick.ac.uk/~edsac/>



# LINK AUKŠTESNIO LYGIO

- **1950 - 1960 pirmosios programavimo kalbos**
- 1943-1945 Konrad Zuse sukuria ir pirmąjį aukšto lygio programavimo kalbą Plankalkül (Plan Calculus)
  - pusiau asembleris, pusiau ALGOL
    - P2 max (V0[:8.0], V1[:8.0]) → R0[:8.0]  
END
- 1952 IBM pasiūlo FORTRAN
  - IF (IB+IC-IA) 777,777,799
- 1958 sukuriama ALGOL (panaši į Pascal)
  - for i:=1 step 1 until N do
  - toliau 30 metų naudojama mokymo institucijose
- 1964 John G. Kemeny ir Thomas E. Kurtz sukuria BASIC
- AT&T kūria C nuo 1969

Basic kūrėjai



John G. Kemeny ir  
Thomas E. Kurtz

# IKI C KALBOS

- 1960 - 1970 kuriamos fundamentinės paradigmos
- BCPL 1966 m. Kembridže Martin Richards sukuria pirmąjį programavimo kalbą su { ir } skliausteliais
  - FOR I = 1 TO 12 DO \$( \$)
  - FOR I = 1 TO 12 DO { }
- 1968 m. Niklaus E. Wirth sukuria Pascal
  - if a > b then WriteLn('Condition met')
- 1969 m. Ken Thompson su Dennis Ritchie sukuria B kalbą
  - if(a=n/b) { printn(a, b); }
- 1970 B pritaikoma UNIX sistemai
  - prieš tai UNIX OS sistema parašyta asembleriu



Niklaus E. Wirth, Pacal  
kūrėjas



# NUO C IKI JAVA

- 1980 - 1990 konsolidacija, moduliai, greitis
- 1972 UNIX kodas perrašomas į C
- C kuriama perrašant UNIX kodą
  - if (test1 > 0) { }
- 1979 m. pradedama C with Classes
- 1983 m. pervadinama į C++
- ANSI C standartas tik 1989 m.
  - pradedamas 1983 m.
  - paskutinė versija C18 2018 m.
- Oak sukuriama 1991 m., skirta televizijai
  - Oak -> Green -> Java Coffee -> Java (1994)
  - Java yra sala Indonezijoje, gaminamos kavos pupelės



Ken Thompson and Dennis Ritchie, Unix ir C kūrėjai



Bjarne Stroustrup C++ kūrėjas

# NUO JAVA IKI JSF

- 1990 - 2000 interneto amžius, skriptai, puslapiai
- 1991 m. pristatomas HTML ir Linux
- 1995 m. sukuriamas PHP - Personal Home Page
- 1996 m., JDK 1.0, Java 7 2011, Java 8 2014
  - `if (memoized.containsKey(fibIndex)) { }`
- Java sukuriama pagal C
- 1997 m. Servlet 1.0 specifikacija HTML puslapiams atsiųsti iš serverio
  - 1998 2.x
- 1999 m. JSP (HTML šablonai)
- 1999 m. sukuriama Cool -> C#
- 2002 sukuriamas Spring
- 2003 m. Scala
- 2004 m. JSF (XHTML šablonai)
  - 2013 išleidžiama JSF 2.2



James Gosling, Java kūrėjas



Anders Hejlsberg, C# dizaineris ←  
ir Rasmus Lerdorf, PHP kūrėjas →



# NUO SERVER PRIE CLIENT SIDE

- 2000-2010 funkcinis programavimas, paralelizmas, statiniai tipai, komponentai, mobile, open-source, kiti paveldėjimo/modularumo mechanizmai: mixin/trait/delegate/aspect
- 2005 F#
- 2006 JPA (DB abstrakcija)
- 2009 Go
- 2009 Servlet 3.x (2013 3.1 ir 2017 4.0)
- 2009 sukuriamas Angular
- 2010 Google perima Angular
- 2011 sukuriamas React (Facebook)
  - paremtas Angular ir PHP idėjomis
  - <https://www.youtube.com/watch?v=GW0rj4sNH2w>
- 2011 Kotlin
- 2014 Swift



Jordan Walke, React kūrėjas



Misko Hevery ← ir Adam Abrons →  
Angular kūrėjai



# NAUJOS VERSIJOS, OPEN SOURCE IR FRAMEWORK'AI

- 2010+ dabartis: naujos versijos, naujas funkcionalumas, daug kas open source, labiau framework'ai nei technologijos
- 2009 sukuriamas Node.js
  - Paypal pradeda naudoti 2013
- 2012 Google open-sourced Angular
- 2013 Facebook open-sourced React
- 2013 sukuriamas [Spring Boot](#)
  - Paypal gržta prie Spring Boot 2016
- 2017 JSF 2.3
- 2017 Java 9 ir Java EE 8
- 2017 Oracle atiduoda Java EE į Eclipse Foundation
- 2018 Oracle verčia pervadinti Java EE į Jakarta EE
  - nori kontroliuoti java brand'ą
- 2018 Java 10 (18.3) ir 11 (18.9)
- 2018 kuriamas [Deno](#) ([TypeScript](#) ant [V8](#)) vietoj Node.js
  - [10 Things I Regret About Node.js](#)



Ryan Dahl, Node.js kūrėjas



Dave Syer ← ir Phillips Web →  
Spring Boot kūrėjai

# **APLIKACIJOS KŪRIMAS. REIKALAVIMAI**



## TECHNINIAI APRIBOJIMAI/REKALAVIMAI

- REST yra ne tik programinis, bet ir architektūrinis stilius
  - REpresentational State Transfer - RESTful
- projektuojant REST servisus, taikomi tokie apribojimai:
  - vieningas interfeisas (Uniform Interface)
  - be būsenos (Stateless)
  - kešuojamai (Cacheable)
  - kliento-serverio architektūra
  - sluoksniuota sistema (Layered System)
  - gali perduoti kodą (Code on Demand; optional)



## TECHNINIAI APRIBOJIMAI/REKALAVIMAI

- REST - ką tai reiškia ir Spring pusės?
  - nėra būsenos!
  - negalime naudoti sesijų
  - sesijas galima valdyti kliento pusėje (pvz. sukurti cookie kuris galios 30 min ir pan.)
  - jei reiktų kažką saugoti serverio pusėje laikinai, tai valdyti reiktų pasitelkiant DB
- viskas, kas mums lieka - request+prototype+singleton



## TECHNINIAI APRIBOJIMAI/REKALAVIMAI

- UI - ką tai reiškia ir React pusės?
  - komponentinis UI
  - puslapis negali atsinaujinti (refresh)
  - naudosim NPM/Node
  - kodas javascript'e
  - teks build'inti ir įdëti į Spring projektą



## REIKALAVIMAI - UŽDUOTIS

- sukurti parduotuvę
- naudojami Rest servisai su Spring Boot
- UI kuriamas su React ir Bootstrap



## REIKALAVIMAI - UŽDUOTIS

- UI turi būti galima:
  - pirmame puslapyje matyti navigacijos eilutę ir produktų sąrašą
  - turi būti galima įvesti naudotojo vardą ir aplikacija su juo turi dirbti
  - produktų administravimo lange pridėti, redaguoti, ištrinti produktą
  - įdėti produktą į nurodyto naudotojo krepšelį
  - peržiūrėti krepšelį
  - ištrinti produktą iš krepšelio
  - navigacijos lange matyti, kiek prekių yra krepšelyje



## REIKALAVIMAI - UŽDUOTIS

- krepšelio reikalavimai
  - pašalinus iš krepšelio prekę turi automatiškai atsinaujinti prekių skaičius navigacijos juosteje
  - krepšelyje turi matytis visų prekių ir jų kiekių suma - galutinė suma
  - į krepšelį neturi būti galima įdėti daugiau produkto negu jo yra (quantity)



# REIKALAVIMAI - UŽDUOTIS

- produktų reikalavimai
    - turi būti papildomas filtravimo pagal pavadinimą laukas, į kurį įvedus prekės pavadinimą, būtų rodomas filtruotas prekių sąraša
    - po produktų sąrašu turi būti diagrama
    - diagramos vaizde turi būti pavaizduotos kainos pagal kiekj:
      - y ašyje - kainų intervalai: 0-10, 10-50, 50-100, 100-1000, 1000 ir daugiau
      - x ašyje - prekių kiekis pagal kainą
      - kategorijos (series) dvi: prekės, kurių title length > 10 ir description length > 10, ir kitos prekės
      - diagramai naudoti react chartjs 2
- <https://github.com/jerairrest/react-chartjs-2>



## REIKALAVIMAI - UŽDUOTIS

- Rest servisuose turi būti galima:
  - CRUD operacijos su produkту
  - gauti tiek sąrašą, tiek vieną produktą
  - CRUD operacijos su krepšeliu kiekvienam naudotojui atskirai pagal jo username
  - produktų sąrašo filtravimas pagal pavadinimą
  - turi būti pasiekiamas Swagger per /swagger-ui.html



## REIKALAVIMAI - UŽDUOTIS

- DB reikalavimai
  - turi būti panaudotas JPA darbui su DB
  - aplikacija turi būti sukurta taip, kad veiktu tiek su tuščia, tiek su užpildyta duomenų baze
  - produktų filtravimas turi vykti per DB
  - turi būti parašyta bent viena sudėtinga DB SQL užklausa
  - trinant produkta, jis turi dingti ir iš visų krepšelių



## REIKALAVIMAI - UŽDUOTIS

- ir kiti techniniai ir funkciniai reikalavimai
- įvairių metų užduočių pavyzdžiai:
  - <http://stasauskas.lt/itpro2018/4-uzduoties-egzamine-pvz.pdf>
  - <http://stasauskas.lt/itpro2018/4-uzduoties-egzamine-pvz2.pdf>



# PLANAVIMAS IR ATLIKIMAS

## **REIKALAVIMŲ SKAITYMAS**

- perskaitome visus reikalavimus
- išsiklausinėjame, kas neaišku



## KAIP PRADĘTI ..

- pirma mintis: per daug visko
- bandome pradęti rašyti kodą, ilgai užtrunka, neišeina
  - atrodo, kad dar labiau PER DAUG
- bandome iš naujo skaityti skaidres.. jų 1000+ :(
- pirmiausia **panika**
- po to **pasiduodame..**
  - o veltui!
- tai ką gi daryti, kad būtų lengviau?



## .. PROJEKTUOTI

- reikia **pasiruošti** ir **planuoti**
- reikia turėti pasiruošus sukonfigūruotą projektą
  - taip bus greičiau pradėti negu su npm create-react-app ar maven archetipu, nes jūs turėsite susikonfigūravę taip, kaip jums labiau priimtina
  - tai svarbu prieš pradedant projektą, nes pasiėmus savo projektą, smegenys jau jį pažsta, joms legviau pradėti kažką daryti negu pasiėmus svetimą
- prieš rašant kodą reikia susikurti planą
  - kiekvienas žmogus skirtingai planuoja, todėl reikia pasidaryti savo planą



# PLANAS

- pirmiausia - niekada nepuolam rašyti kodo
  - pradėjus rašyti kodą smegenys nebegali efektyviai planuoti
- pvz. planas gali prasidėti taip:
  - suplanuoti Rest API
    - GET /api/products , POST /api/products, DELETE /api/products/{productId} ...
  - suplanuoti UI nuorodas-langus
    - /products - visi produktai, /products/:productId - vienas produktas, /cart - krepšelis



## PLANAS - CRUD REST PRODUKTUI

- GET /api/products
- GET /api/products/{productId}
- POST /api/products
  - {title,desc,img,price,quantity}
- PUT /api/products/{productId}
  - {title,desc,img,price,quantity}
- DELETE /api/products/{productId}



## PLANAS - CRUD REST KREPŠELIUI

- GET /api/users/{username}/cart-products
- POST /api/users/{username}/cart-products
  - {productId, quantity}
- PUT /api/users/{username}/cart-products/{productId}
  - {quantity}
- DELETE /api/users/{username}/cart-products/{productId}

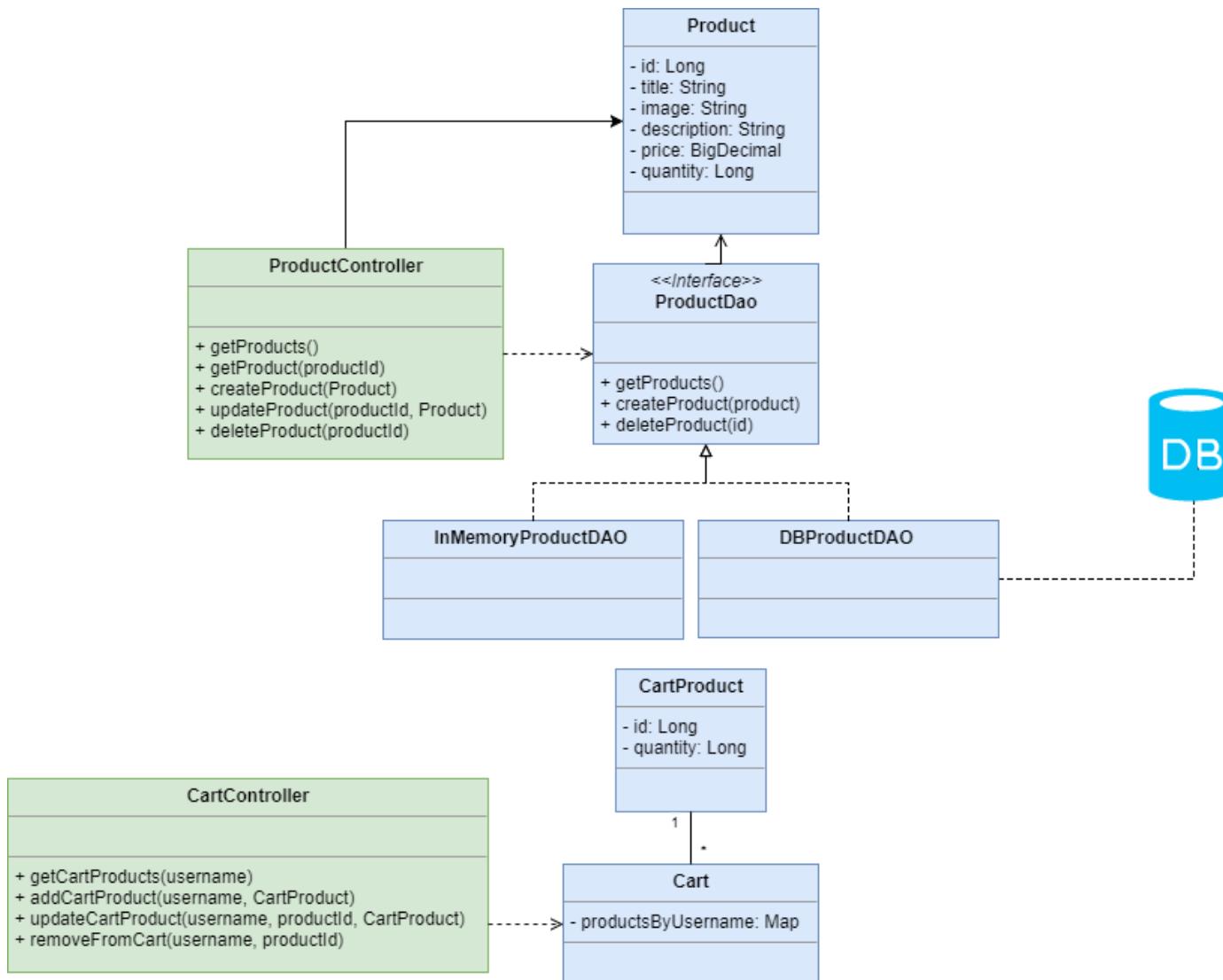


## PLANAS - NARŠYKLĖS URI-LANGAI

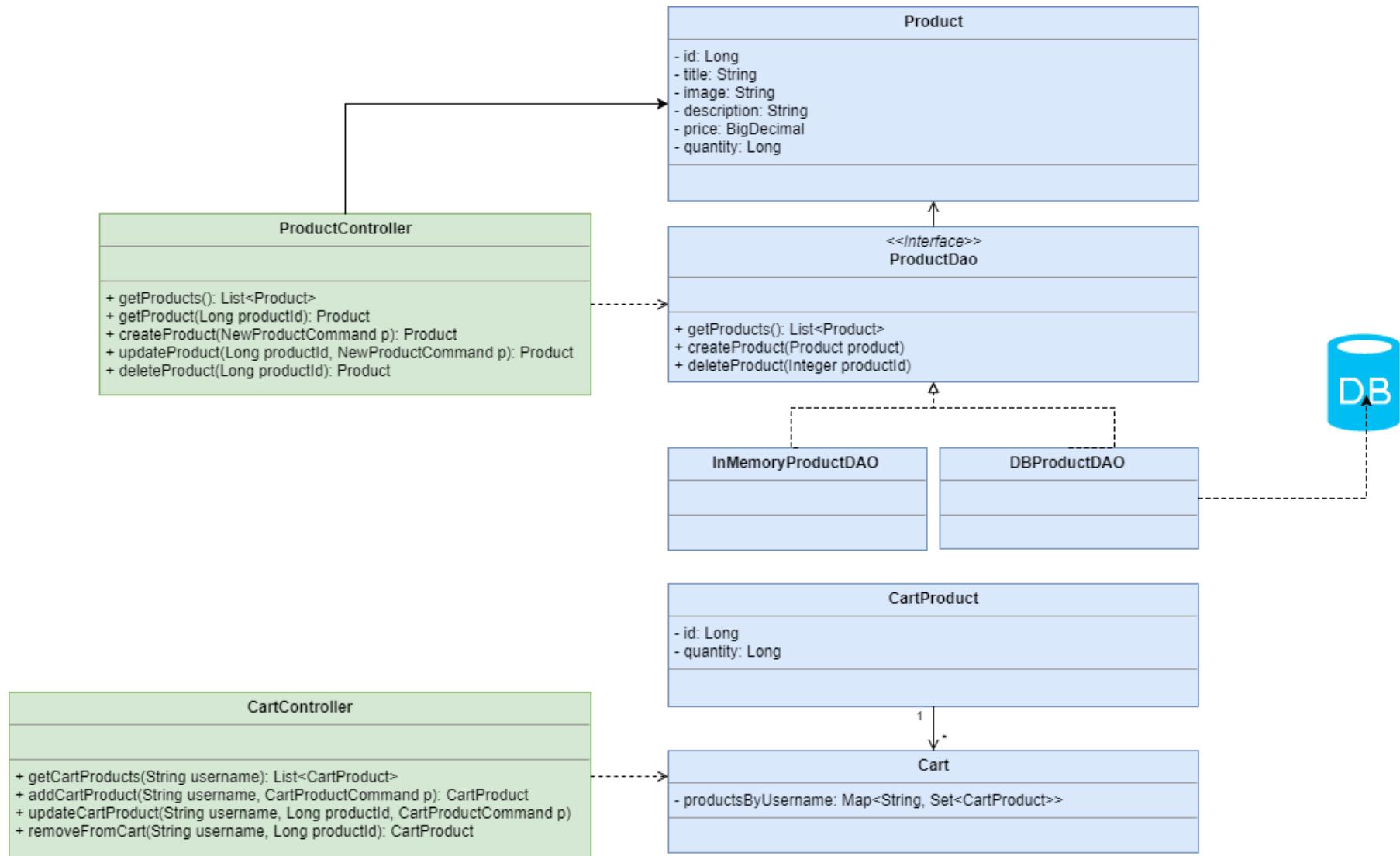
- /products - visi produktai
- /products/:productId - vienas produktas (su mygtuku įdėti į krepšelį)
- /cart - krepšelis (su kieku lauku ir galimybe keisti kiekį)
- /admin - administruojamų produktų sąrašas-lentelė
- /admin/products/new - naujo produkto forma
- /admin/products/:productId - vieno produkto keitimas



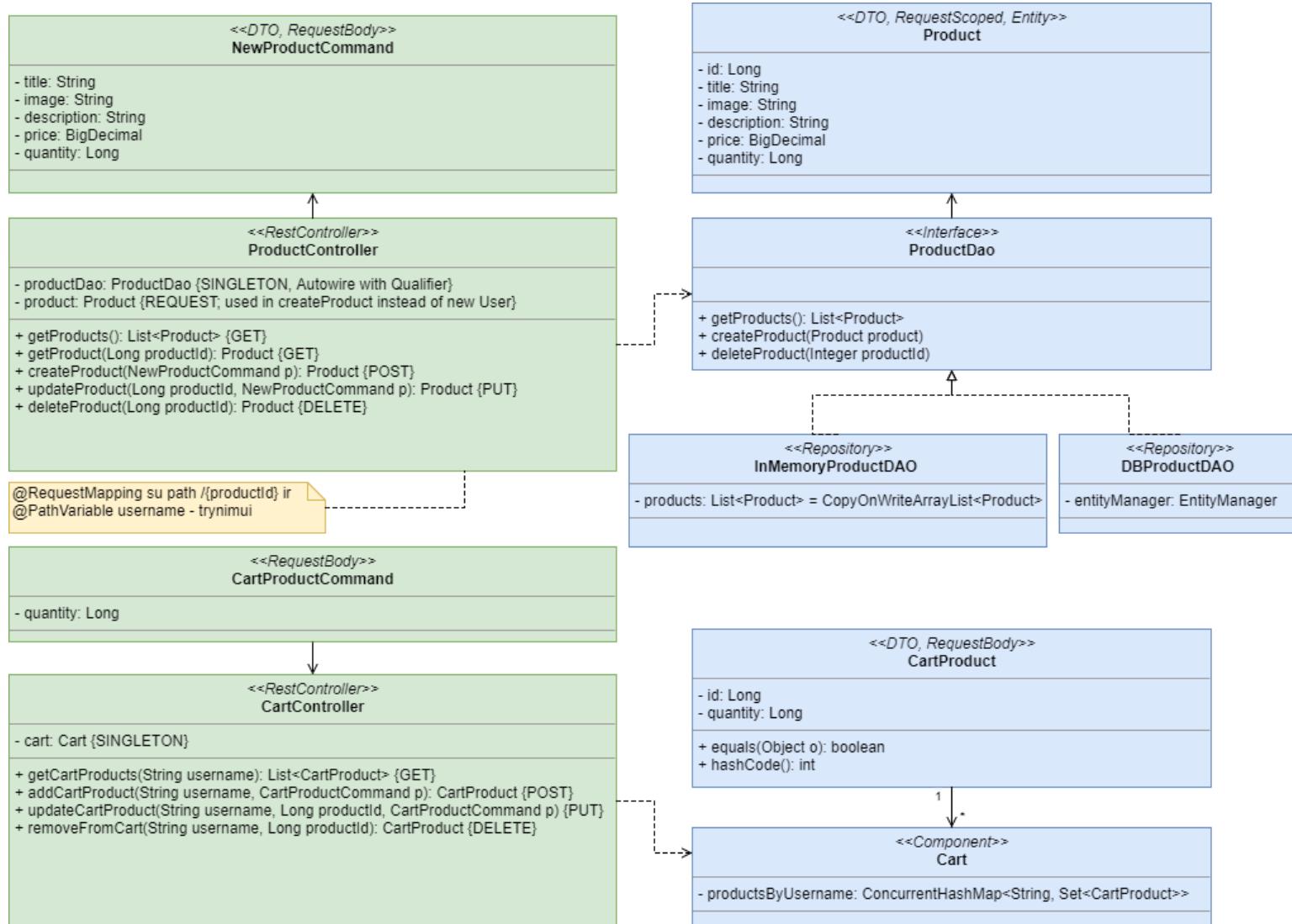
# PLANAS - SPRING JAVA



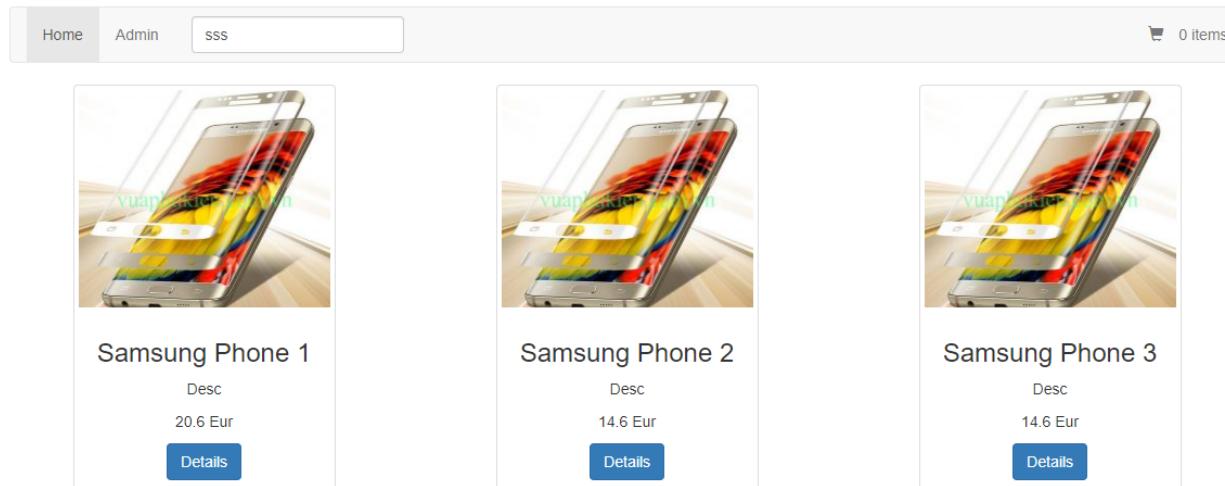
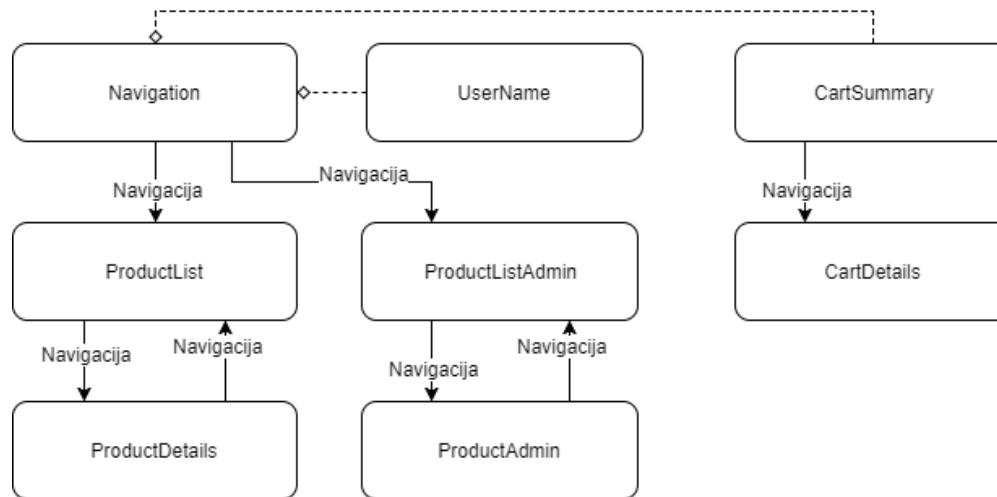
# PLANAS - SPRING JAVA



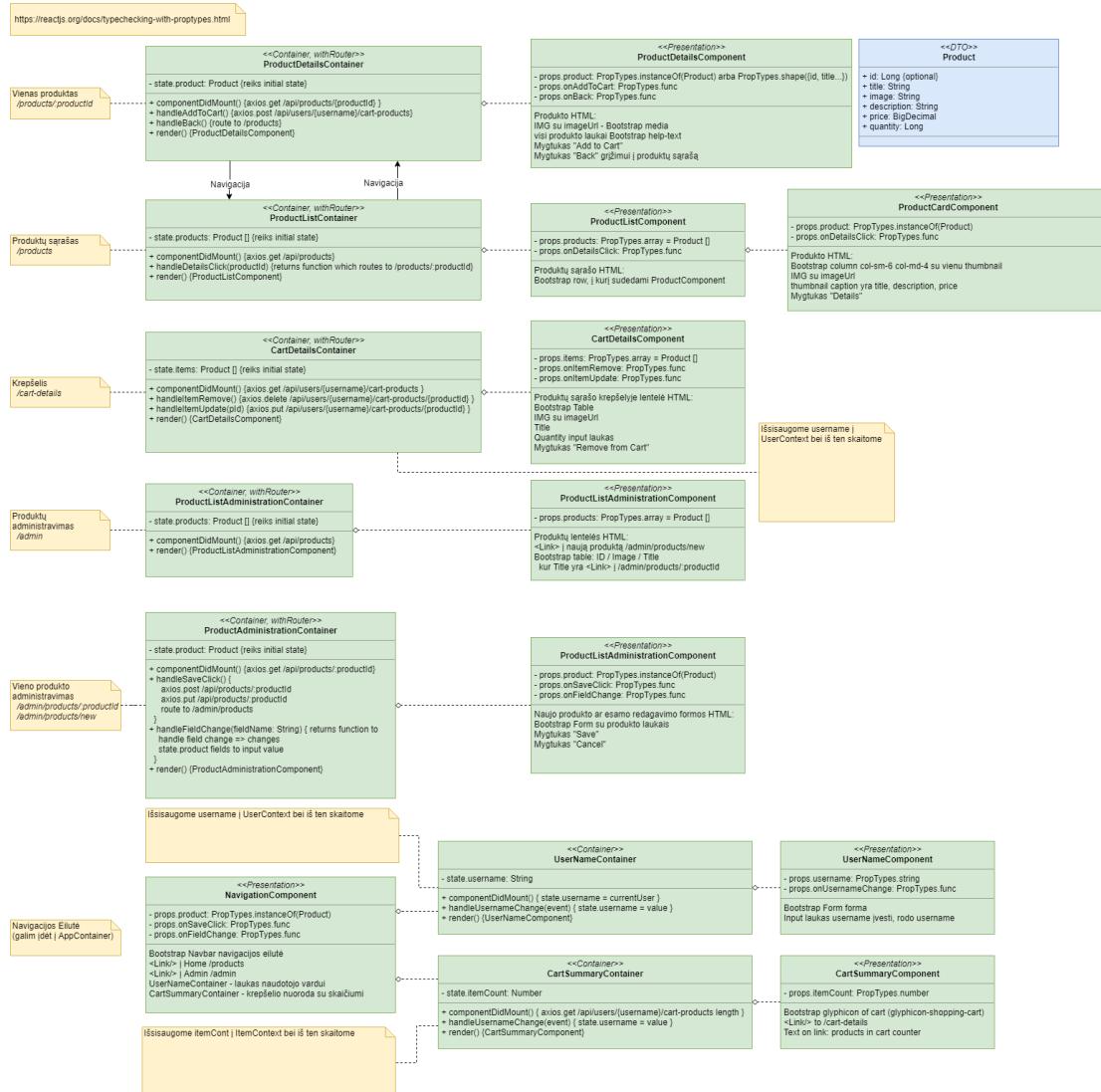
# PLANAS - SPRING JAVA



# PLANAS - SPRING REACT



# PLANAS - SPRING REACT



## PLANAS

- pasipaišyti klasių kvadratukus su pavadinimais
  - greičiausiai ant lapo
  - Backend (Rest/Spring) ir Frontend (UI/React) atskirai
    - ProductsController ir NewProductCommand,
    - ProductsDao ir ProductDto ..
    - ProductsContainer ir ProductsComponent,
    - OneProductContainer ir OneProductComponent ..
- vienas iš plano vykdymo variantų - iš viršaus į apačią



# PLANO VYKDYMAS



## PLANO VYKDYMAS - KLASIŲ KŪRIMAS IR SPRING

- sukurti VISUS tuščius failus (=klasė) projekte
  - norime panašius darbus atlikti kartu - optimizuotai
  - na, galime UI ir server-side atskirai
- pirma REST. Kiekvienam iš failų galime susikurti arba paprastą galutinę klasę (pvz. DTO atveju, nes jau turėsime pasipiešę), arba klasės griaučius
  - prisiminkime: sukūrėme UserController klasę, pažymėjome @RestController, sukūrėme metodus su @RequestMapping, tačiau metoduose tik println
    - vėliau kūrėme userDao, sugrįžome į UserController ir metodų viduje atlikome kvietimus į userDao servisa



## PLANO VYKDYMAS - REACT KOMPONENTAI

- jei visas projektas jau sukonfigūruotas, tai po klasių kūrimo per gana trumpą laiką mes turėsime pasirašę visus servisus ir veikiančią aplikaciją. Liks tik pabaigti
- toliau galime pereiti prie React'o. Taip pat susikuriame visus katalogus ir tuščius JS failus (=klasės/moduliai)
  - vėlgi, viduje susikuriame klasses (extends Component) jei tai container'is su tuščiu render() metodu, arba jei tai presentation html kodas, tai kintamuosius-arrow funkcijas, kurie tiesiog grąžina <div/> žymę; ir komponentus eksportuojame



## PLANO VYKDYMAS - URL NAVIGACIJA

- jei jau turėsime savo projektą, tai būsime iš anksto index.js susikonfigūravę Router navigacijai
  - tada galima susikurti visus Route su visais URI, pvz. /, /products ir pan., kuriuos projektavimo metu būsime jau apgalvojė, ir index.js faile importuoti ir priskirti skirtinges komponentus container'ius prie kiekvieno Route kelio
- taigi per gana trumpą laiką turėsime aplikacijos griaučius ir galėsime naviguoti bent jau įrašę URL į naršyklę
  - vėliau iš skirtingu vietų tereiks atlikti history.push



## PLANO VYKDYMAS - PRODUCTS UI

- jei gerai ir greitai sekasi, galime tokiu principu pabaigti ir Cart krepšelio servisus, juo labiau kad į duomenų bazę krepšelio nesaugosime
- grįžtame prie React, naviguojame naršyklėje į /products
  - susiimportuojame į ProductsContainer komponentą ProductsComponent, įdedame į render()
  - ProductsComponent sukuriame produktų tinklalį (pvz. Bootstrap row), importuojame OneProductComponent
  - OneProductComponent jau turi atvaizduoti produkta, tai sukuriame Produkto html (pvz. Bootstrap columns)



## PLANO VYKDYMAS - AXIOS

- kadangi Rest jau grąžina produktų sąrašą, einame į ProductsContainer
  - susikuriame state su produktais
  - sukuriame componentDidMount arrow funkciją
  - kviečiame per axios savo Rest servisą, produktus saugome į state
  - render() metode perduodame per atributą produktus į ProductsComponent



## PLANO VYKDYMAS - PRODUKTO VAIZDAVIMAS

- ProductsComponent produktus map'inam į ProductComponent ir perduodam jam jau tik vieną produkta
- gavę produkta į ProductComponent, pasiimam jį iš props ir iš props.product susidedame title ir kitus atributus į atitinkamas vietas savo HTML
- taigi jau turime produkų Rest servisą, per axios nusiskaitome ir naršykėje jau matome tikrus produktus
- analogiškai vis grįztame prie Spring Rest, pabaigiamo kažkurį servisą, tuomet vėl prie React



## PLANO VYKDYMAS - NAVIGACIJOS UŽBAIGIMAS

- Galime programuodami komponentą iškart atlikti ir navigaciją, arba tai pasilikti vėliau
- taigi einame į kiekvieną komponentą ir kuriame mygtuką ar nuorodą navigacijai kartu su callback metodu-arrow funkcija, kuri gauna event ir gali naviuoti
- pvz. NavigationComponent turėsime navigacijos juostą, kurios pagalba galima naviuoti į Products/Home, į ProductsAdministration, į Cart ir pan.
  - turint NavigationComponent nebereiks vesti URL, galesime spaudytį nuorodas ir eiti per komponentus



## PLANO VYKDYMAS - DB PRIJUNGIMAS

- čia jau turime pasirinkti, ar vis tik norime pabaigtis visus Spring servisus ir visus React komponentus ir tik tada prijungti DB, ar norime tai padaryti anksčiau
- jei aplikacijoje turėsime susikonfigūravę DB, tai tereiks kurti Entity lenteles-klases ir DAO servisus su tomis klasėmis dirbtis
- jei buvome pasidare InMemoryProductsDao, tai galėsime suteikti Qualifier naujam ProductsDatabaseDao ir kai darysim Autowire nurodysim tą Qualifier, taip pakeisdami savo ProductsController servise iš memory į DB dao



## PLANO VYKDYMAS - UŽBAIGIMAS

- tokiu principu pabaigiamo tiek Spring Rest servisus, tiek Dao ir kitus servisus, taip pat React komponentus
- tada dar kartą grjžtame ir skaitome užduotį - gal ką praleidam ar ne taip padarėme, gal yra papildomų techninių reikalavimų, gal reikia atlikti produktų filtravimą pagal kriterijus, o galbūt tiesiog yra papildomų užduočių
- po kiekvieno sėkmingo servisu ar komponentu sukūrimo tikslingo atlikti git add . ir git commit, o gal ir net ir git push, juolab kad nereiks mergint bent jau kontrolinio metu
  - taip git'e turėsime veikiančias dalis

## REIKALAVIMŲ SKAITYMAS

- dar kartą perskaitome visus reikalavimus
- pasitikriname, ar tikrai aplikacija padaryta pagal reikalavimus
- taisome, kas ne pagal reikalavimus
- vėl git add/commit/push



# KITOJE PASKAITOJE

JPA. Spring Boot prijungimas prie DB



## IŠ PRAEITOS PASKAITOS - LOMBOK

- @Data anotacija leidžia nerašyti get/set metodų
- Norint, kad neraudonuotų sakiniai IDE, reikia
  - atsisiu̇sti [lombok.jar](#) failą
  - uždaryti Eclipse ir paleisti

```
$ java -jar lombok.jar
```

- nurodyti, kur yra eclipse, ir instaliuoti
  - pvz. ~/eclipse/jee-2018-09/eclipse/
- paleisti Eclipse ir atnaujinti projektus refresh/clean
- Idea turētų padėti [Lombok plugin'as](#)





**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

# **ORM. JPA. SPRING DATA/DB PRIJUNGIMAS PRIE SPRING BOOT**

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# TURINYS

- Technologijos
  - nuo JDBC iki Hibernate
- ORM
- JPA
- EntityManager
  - Transakcijos
- Servisų kūrimas
- Spring Data
  - DAO
- JPQL užklausų kalba



# **TECHNOLOGIJOS. NUO JDBC IKI HIBERNATE**



**AKADEMIJA.LT**  
INFOBALT IR TECH CITY

# JDBC 1.X 1997 JAVA 1

## • Java Database Connectivity

```
CREATE TABLE "employee"
  (
    "CUST_ID" INT(10) unsigned NOT NULL AUTO_INCREMENT,
    "NAME" varchar(100) NOT NULL,
    "AGE" int(10) NOT NULL,
    PRIMARY KEY ("CUST_ID"),
    ENGINE=AnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

import java.sql.*;
public class EmployeeDAO implements EmployeeDAO{
    public void save(int empNo, String name, double sal, int doj) throws Exception{
        //Here we want to Insert the given employee details to the DB
        //So we will use JDBC API to connect to DB
        String sql = "insert into emp values('"+empNo+"','"+name."','".$sal."','".$doj+"')";
        Connection con = null;
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","system","manager");
            Statement st = con.createStatement();
            st.executeUpdate(sql);
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(con!=null)
                    con.close();
            }catch(Exception e){}
        }
    }
    public boolean updateSal(int empNo, double newSal) throws Exception{
        String sql = "update emp set sal='"+newSal+"' where empno='"+empNo';
        Connection con = null;
        boolean flag = false;
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","system","manager");
            Statement st = con.createStatement();
            if(st.executeUpdate(sql)) {
                flag = true;
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(con!=null)
                    con.close();
            }catch(Exception e){}
        }
        return flag;
    }
    public String getEmp(int empNo) throws Exception{
        String sql = "select * from emp where empno='"+empNo;
        StringBuilder sb = new StringBuilder();
        Connection con = null;
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","system","manager");
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery(sql);
            if(rs.next()){
                sb.append(rs.getString(1));
                sb.append(rs.getString(2));
                sb.append(rs.getDouble(3));
                sb.append(rs.getInt(4));
                sb.append(rs.getString(5));
                sb.append(rs.getDouble(6));
                sb.append(rs.getInt(7));
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(con!=null)
                    con.close();
            }catch(Exception e){}
        }
        return sb.toString();
    }
    public Boolean delete(int empNo) throws Exception{
        String sql = "Delete from emp where empno='"+empNo;
        Connection con = null;
        boolean flag = false;
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","system","manager");
            Statement st = con.createStatement();
            if(st.executeUpdate(sql)) {
                flag = true;
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(con!=null)
                    con.close();
            }catch(Exception e){}
        }
        return flag;
    }
}
```



# JDBC 2.X/DATASOURCE/DRIVERMANAGER 1998 JAVA 2

- DB drivers, parameters validation

```
CREATE TABLE `employees` {
    `CUST_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `NAME` varchar(100) NOT NULL,
    `AGE` int(10) unsigned NOT NULL,
    PRIMARY KEY (`CUST_ID`),
    ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
}
package com.mkyong.customer.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;
import javax.sql.DataSource;
import com.mkyong.customer.dao.CustomerDAO;
import com.mkyong.customer.model.Customer;

public class JdbcCustomerDAO implements CustomerDAO {
    private DataSource dataSource;
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }
    public void insert(Customer customer) {
        String sql = "INSERT INTO CUSTOMER " +
                    "(CUST_ID, NAME, AGE) VALUES (?, ?, ?)";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, customer.getCustId());
            ps.setString(2, customer.getName());
            ps.setInt(3, customer.getAge());
            ps.executeUpdate();
            ps.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {}
            }
        }
    }
    public Customer findByCustomerId(int custId) {
        String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, custId);
            Customer customer = null;
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                customer = new Customer(
                    rs.getInt("CUST_ID"),
                    rs.getString("NAME"),
                    rs.getInt("Age")
                );
            }
            rs.close();
            ps.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {}
            }
        }
    }
}
```



# HIBERNATE 2001

- Cross-DB SQL clauses/types, persistence of POJOs

```

    if (is_prime(n)) {
        for (int i = 2; i < n; i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }
    else {
        return false;
    }
}
else {
    return false;
}
}

int main() {
    cout << "Enter a number: ";
    int n;
    cin >> n;

    if (is_prime(n)) {
        cout << n << " is prime." << endl;
    }
    else {
        cout << n << " is not prime." << endl;
    }

    return 0;
}

```

## JAVA PASAULIS PATOBULĖJO

- EJB 2 -> EJB 3 entities
  - JPA 2006
- Spring 2001
- Hibernate įgyvendino JPA specifikaciją



# HIBERNATE/SPRING 2005/2011

- dar neseniai akademijoje

# HIBERNATE/SPRING 2005/2011

- daugiau konfigūracijos, bet daugiau ir automatizacijos, o  
ir mažiau kodo
- kelių CRUD metodų+paieškos pavyzdys (CRD be update):

```
@Repository
public class DBUserDAO implements UserDao {
    @PersistenceContext private EntityManager entityManager;
    public List<User> getUsers() {
        return entityManager.createQuery("SELECT u from User u", User.class).getResultList();
    }
    public void createUser(User user) { entityManager.persist(user); }
    public void deleteUser(String username) {
        User user = entityManager
            .createQuery("SELECT u from User u where username = :un", User.class)
            .setParameter("un", username).getSingleResult();
        if (entityManager.contains(user)) { entityManager.remove(user); }
        else { entityManager.remove(entityManager.merge(user)); }
    }
}
```



# HIBERNATE/SPRING DATA 1.0-1.09 2011/2016

```
<?xml version="1.0" encoding="UTF-8"?>
<project ...>
  ...
  <!-- use whatever versions make sense for your project. ... -->
  <properties>
    hibernate.version=4.1.1.Final
    spring-data.version=1.0.9.RELEASE
    spring-data-jpa.version=1.0.9.RELEASE
    spring-data-mongodb.version=1.0.9.RELEASE
  </properties>

  <dependencies>
    ...
    <dependency>
      standard Spring dependencies (beans, context, core, etc) ...
    </dependency>
    <dependency>
      org.springframework.data.mongodb:*
```

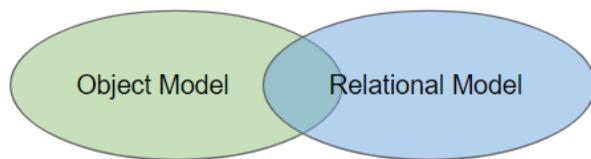


# **2017-2018. HIBERNATE. SPRING BOOT. SPRING DATA 1.11+,2.0+**

# ORM

# OBJEKTINIO-RELIACINIO MODELIO SUSIEJIMAS (ORM)

- Dauguma programų sąveikauja su RDBVS
- Dauguma DBVS gali dirbti tik su primityviais tipais – tekstiniais laukais, datomis, skaičiais.
- RDBVS neoperuoja objektais ar sudėtiniais tipais.
- Programos autorius sprendžia kaip lentelėje esančius duomenis perkelti į objektų grafą.
- Technika, padedanti sujungti šiuos modelius (objektinį ir reliacinių) vadina object-relation mapping (ORM).



# OBJEKTINIO-RELIACINIO MODELIO SUSIEJIMAS (ORM)

- Impedance mismatch: iššūkio esmė susieti modelius, nors tam tikros koncepcijos egzistuojančios viename modelyje neturi atitinkmens kitame modelyje. Pvz. Asmuo <-> Darbovietė susiejimas.
  - Polimorfizmas
  - Identitetas (equals(), ==, = (SQL))
  - Ryšiai (Java nuorodos ir išoriniai raktai).
  - Enkapsuliacija
  - Navigavimas ir skaitymas (nuorodos vs SQL rezultatų sąrašai)



# OBJEKTINIO-RELIACINIO MODELIO SUSIEJIMAS (ORM)

- Duomenų saugojimo būdai RDBVS
  - Išreikštinai „gauti“ reikšmes iš objektų ir išreikštinai išsaugoti naudoti SQL INSERT sakinius.
    - Tam naudojama žemesnio lygio specifikacija JDBC.
  - Naudoti aukštesnio abstrakcijos lygio susiejimą siejant Klases iš objektinio programavimo su Lentelėmis iš RDBVS. Toks susiejimas ir vadinas ObjectRelational Mapping (ORM).
    - Tam naudojama aukštesnio lygio specifikacija JPA.



## JDBC

- JDBC atsirado pats pirmas, kaip būdas saugoti duomenis DB.
- JDBC pateikia standartizuotą abstrakciją, kaip bendrauti su įvairių gamintojų duomenų bazių sąsajomis.
  - JDBC kodas yra portabilus, tačiau **SQL kalba – nėra portabili**
  - Realybėje praktiškai nepavyks parašyti SQL kodo, kuris nepakeistas veiktu ant bet kurių dviejų populiarųjų DB platformų



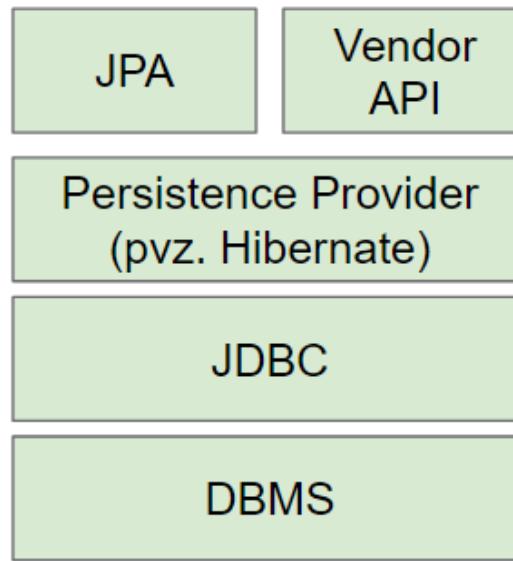
# JPA

## JDBC VS JPA

- JPA (Java Persistence Application programming interface arba API) suteikia priemones paprastų Java objektų saugojimui duomenų bazių valdymo sistemose.
- JPA gali būti naudojamas Java SE ir EE aplinkose
- JPA gali automatiškai susieti Java objektus ir duomenų bazių lentelių (ir kitų elementų) įrašus (eilutes).
- JPA yra aukštesnio lygio abstrakcija, nei JDBC.
- Java kodas gali būti\* izoliuotas nuo DBVS specifikos.
- Naujausia JPA standarto versija - 2.2 (hibernate - 2.1)



# JPA ARCHITEKTŪRA



- Persistence provider rūpinasi ORM
- JDBC suteikia metodus užklausoms ir duomenų atnaujinimui duomenų bazėje

# JPA SAVYBĖS

- Pakeičiama realizacija, galima naudoti skirtingų gamintojų produktus.
  - JPA – tai specifikacija. JPA realizacija – konkretus produktas, atitinkantis specifikaciją
  - JPA 2.1 Hibernate EclipseLink DataNucleus(2.2)
- Dinaminės, Type-safe užklausos.
- JPQL – giminininga SQL kalbai užklausų kalba, leidžianti daug paprasčiau operuoti objektais ir jų atributais, o ne duomenų bazės lentelėmis ir stulpeliais.
  - `SELECT c FROM Category c WHERE c.items is NOT EMPTY`



# JPA SAVYBĖS

- Gali sugeneruoti DB schemų kūrimo skriptus, juos įvykdyti. Gali perkurti visą DB.
- POJO (Plain Old Java Object) Saugojimas:

```
@Entity
public class Customer implements Serializable {
    @Id protected Long id;
    protected String name;
    public Customer() {}
    public Long getId() {return id;}
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
}
```



## ESYBĖS (ENTITIES)

- Reliacinėje paradigmoje esybė (Entity) dažniausiai yra atpažystamas ir identifikuojamas realaus pasaulio objekto atitikmuo, turintis atributus bei sąryšius su kitomis esybėmis. Pvz. Darbdavys, darbuotojas, darbovietė, prekė...
- Objektinio-reliacinio susiejimo pagrindas:
  - Entity klasės atitinka lenteles
  - Entity objektais atitinka įrašus lentelėse
  - Susiejimo detalių kontroliuojamos Java anotacijomis



# ENTITY PAVYZDYS

```
@Entity  
@Table(name = "customer")  
public class Customer {  
    @Id  
    public int id;  
    ...  
    public String name;  
    @Column(name="CREDIT")  
    public int c_rating;  
    @LOB public Image photo;  
    ...  
}
```



## REIKALAVIMAI ENTITY KLASĒMS

- Entity klasēs privalo atitikti reikalavimus:
  - Entity klasē privalo turėti bent vieną public arba protected konstruktorių be parametru
  - Entity klasē negali būti pažymėta final
  - Entity savybės ar metodai negali būti pažymėti final
  - Klasēs kintamieji privalo būti private/protected/package-private pasiekiamumo ir kreipimasis į juos galimas tik per klasēs metodus



# REIKALAVIMAI ENTITY KLASĒMS

- Entity klasės laukai turi būti šių tipų:
  - Primityvūs
  - java.lang.String
  - Serializuojami tipai
    - Wrappers of Java primitive types
    - java.math.BigInteger, java.math.BigDecimal
    - java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp
    - User-defined serializable types
  - byte[], Byte[], char[], Character[]
  - Enumeruojami tipai
  - Kiti Entities tipų laukai arba jų kolekcijos
  - @Embeddable pažymėtos klasės



## REIKALAVIMAI ENTITY KLASĖMS

- Laukai, pažymėti javax.persistence.Transient arba Java raktažodžiu transient NEbus saugomi į db.
- Entity klasės turi laikytis JavaBeans pavadinimų suteikimo taisyklių (setProperty, getProperty, isProperty).
- Naudojamos kolekcijos turi būti vienos iš:
  - java.util.Collection
  - java.util.Set
  - java.util.List
  - java.util.Map



## TABLE

- Lentelei duomenų bazėje galime suteikti kitą pavadinimą

```
@Entity  
@Table(name = "klientas")  
public class Customer {
```

- Visur java ir JPA naudosime Customer, tačiau duomenų bazėje lentelė bus saugoma kaip klientas



## COLUMN

- Lentelės stulpeliai žymimi @Column anotacija

```
@Column // nebūtina anotacija  
public int c_rating;  
@Column(name="DB_column", nullable=false, length=5)  
private String someProperty;
```

- Kaip ir Table atveju, pavadinimą tikroje duomenų bazėje galime pakeisti
- someProperty duomenų bazėje saugomas stulpelyje DB\_column, turi ribojimą NOT NULL ir yra VARCHAR(5) tipo.



## COLUMN

- Lentelės stulpeliai žymimi @Column/@Temporal anotacija

```
@Column(precision=4, scale=1)
private Double someNumber;
@Temporal(TemporalType.DATE)
private Date someDate;
@Temporal(TemporalType.TIMESTAMP)
private Date someOtherDate;
```

- DECIMAL(4, 1), max skaičius: 999.9
- someDate saugomas DATE tipo stulpelyje
- someOtherDate saugomas TIMESTAMP tipo stulpelyje



## ENUMERATED

- Lentelės stulpeliai žymimi @Enumerated anotacija

```
public enum SalaryRate { JUNIOR, SENIOR, MANAGER, EXECUTIVE }  
..  
@Enumerated(STRING)  
private SalaryRate salaryRate;
```

- Nėra tipo atitikmens duomenų bazėse, todėl anotacija leidžia parinkti duomenų bazės atitikmenis
  - raidinę (string)
  - enumeratoriaus eilės numerio (ordinal)



## TRANSIENT

- Lentelėje nesaugomi duomenys žymimi @Transient anotacija

```
@Transient  
private int myCounter;
```

- myCounter nebus saugomas į DB ir nebus valdomas



# ENTITYMANAGER



## ENTITYMANAGER

- Entitymanager klasė yra centrinis entity klasių valdymo taškas.
- Persistence context – registras kur saugoma entity objektų būsenos.
- Jis valdo susiejimą tarp iš anksto žinomų entity klasių ir ORM duomenų šaltinio.
  - Suteikia API užklausoms, objektų paieškai, sinchronizacijai, duomenų saugojimui DB.



## ENTITYMANAGER

- `find(Class, Object)` - surasti objektą pagal pirminį raktą
- `persist(Object)` - išsaugoti ir pradėti valdyti objektą
- `merge(Object)` - išsaugoti objekto pakeitimus
- `refresh(Object)` - gauti objekto pasikeitimus iš DB
- `remove(Object)` - pašalinti objektą
- `createQuery(String)` - sukurti JPQL užklausą
- Visos operacijos gali būti taikomos ir pilnam objektų grafui

## EM.PERSIST

```
Customer customer = new Customer(1, "John Bow");  
em.persist(customer);
```

- Entitymanager'io klasės persist() metodas užregistruoja entity į persistence context.
- Nuo šio momento entity yra valdomas ir Entitymanger užtikrina, kad šio objekto duomenys būtų sinchronizuoti su duomenų baze



## EM.FIND

```
Customer customer = em.find(Customer.class, id);
```

- Entitymanager find() metodas leidžia pagal entity klasę ir pirminio rakto identifikatorių surasti entity duomenų bazėje.
- Jeigu operacija sėkmingai pasibaigia, grąžintas Customer objektas taps valdomas/prižiūrimas.
  - Tačiau, jei objektas nebus surastas - find() grąžins null



## EM.REMOVE

```
Customer customer = em.remove(customer);
```

- Entitymanager remove() metodas pašalina duomenis susijusius su šiuo objektu iš duomenų bazės.
- remove() metodas pašalins objektą iš persistence context.  
T.y. jo būsena nebebus sekama



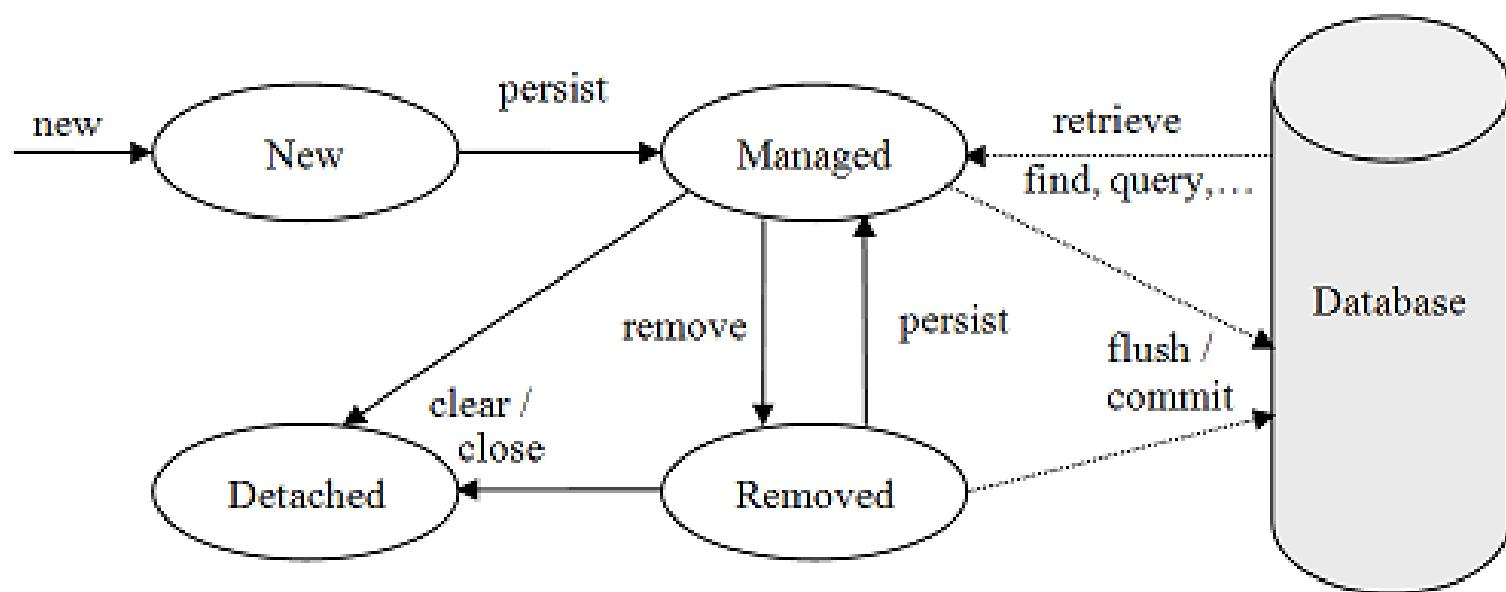
## EM.CREATEQUERY

```
Query q = entityManager.createQuery("select g from Good g");  
List<Good> goods = q.getResultList();
```

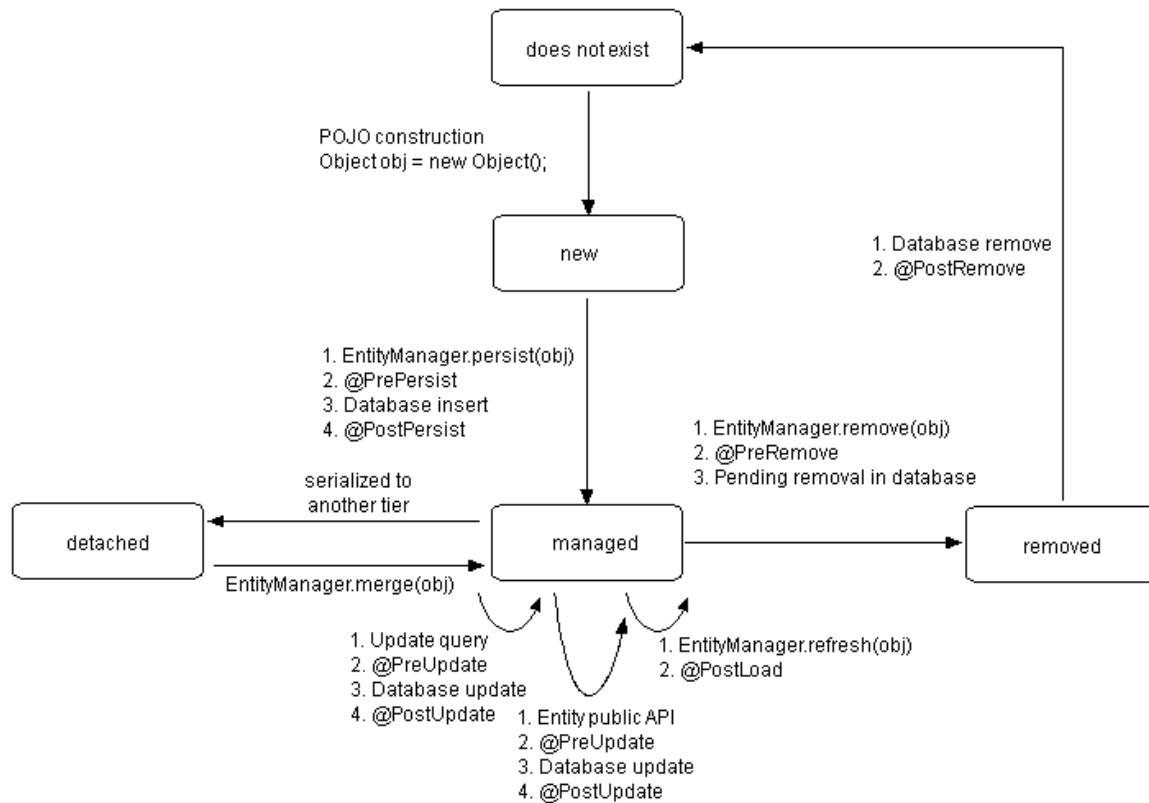
- `createQuery()` konvertuos JPQL užklausą į SQL užklausą ir ją įvykdys
- SQL užklausos rezultatą konvertuos į objektus



# ENTITY LIFECYCLE



# ENTITY LIFECYCLE



## ENTITY IDENTIFIKATORIAUS SUTEIKIMAS (GENERAVIMAS)

- Identifikatorius Entity objektams gali būti sugeneruojamas JPA realizacijos (persistence provider) automatiškai, pagal nurodytą tipą.
- Tam naudojama @GeneratedValue anotacija
- Programuotojas gali pasirinkti vieną iš kelių strategijų:
  - AUTO - provider chooses for us
  - TABLE - naudoja generator table
  - SEQUENCE - DB feature
  - IDENTITY - DB feature



# ENTITY IDENTIFIKATORIAUS SUTEIKIMAS (GENERAVIMAS)

```
@Entity  
public class User {  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
  
    @Column  
    private String name;  
  
    // geteriai ir seteriai  
}
```

- Auto naudoti reikėtų tik kuriant, realiems produktams - nurodyti konkretų tipą

# TRANSAKCIJOS (OPERACIJOS)

## TRANSAKCIJOS (OPERACIJOS)

- Metodai, kurių veiksmai vykdomi transakcijose pažymimi @Transactional
- Transakcija yra būtina visiems metodams dirbantiems su duomenų baze
- @Transactional pradės transakciją tik jei komponentas sukurtas per Spring ir kviečiamas iš kitos klasės



## TRANSAKCIJŲ VALDYMAS

- JTA ir Resource Local tipai:
  - JTA - transakcijos valdomos per JTA API. Transakcijas valdo išorinis komponentas. Palaikoma Java EE aplinkoje. Transakcijos gali būti pradėtos ir baigtos išoriniuose (mūsų programuojamiems) komponentams.
  - Resource Local – transakcijos valdomos EntityManager pagalba. Palaikoma tiek Java EE, tiek Java SE aplinkose.



# TRANSAKCIJŲ VALDYMAS

```
@Transactional  
public void updateGood(Integer goodId) {  
    Good good = entityManager.find(Good.class, goodId);  
    good.setName("Test");  
}
```

- Pakeitimai bus išsaugoti automatiškai pasibaigus transakcijai - nereikės kvesti em.persist()



# TRANSAKCIJŲ VALDYMAS

```
@Transactional  
public void updateGood(Good good) {  
    good.setName("Test");  
}
```

- Pakeitimas automatiškai į duomenų bazę išsaugotas nebus, nes good greičiausiai buvo užkrautas kitoje transakcijoje ir todėl nepriklauso esamai sesijai
  - čia gautas Good yra detached



# TRANSAKCIJŲ VALDYMAS

```
@Transactional  
public void updateGood(Good good) {  
    good = entityManager.merge(good);  
    good.setName("Test");  
}
```

- Pakeitimai bus išsaugoti, nes po merge(...) operacijos grąžintas good jau yra šios sesijos dalis



# TRANSAKCIJŲ VALDYMAS

```
@Transactional  
public void updateGood(Good good) {  
    good.setName("Test");  
    Good savedGood = goodRepository.save(good);  
}
```

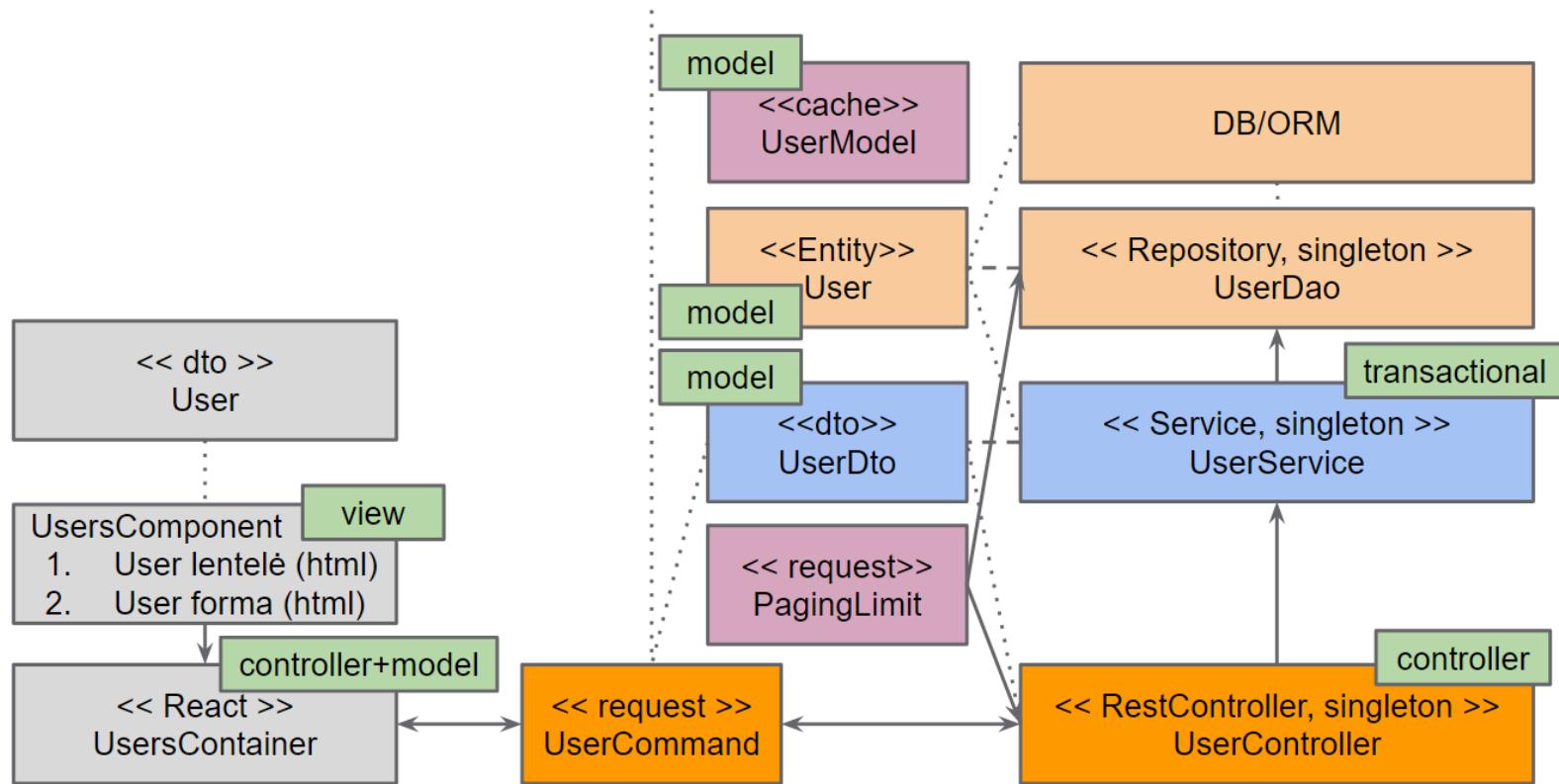
- naudojant Spring Data save() metodą, merge būtų iškviestas automatiškai
- čia good yra detached, o savedGood - jau attached (valdomas)



# SERVISŲ KŪRIMAS



# MVC - TRANSACTIONAL



## SERVISŲ SLUOKSNIS

- SoC - separation of concerns, dizaino principas
  - TCP turi 7 sluoksnius..
  - .. arba HTML/CSS/JS, etc.
- Pagal REST standartą entity objektai neturi būti perduoti į per Rest servisą atgal klientui
  - klientas dirba su Rest resursais. Resursas != Esybė
  - Esybės ID, vidiniai/transient parametrai ne resurso dalis. Pvz. jei turime 2 servisus ir jie dirba su skirtina DB, kiekvienoje DB jų ID gali būti vienodas



## SERVISŲ SLUOKSNIS

- serviso užduotis - atlikti biznio transakciją
  - pridėti User, priskirti user'iui rolę
- jei ji nepavyko, tai užduotis: atlikti rollback
- RestController gali kvesti keletą verslo transakcijų
- RestController užduotis - priimti užklausas ir pateikti atsakymus
- Repository/DAO užduotis - atlikti atomines operacijas su esybėmis



## SERVISŲ SLUOKSNIS

- norint atlikti skirtingas validacijas
  - užklausos duomenims
  - servisui reikalingiems duomenims
  - DB reikalingiems duomenims
- pvz. gal mes priimam ir 254 simbolių ilgio vardą, bet servisas dirba su simbolių eilute, kurią turi užkoduoti, tai jam 200 gali būti maksimumas
  - o DB pvz. jei prijungiam H2 - leidžiam 254, o jei prijungiam seną Mainframe vietoj DB - tai tik 50



## SERVISŲ SLUOKSNIS

- darbui su DB reikalinga transakcija, bet RestController atitinkmuo nebūtinai kviečiamas iš Spring. Kitaip sukonfigūravus aplikaciją, tai gali būti tiesiog klasė, todėl joje transkacijos su @Transactional neveiks, nes jos veikia tik kai yra kviečiamos iš Spring
- beje, ne visada naudojama ir Jackson biblioteka JSON kodavimui ar atkodavimui
  - pvz JsonIgnore su kitomis bibliotekomis neveiks, tai kitur ID nebus galima tiesiog ignoruoti



## UŽDUOTIS #1 - DAO SU EM

- pirmiausia reikia sukonfigūruoti Spring Boot aplikaciją
- konfigūracijos tikslas: turėti H2 duomenų bazę, veikiančią su Hibernate, kurį naudoja Spring Data, kad viskas veiktu Spring Boot aplikacijoje, o duomenų bazę mums sukurtų automatiškai mūsų nurodytoje vietoje ir išjungiant aplikaciją duomenys išliktų
- į pom.xml prisidėti priklausomybę

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```



## UŽDUOTIS #1 - DAO SU EM

- į application.properties failą įsidėti DB nustatymus:

```
###  
# Database Settings  
###  
spring.datasource.url=jdbc:h2:file:///tmp/test2235.db;AUTO_SERVER=TRUE;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE  
# ;AUTO_SERVER=TRUE;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE  
spring.datasource.platform=h2  
spring.datasource.username = sa  
spring.datasource.password =  
spring.datasource.driverClassName = org.h2.Driver  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

- DB bus sukurta /tmp/test2235.db, todėl atnaujinant DB struktūrą reiks ištrinti šiuos failus



## UŽDUOTIS #1 - DAO SU EM

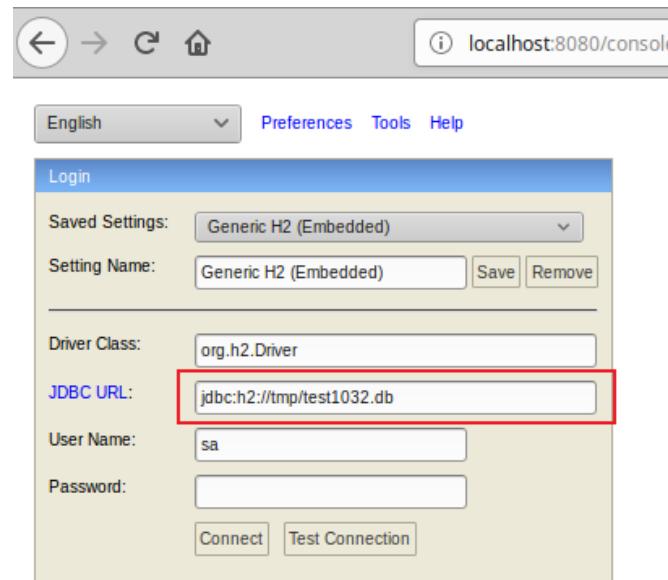
- į application.properties failą įsidėti H2 DB nustatymus:

```
###  
#   H2 Settings  
###  
spring.h2.console.enabled=true  
spring.h2.console.path=/console  
spring.h2.console.settings.trace=false  
spring.h2.console.settings.web-allow-others=false
```

- prie konsolės prieiti galima <http://localhost:8081/console>



# UŽDUOTIS #1 - DAO SU EM



- naudodamiesi H2 konsole, nepamirškite įsivesti savo DB kelią, kurį nurodėte application.properties faile

```
spring.datasource.url=jdbc:h2:file://tmp/test2235.db;AUTO_<..>
```

## UŽDUOTIS #1 - DAO SU EM

- į application.properties failą įsidėti Hibernate nustatymus:

```
###  
#   Hibernate Settings  
###  
spring.jpa.hibernate.ddl-auto = update  
spring.jpa.properties.hibernate.show_sql=true  
spring.jpa.properties.hibernate.use_sql_comments=false  
spring.jpa.properties.hibernate.format_sql=true  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect  
spring.jpa.properties.hibernate.hbm2ddl.auto=update  
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=true  
spring.jpa.properties javax.persistence.validation.mode=auto
```



## UŽDUOTIS #1 - DAO SU EM

- User klasę pažymėti @Entity
  - atsargiai - mes naudojam paprastą JPA, todėl anotacijos taip pat iš javax.persistence.\* , pvz.:

```
import javax.persistence.Entity;
```

- User klasės atributus pažymėti @Column
- taip pat sukurti ir auto generated id
- pastaba: klasė negali buti final



## UŽDUOTIS #1 - DAO SU EM

- implementuoti DAO su EM - galime pasinaudoti prieš tai rodytu kodu:

```
@Repository
public class DBUserDAO implements UserDao {
    @PersistenceContext private EntityManager entityManager;
    public List<User> getUsers() {
        return entityManager.createQuery("SELECT u from User u", User.class).getResultList();
    }
    public void createUser(User user) { entityManager.persist(user); }
    public void deleteUser(String username) {
        User user = entityManager
            .createQuery("SELECT u from User u where username = :un", User.class)
            .setParameter("un", username).getSingleResult();
        if (entityManager.contains(user)) { entityManager.remove(user); }
        else { entityManager.remove(entityManager.merge(user)); }
    }
}
```



# UŽDUOTIS #1 - DAO SU EM

- sukurti UserService
- iš UserController kvesti tik UserService metodus
  - userDao iš UserController pašalinti
- iš UserService kvesti UserDao

```
@Service
public class UserService {
    @Autowired @Qualifier("repoUserDao")
    private UserDao userDao;

    @Transactional(readOnly = true)
    public List<User> getUsers() {
        return userDao.getUsers();
    }

    @Transactional
    public void createUser(User user) {
        userDao.createUser(user);
    }
    ...
}
```



## UŽDUOTIS #2 - REQUEST SCOPE

- susikurti request scope bean PagingData

```
@Component
@Scope(value = "request", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class PagingData {
    private int limit;
    public PagingData() {
        this.limit = 5; // <numatytais filtras>
    }
    // getters and setters
}
```

- UserController įsidėti kaip Autowired, o getUsers nustatyti pvz setLimit(10)
- UserDao įsidėti kaip Autowired ir pries getResultList pakvesti .setMaxResults() su getLimit()



# SPRING DATA. DAO



## SPRING DATA

- o kas jeigu nenorime rūpintis paprastomis užklausomis, juk jos visada vienodos
  - mes nenorime išradinėti dviračio
- Spring Data leidžia
  - atsisakyti paprastų užklausų
  - lengviau rašyti vidutinio sudėtingumo užklausas
  - visiškai nekurti repository klasės



# DAO SU SPRING DATA

- CRUD įgyvendinantis DAO su Spring Data atrodo taip:

```
public interface UserRepository extends JpaRepository<User, Long> { }
```

- nereikia implementacijos!!
- jei kitoje klasėje pasiimsime su Autowired, galėsime kvieсти įvairius metodus, pvz.:
  - userRepository.save(user); Create/Update
  - userRepository.findAll(); Read
  - userRepository.delete(User); Delete
    - ir kt.
- interfeisas: JpaRepository<Entity, ID>



## DAO SU SPRING DATA

- `JpaRepository` yra `CrudRepository` + `PagingAndSortingRepository`
  - `Paging/Sorting`: galima naudoti `Pageable` ir `Sort` parametrus
  - `Jpa`: persistence contexto detalesnis valdymas, įrašų įterpimas/trynimas/ieškojimas batch'u (po daug vienu metu)
    - pvz. susirasti įrašus pagal jų ID viena užklausa:

```
List<User> findAllById(Iterable<ID> ids)
```



## DAO SU SPRING DATA

- Tampa ypatingai svarbus servisų sluoksnis
  - įsitikinkite, kad kviečiate Dao ar Repository tik iš servisų sluoksnio, ne iš rest controller'io
    - kitu atveju tiesiog galite nesuprasti, kodėl vienas ar kitas dalykas neveikia
  - serviso sluoksnio metodai turi būti anotuoti @Transactional
    - tuomet teisingai veikia DB trasakcijų automatizavimas



## DAO SU SPRING DATA

- papildome metodu, kuris ieško User pagal tikslų email:

```
public interface UserRepository extends JpaRepository<User, Long> {  
    User findByEmail(String email);  
}
```



## DAO SU SPRING DATA

- papildome metodu, kuris ieško visų User, kurių email dalis yra partOfEmail:

```
public interface UserRepository extends JpaRepository<User, Long> {  
    List<User> findUsersByEmailContaining(String partOfEmail);  
}
```



## DAO SU SPRING DATA

- papildome metodu, kuris trina visus User pagal username:

```
public interface UserRepository extends JpaRepository<User, Long> {  
    void deleteByUsername(String username);  
}
```



# SPRING DATA TAISYKLĖS

- galima susikurti bet kokius metodus, sujungiant operacijos pavadinimą (pvz `delete`), tuomet tai, ką norime rasti, pvz `User`, žodj `by` bei reikalingus laukus
- galimos operacijos:
  - `find`, `read`, `query`, `count`, `get`, `delete`, `exists`
- neįrašius ieškomojį, bus pasirinkta iš interfeiso
- galima patikslinti, ko tiksliau ieškome:
  - `findFirst...`, `findTop...`, `findDistinct...`, `findUser...`,  
`findFirstUser...`, `findDistinctUser...`



# SPRING DATA TAISYKLĖS

- laukus galima jungti su And ir Or, pvz.:
  - `findUsersByUsernameAndEmail`,  
`findByUsernameOrEmail`
- po laukų galime nurodyti jų naudojimo taisykles, pvz.  
`Containing`, `IgnoreCase`
- taip pat galime nurodyti rūšiavimą `OrderByEmailAsc` ar  
`OrderByUsernameDesc`
- daugiau informacijos ir visi raktiniai žodžiai, kuriuos  
galime naudoti:
  - [Spring Data Query Creation](#)



# SPRING DATA TAISYKLĖS

- keletas pavyzdžių, ką galime parašyti ir kaip komponuoti:

```
List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);  
// Enables the distinct flag for the query  
List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);  
List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);  
// Enabling ignoring case for an individual property  
List<Person> findByLastnameIgnoreCase(String lastname);  
// Enabling ignoring case for all suitable properties  
List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);  
// Enabling static ORDER BY for a query  
List<Person> findByLastnameOrderByFirstnameAsc(String lastname);  
List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
```



# SPRING DATA TAISYKLĖS

- norėdami naudoti savo SQL, naudojame **@Query**:

```
@Query("select u from User u where u.emailAddress = ?1")
User findByEmailAddress(String emailAddress);
```

- norėdami modifikuoti duomenis, naudojame **@Modifying**:

```
@Modifying
@Query("update User u set u.firstname = ?1 where u.lastname = ?2")
int setFixedFirstnameFor(String firstname, String lastname);
```

- arba kuriame savo DAO ir tame naudojame EntityManager bei rašome savo Java kodą bei JPQL



## PAVYZDYS EXISTSBY

- Spring Data existsBy galima apsirašyti taip:

```
boolean existsByNickname(String nickname);
```

- Bet galima panaudoti ir Query anotaciją:

```
@Query(value =
    "SELECT CASE WHEN count(u)> 0 THEN true ELSE false END"
    +" FROM User u WHERE nickname = ?1")
boolean existsByNickname(String nickname);
```

- arba findBy metodą:

```
User findByNickName(String nickname);
default boolean existsByNickname(String nickname) {
    return findByNickName(nickname) != null;
}
```

## UŽDUOTIS #3 - SPRING DATA

- sukurkite Rest servisą ieškoti naudotojams pagal vardą ir pavardę
- paiešką suprogramuokite sukurdami Spring Data interfeiso metodą



# JPQL UŽKLAUSŲ KALBA



## JPQL

- The Java Persistence Query language (JPQL) tai nuo platformos nepriklausoma objektiškai orientuota užklausų kalba. Ji yra dalis JPA specifikacijos.
- Pagrindinis skirtumas nuo SQL yra:
  - JPQL operuoja esybėmis ir objektais (graifais)
  - SQL operuoja lentelėmis, stulpeliais ir įrašais
- JPQL palaiko trijų tipų sakinius: SELECT, UPDATE, DELETE



# JPQL

- JPQL panašus į SQL, bet tai ne tas pats SQL:

```
SELECT c FROM Customer c
```

- FROM sąlyga JPQL kalboje nurodo entity, kuris bus naudojamas užklausoje.
- Pavyzdyje Customer yra entity klasė, c – objekto identifikatorius.
  - Identifikatorius gali būti naudojamas toliau sąlygoje nurodant sąryšius ar savybes (where sąlygoje)



# JPQL

- Išraiškose galima naudoti taško sintaksę prieiti prie objekto laukų (path expressions):

```
SELECT c FROM Customer c where c.address IS NOT EMPTY
```

- Path expressions gali būti naudojama:
  - WHERE sąlygose
  - ORDER BY sąlygose
- JPQL galima naudoti JOIN'us, tačiau užtenka susieti su objektu, nereikia nurodyti pirminio/išorinio rakto.

```
SELECT r.loginName FROM Customer c JOIN c.credentials r  
WHERE c.address.street = 'Fast'  
SELECT r.loginName FROM Customer c, Credentials r  
WHERE c=r.customer AND c.address.street = 'Fast'
```



## QUERY API

- JPA technologija leidžia TRIMIS skirtingais būdais pasiekti duomenis:
  - EntityManager.find()
  - JPQL užklausomis
  - SQL užklausomis
- JPQL ir SQL užklausos vykdomos JPA Query API priemonėmis
- Naudojant hibernate, JPQL išplečia HQL



# QUERY API

- Query API palaiko dviejų rūšių užklausas. Abiem atvejais naudojamas Entity manager: Named Query ir Query
- Vardinės užklausos (Named queries) rašomos, kai užklausos kriterijai žinomi iš anksto
- Dinaminės užklausos konstruojamos ir vykdomos dinamiškai
- Užklausas, kurios naudojamos daugiau nei vienoje vietoje, geriau realizuoti kaip vardines.
- Jos pasižymi geresne greitaveika, nes yra suformuojamos vieną kartą ir vėliau pernaudojamos.
- Vardinės užklausos aprašomos entity klasėje naudojant anotaciją @NamedQuery ir vėliau naudojamos siejant Spring Data metodus arba em.createNamedQuery("uzklausosPavadinimas")



# QUERY API

- vardinė užklausa @Entity esybėje:

```
@Entity  
@NamedQuery(name = "User.findByEmailAddress",  
    query = "select u from User u where u.emailAddress = ?1")  
public class User { }
```

- ir Spring Data repository ją panaudos aprašius taip:

```
public interface UserRepository extends JpaRepository<User, Long> {  
    User findByEmailAddress(String emailAddress);  
}
```



## QUERY API

- Dinaminės užklausos kuriamos vykdymo metu ir yra naudojamos, kai pati užklausa priklauso nuo situacijos (pvz. kiek duomenų įvesta į paieškos laukelį)

```
Query query = em.createQuery("SELECT c FROM Customer c");
```

- Query klasės objektai atitinka vieną užklausą ir pateikia keletą metodų nurodyti užklausos parametrus
- Parametrus galima nurodyti dviem būdais:
  - Pagal parametrovardą
  - Pagal eilės numerį
- Spring Data šios užklausos rašomos naudojant @Query



# QUERY API

- parametrų pvz.

```
Query query=em.createQuery("SELECT i FROM Item i WHERE i.name = ?1");
query.setParameter(1, "Car");
List<Item> items = query.getResultList();

Query query = em.createQuery("SELECT i FROM Item i WHERE i.name =
:itemName");
query.setParameter("itemName", "Car");
List<Item> items = query.getResultList();

@Query("select u from User u where u.firstname like %?1")
List<User> findByFirstnameEndsWith(String firstname);

@Query("select u from User u where u.firstname like %:name")
List<User> findByNameEndsWith(@Param("name") String name);
```



## UŽDUOTIS #4 - JPQL

- User sukurkite age lauką
- sukurkite Rest servisą seniausiam naudotojui gauti
  - nepamirškite ir UserService
- parašykite dinaminę SQL užklausą, kuri išrenka seniausią naudotoją
- užklausą realizuokite repozitorijos interfeise pasinaudodami Spring Data @Query
- sukurkite default User findOldestUser() interfeiso metodą
  - implementacijoje tiesiog pakvieskite Spring Data metodą paieškai



## UŽDUOTIS #5 - PARDUOTUVĖS DAO

- sukurkite DAO tvarkytis su produktais
- įsivaizduokime, jog norėsime ir krepšelį saugoti duomenų bazėje
  - sukurkite kitą DAO bei krepšelio tarpinį servisą, skirtą dirbti su krepšeliu
- panaudokite abu repository DAO Rest servisuose
  - kol kas krepšelyje saugomi produktai bus atskiri krepšelio produktai, turintys tikrųjų produktų ID
  - .. vėliau išmoksime susieti kelis @Entity tarpusavyje



# KITOJE PASKAITOJE

JPA priklausomybės. Live coding. Užduotys



AKADEMIJA.IT  
INFOBALT IR TECH CITY

## IŠ PRAEITOS PASKAITOS

- JpaRepository API yra User [Entity]
  - tik aplinkiniai sluoksniai, DB (tiksliau Entity Manager šiuo atveju) arba servisas, gali keistis duomenimis su šiuo API
  - taigi čia galima būtų bendrauti su serviso DTO objektais, tačiau kadangi mes šio sluoksnio neprogramuojam, tai nebūtinai
- pvz. jis gali išsaugoti User su id, firstname, lastname, username



# IŠ PRAEITOS PASKAITOS

- Service API yra User[Service]
  - tik aplinkiniai sluoksniai, JpaRepository arba RestController, gali su juo keistis duomenimis
  - skirtas duomenis transformuoti ir perduoti iš DB arba atgal į DB
- jis gali sukurti User su daliniais duomenimis, pvz. gaudamas tik firstname+lastname
  - turi pasirūpinti ID generavimu (jei ne AUTO), ką rašyti į username laukelį, taip pat patikra ar User su tokiu firstname+lastname dar nėra, o jei yra ir firstname+lastname+username derinys unikalus, sugeneruoti kokį nors username



# IŠ PRAEITOS PASKAITOS

- RestController API yra User[Rest]
  - tik aplinkiniai sluoksniai, Rest naudotojas (React) arba Service, gali su juo keistis duomenimis
  - skirtas surinkti duomenis iš Rest naudotojo įvairiais formatais ir perduoti į Service apdorojimui, o taip pat gautus duomenis iš Service paversti Rest naudotojui suprantamu formatu ir grąžinti
- jis gali turėti pvz /create-user ir leisti užklausą User kūrimui paduodant tik {firstName, lastName}
  - šiuo atveju User[Rest] bus tik su firstName ir lastName



## IŠ PRAEITOS PASKAITOS

- Vėliau kai suprasite, kaip viskas veikia, galite sugalvoti, kaip sumažinti objekto skaičių, tačiau turite pasirūpinti, kad:
  - User[Entity] tiesiogiai niekada nenuėitų Rest klientui
    - ir dėl saugumo, ir dėl to, kad klientas neturi prisirišti pvz. prie duomenų bazės vidinių savybių
  - neprašyti kliento User[Entity] duomenų
    - pvz. vienoj duomenų bazėj User[Entity] ID bus Long, kitoj String. Jei reikia unikalai identifikuoti User, geriau tai daryti pvz. per username arba leisti UserService sugeneruoti ir išsaugoti nuo DB nepriklausantį unikalų ID





**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

**JPA SĄRYŠIAI TARP ESYBIŲ. KOLEKCIJŲ  
SAUGOJIMAS. PAVELDĖJIMAS. KASKADINĖS  
OPERACIJOS.**

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# TURINYS

- Sąryšiai tarp esybių
- Kolekcijų saugojimas
- Kaskadinės operacijos
- Duomenų tikrinimas - validacija
- Entity paveldėjimas

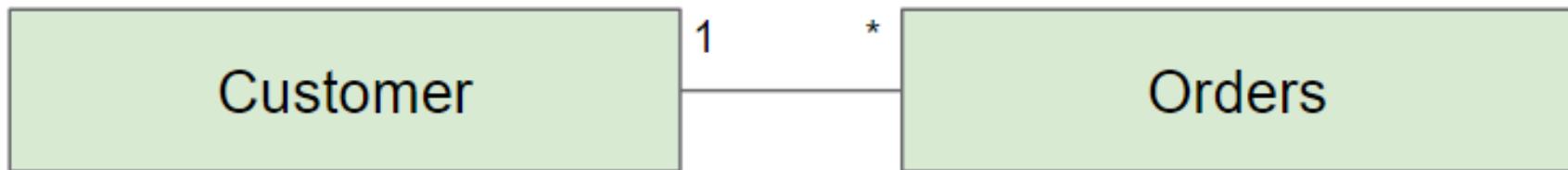


# SĄRYŠIAI TARP ESYBIŲ



AKADEMIJA.LT  
INFOBALT IR TECH CITY

# SĄRYŠIŲ TARP ESYBIŲ VALDYMAS



- Saryšiai gali būti:
  - One-to-one
  - one-to-many
  - many-to-many
  - many-to-one
- Vienpusiai ir abipusiai
- Palaikantys Collection, Set, List ir Map tipus



## SĄRYŠIAI: VIENAS-SU-VIENU

```
@Entity  
public class Customer {  
    @OneToOne  
    private Credentials credentials;  
}
```

- Turime nuorodą į vieną Credentials.



# SĄRYŠIAI: VIENAS-SU-VIENU

```
@Entity  
public class Customer {  
    @Id  
    public Long id;  
    ...  
    @OneToOne  
    public Credentials credentials;  
    ...  
}
```

```
@Entity  
public class Credentials {  
    @Id  
    public Long id;  
    public String username;  
    ...  
    @OneToOne(mappedBy="credentials")  
    public Customer customer;  
    ...  
}
```

- one-to-one sąryšis aprašomas `@OneToOne` anotacija
- Jei sąryšis dvipusis, ne savininko pusė turi naudoti `mappedBy` žymėjimą. Šiuo atveju `Customer` yra savininkas ir turės išorinį raktą į `Credentials`.



## SĄRYŠIAI: VIENAS-SU-VIENU

```
@Entity  
public class Customer {  
    @OneToOne  
    @JoinColumn(name="credentials_id")  
    private Credentials credentials;  
}
```

- Lentelėje Customer yra nuoroda: stulpelis credentials\_id su foreign key į Credentials



# SĄRYŠIAI: VIENAS-SU-DAUG

```
@Entity  
class Cart {  
    @Id  
    private Integer id;  
    @OneToMany  
    private List<Good> goods;  
}
```

```
@Entity  
class Good {  
    @Id  
    private Integer id;  
    @ManyToOne  
    private Cart cart;  
}
```

- many-to-one ryšys pažymimas @ManyToOne anotacija ir naudojamas esybėje, kurių bus "daug"
- one-to-many žymimas @OneToMany anotacija ir naudojamas klasėje, kurių bus tik "viena"
- Lentelėje Good yra stulpelis cart\_id su FK į Cart



# SĄRYŠIAI: VIENAS-SU-DAUG

```
@Entity  
class Cart {  
    @Id  
    private Long id;  
    @OneToMany(mappedBy="cart")  
    private Set<Good> goods = new HashSet();  
  
    public void addGood(Good good) {  
        this.goods.add(good);  
        good.setCart(this);  
    }  
}
```

```
@Entity  
class Good {  
    @Id  
    private Long id;  
    @ManyToOne  
    @JoinColumn(name = "cart_id")  
    private Cart cart;  
  
    public Cart getCart() {  
        return cart;  
    }  
}
```

- mappedBy elementas parodo, kad išorinj raktą turi kita esybė.
- Good lentelėje bus "cart" stulpelis - išorinis raktas į Cart lentelę.
- Good yra "savininkas", todėl kiekvieną kartą pridedant naują prekę į krepšelį, turi būti susiejamas krepšelis, kviečiant savininko susiejimo metodą setCart().

## SĄRYŠIAI: DAUG-SU-DAUG

```
@Entity  
class Cart {  
    @Id  
    private Integer id;  
    @ManyToMany  
    private List<Good> goods;  
}
```

```
@Entity  
class Good {  
    @Id  
    private Integer id;  
    @ManyToMany  
    private List<Cart> carts;  
}
```

- Sąryšis žymimas @ManyToMany anotacija
- lentelės sujungtos trečia jungimo lentele Cart\_Good



# SĄRYŠIAI: DAUG-SU-DAUG

```
@Entity  
class Cart {  
    @Id  
    private Integer id;  
    @ManyToMany(mappedBy="carts")  
    private List<Good> goods;  
}
```

```
@Entity  
class Good {  
    @Id  
    private Integer id;  
    @ManyToMany  
    @JoinTable(table=@Table(name="G_C"),  
    joinColumns=@JoinColumn(name="G_ID"),  
    inverseJoinColumns=@JoinColumn(name="C_ID"))  
    private List<Cart> carts;  
}
```

- sąryšis realizuojamas naudojant asociatyvines lenteles (join table). Šiuo atveju lentelė "G\_C"
- joinColumns nurodo esamos lentelės (Good) id stulpelį, inverseJoinColumns - susietos (Cart)
- jei sąryšis dvipusis, kita sąryšio pusė turi naudoti mappedBy elementą

# KOLEKCIJŲ SAUGOJIMAS



AKADEMIJA.LT  
INFOBALT IR TECH CITY

# JPA PERSIST MAP

- savininko puseje reikalinga nurodyti @MapKey

```
@Entity
public class Author {
    @ManyToMany
    @JoinTable(
        name="AuthorBookGroup",
        joinColumns={@JoinColumn(name="fk_author",
            referencedColumnName="id")},
        inverseJoinColumns={@JoinColumn(name="fk_group",
            referencedColumnName="id")})
    @MapKey(name = "title")
    private Map<String, Book> books = new HashMap<String, Book>();
}
```



## JPA PERSIST MAP

- priklausomai nuo key tipo:
  - `@MapKey` - unikalus atributas Map#value klasės lauke, jei raktas bazinio tipo
  - `@MapKeyEnumerated` - jei raktas Enum tipo
  - `@MapKeyTemporal` - jei raktas Date ar Calendar tipo
  - `@MapKeyJoinColumn(name="id_rakte")` - jei raktas sudėtingesnio Class tipo (raktas - objektas)
    - nesupainiokite: `@MapKeyColumn` - stulpelis su `@ElementCollection`
- raktai-laukai aišku turi būti pažymėti `@Temporal`, `@Enumerated`, raktai-klasės - `@Entity`



## JPA PERSIST COLLECTION

- Map<Long, Long> ar List<Long>
  - galima tiesiog susikurti savo klasę MyNumber1 ir/ar MyNumber2, viduje turėti private Long atributą
  - tuomet tai pavirs į Map<Long, MyNumber1> ar Map<MyNumber1, MyNumber2> ar List<MyNumber1>
- galima bus naudoti įprastas anotacijas @OneToMany, @ManyToMany ir t.t.
- daugiau kontrolės, nes yra atskirose lentelėse ar laukai



## JPA PERSIST COLLECTION

- arba naudoti `@ElementCollection` paprastiems tipams

```
@ElementCollection  
private Collection<String> l = new ArrayList<>();  
@ElementCollection  
private Map<Long, Long> m = new HashMap<>();
```

- bet norint dirbt su klasėmis, reiks žinoti `@Embeded`, `@Embeddable`, `@MapKeyColumn`, `@CollectionTable` ir kitas susijusias anotacijas
- mažiau kontrolės ir sunkiau keisti kodą (refactoring) ir migruoti duomenis



# KASKADINĖS OPERACIJOS

## KASKADINĖS OPERACIJOS

- jei nenurodyta kitaip, EntityManager operacijos taikomos tik tam entity objektui, kuris perduotas kaip parametras.
- Operacijos NEPERSIDUOS kitiems entity objektams, kurie pasiekiami per sąryšius.
- remove() operacijai tai dažnai ir yra norimas veikimas

```
class Cart {  
    @OneToMany  
    private List<Good> goods;  
}  
entityManager.remove(cart); // iš DB išstrins tik Cart
```



## KASKADINĖS OPERACIJOS

- Jeigu turime dvi susietas esybes, turime jas išsaugoti
  - pradedant nuo savarankiškų ir baigiant „savininkais“

```
Customer customer = new Customer();
Credentials cred = new Credentials();
customer.setCredentials(cred);
em.persist(credentials);
em.persist(customer);
```



# KASKADINĖS OPERACIJOS

- Tačiau galima saugojimo operacijas kaskaduoti susijusiems entity objektams, pvz.: persist() gali būti įvykdyta visoms susijusioms esybėms, nors kaip parametras perduodamas tik vienas entity objektas
- Reikia įsitikinti, kad susijusi esybė susieta (merged) prieš kviečiant persist(), t.y. ji negali būti detached

```
@Entity
public class Customer {
    @Id
    public Long id;
    ...
    @OneToOne(cascade=CascadeType.Persist)
    public Credentials credentials;
    ...
}
```



# OPERACIJŲ TIPAI

- Galimi cascade tipai:
  - REMOVE - kaskadinis priklausančių objektų trynimas
  - PERSIST - priklausančių objektų išsaugojimas kartu
  - REFRESH - priklausančių objektų atnaujinimas
  - MERGE - objektų būsenos saugojimas ir užkrovimas
  - DETACH - objektų būsenos atsiejimas
  - ALL - visų operacijų kaskadinis vykdymas priklausantiems objektams
- Dvipusiams ryšiams reikia nurodyti kaskadas iš abiejų pusiu, jeigu norima, kad jos veiktų simetriškai.



## OPERACIJŲ TIPAI

- Dažnai yra naudojami MERGE ir DETACH tipai kartu

```
class User {  
    @ManyToMany(cascade = {CascadeType.MERGE, CascadeType.DETACH})  
    private List<Role> roles;  
}
```

- tai reiškia, kad jei bus atsiejama User esybė, tai automatiškai bus atsietos ir Role esybės
- ir atvirkšiai: saugant User pakeitimus (merge) išsisaugos ir Role pakeitimai



# SĄRYŠIŲ ELEMENTŲ ĮTERPIMAS

```
@Entity  
class Cart {  
    @Id  
    private Long id;  
    @OneToMany(mappedBy="cart",  
        cascade = CascadeType.ALL)  
    private Set<Good> goods = new HashSet();  
  
    public void addGood(Good good) {  
        this.goods.add(good);  
        good.setCart(this);  
    }  
}
```

```
@Entity  
class Good {  
    @Id  
    private Long id;  
    @ManyToOne(cascade =  
        {CascadeType.MERGE, CascadeType.DETACH})  
    @JoinColumn(name = "cart_id")  
    private Cart cart;  
  
    public Cart getCart() {  
        return cart;  
    }  
}
```

- atkreipkite dėmesį dar kartą: addGood metodas yra ne savininko pusėje tam, kad nustatytu ant norimo pridėti savininko.. save
- ManyToMany atveju galioja ta pati taisyklė
  - Cascade PERSIST netiks, nes atbulai kaskadinimas dažniausiai nevyksta jei egzistuoja savininkas-owner (bent jau su Hibernate)



# SĄRYŠIŲ PAŠALINIMAS

- ManyToMany ar kitais atvejais, kai sąryšiams naudojama papildoma lentelė, ką reiškia REMOVE?
  - ar tai sąryšio pašalinimas ?
  - ar tai sąryšio ir susietos klasės objekto (lentelės įrašo) pašalinimas?
- @OneToOne ir @OneToMany turi orphanRemoval

*Whether to apply the remove operation to entities that have been removed from the relationship and to cascade the remove operation to those entities.*



## SĄRYŠIŲ PAŠALINIMAS

```
@OneToOne (orphanRemoval=true)  
@OneToMany (orphanRemoval=true)
```

- kuris reiškia, kad bus trinami ne tik sąryšiai, bet ir surištas objektas, jei jis yra "našlaitis" (angl. orphan), t.y. prie nieko neprikabintas
  - ir jei jis nėra detached, nėra new, ir nėra removed



## UŽDUOTIS #1 - ONETOONE

- sukurti prekei klasę ProductDetails papildomai informacijai saugoti
- ProductDetails turi turėti image ir description
  - prie produkto šių laukų turi nebelikti
- įgyvendinti OneToOne ryšį tarp Product ir ProductDetails
  - t.y. jei anksčiau turėjome product.getDescription() tai dabar bus product.getProductDetails().getDescription()
- turi pilnai veikti CRUD operacijos tiek produktui, tiek produkto papildomos informacijos esybei
- neturi kristi išimčių (exception)



## UŽDUOTIS #1 - MANYTOMANY

- įgyvendinti ManyToMany ryšį tarp krepšelio ir prekių
- turi pilnai veikti CRUD operacijos tiek krepšeliui, tiek prekėms
- neturi kristi išimčių (exception)



# ATIDÉTAS UŽKROVIMAS

## ATIDĖTAS UŽKROVIMAS (LAZY LOADING)

- retai kada prieikia visų objekto sąryšių vienu metu
  - dažnai užtenka vieno ar poros sąryšių
- programos greitaveika gali būti optimizuojama nurodant, kad sąryšių užkrovimas yra atidėtas iki to momento, kol bus „paprašyta“
  - iškviestas getXxx() metodas
- Tai vadinama lazy loading
  - prisiminkime lazy loading beans



## ATIDĖTAS UŽKROVIMAS (LAZY LOADING)

- Jeigu sąryšio fetch mode nėra nurodyta:
  - Vienos reikšmės sąryšiuose susijęs objektas užkraunamas iš karto, neatidedant.
  - Kolekcijos tipo sąryšiai pagal nutylėjimą – atidedami.
- Dvipusiuose sąryšiuose iš vienos pusės gali būti nustatytas atidėtas, iš kitos - momentinis užkrovimas
  - Normalu, kad poreikis gali skirtis, priklausomai iš krypties iš kurios daroma užklausa



## ATIDĘTAS UŽKROVIMAS (LAZY LOADING)

- Atidėto užkrovimo direktyva `lazy` gali būti ignoruojama JPA implementacijos, nes iš principio paankstinus nebus pažeistas korektišumas.
- Priešingai – išankstinio užkrovimo direktyva negali būti ignoruojama, nes sugriautų objekto korektiškumą, jeigu po užkrovimo entity pereitų į detached būseną – sąryšis taptų nebepasiekiamas.



## ATIDĒTAS - LAZY

```
class Cart {  
    @OneToMany(fetch=FetchType.LAZY)  
    private List<Good> goods;  
    public List<Good> getGoods() { return goods; }  
}
```

- Goods iš DB bus atsiųstas tik iškvietus getGoods(...) metodą
- Jei metodas niekada nebus iškviestas - duomenys niekada nebus užkrauti iš DB



## IŠ KARTO - EAGER

```
class Cart {  
    @OneToMany(fetch=FetchType.EAGER)  
    private List<Good> goods;  
    public List<Good> getGoods() { return goods; }  
}
```

- Duomenys užkraunami iš kart



# **ENTITY PAVELDĘJIMAS, VALIDAVIMAS, INDEKSAI IR KITA**

## DUOMENŲ TIKRINIMAS - VALIDACIJA

- Bean Validation – taip pat kaip ir RestController validavimo atveju:
  - @NotNull
  - @DecimalMin ar @Digits
  - @Min ir @Max
  - @Size(min,max)
  - @Future ir @Past
- Kitas anotacijas galima rasti [JSR 380 standarte](#)
- Ijungtas ne anotacija, o application.properties faile
  - nustatyti none norint išjungti ([dokumentacija](#))



## INDEKSŲ GENERAVIMAS

- indeksus dažniausiai naudojame:
  - jei nenorime tikrinti, ar jau egzistuoja kokių nors duomenų, kurie turi būti unikalūs (pvz. email)
  - jei norime, kad paieška ar rūšiavimas vyktų greičiau
- @Index anotacija leidžia apsirašyti ne-automatinius (PK/FK/Unique) indeksus
- @Index naudojama tik kaip kitų anotacijų sudedamoji dalis: Table, SecondaryTable, CollectionTable, JoinTable, TableGenerator



# INDEKSŲ GENERAVIMAS

```
@Entity
@Table(name = "region", indexes = {
    @Index(name = "idx_code", columnList="code", unique = true),
    @Index(name = "idx_name", columnList="name ASC", unique = false),
    @Index(name = "idx_name_code", columnList="name,code DESC")
})
public class Region{
    @Column(name = "code", nullable = false)
    private String code;
    @Column(name = "name", nullable = false)
    private String name;
}
```



## DUOMENŲ PATIKRINIMAS AR INDEKSAI?

- Jeigu naudojate pvz. `existsBy` duomenų patikrai prieš kuriant įrašą, pirma jūs galite patikrinti, o tuo metu kas nors gali įdėti toki patį įrašą dar prieš jums spėjant įvykdyti transakciją
- Todėl norint užtikrinti, kad laukas bus unikalus, vis tiek reikia prie lentelės sukurti `@Index` unikalų indeksą

```
@Table(name = "user", indexes = {  
    @Index(name = "idx_nickname",  
           columnList = "nickname",  
           unique = true) } )
```



## JPA LIFECYCLE METODAI

- Specialios anotacijos entity metodams pažymėti, jeigu jie turi būti iškviečiami tam tikru momentu gyvavimo cikle
  - @PrePersist
  - @PreRemove
  - @PostPersist
  - @PostRemove
  - @PreUpdate
  - @PostUpdate
  - @PostLoad
- daugiau informacijos **specifikacijoje** ir **čia**



# JPA LIFECYCLE METODAI

- turėdami faktūrą ir kliento duomenis joje, galime automatiškai sukurti klientą, tiesiog saugodami faktūrą

```
@Entity class Invoice {  
    private Customer cust;  
    private String name;  
    private Address address;  
    @PreCreate  
    public void onPreCreate() {  
        // Automatically create a new customer  
        if (getCustomer() == null) {  
            Customer cust = new Customer();  
            cust.setName(getName());  
            cust.setAddress(getAddress());  
            cust = em.merge(cust); // attach this new entity  
            setCustomer(cust); // and here we change a relationship  
        }  
    }  
}
```



## ENTITY PAVELDĘJIMAS - TĖVAS

- kai norime aprašyti bendras savybes grupei Entity klasių, tačiau bazinės klasės nenorime turėti kaip atskirą entity:

```
@MappedSuperclass
public class Person {
    @Id
    protected Long id;
    protected String name;
    @Embedded
    protected Address address;
}
@Entity
public class Customer extends Person {
    @Transient
    protected int orderCount;
    @OneToMany
    protected Set<Order> orders = new HashSet();
}
```



# ENTITY PAVELDĖJIMAS - VIENOJE LENTELĖJE

- kai norime aprašyti bendras savybes grupei Entity klasių, tačiau bazine klasės nenorime turėti kaip atskirą entity:

```
@Entity
@Table(name = "VEHICLE")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "VEHICLE_TYPE")
public abstract class Vehicle { /* Vehicle class code */ }
@MappedSuperclass public abstract class PassengerV extends Vehicle {
    @Entity
    @DiscriminatorValue(value = "Bike")
    public class Bike extends PassengerV { }
    @Entity
    @DiscriminatorValue(value = "Car")
    public class Car extends PassengerV { }
```

- daugiau informacijos [čia](#) ir [čia](#)

# PAPILDOMOS GALIMYBĖS

- Rakinimas (Locking) yra technika norint suvaldyti situacijas, kai tuos pačius duomenis vienu metu keičia keli naudotojai. Užrakinant duomenis užtikriname, kad tik vienas iš naudotojų vienu metu galės keisti duomenis.
  - JPA palaiko pesimistinį ir optimistinį režimus
    - kontroliuojama su anotacijomis @Version ir @LockModeType
    - Optimistinis: visiem leidžiama skaityti ir keisti, prieš commit įvyksta patikrinimas, ar įrašo versija nepasikeitė. Jei pasikeitė – metamas exception. Geresnė greitaveika.
    - Pesimistinis: užrakinama DB įrašo lygyje ir neleidžiama rašyti ir/arba skaityti.
- Trys būdai, kaip atlikti rakinimą:
  - EntityManager methodai: lock, find, refresh
  - Query metodas: setLockMode
  - NamedQuery anotacija: lockMode elementas



## KITA, KAS SVARBU

- kaip spręsti problemas pesimistic/optimistic lock
- Criteria API
- JPQL JOIN TREAT
- reikšmių konvertavimas su @Convert
- @NamedEntityGraph
- @ConstructorResult
- Java klasių generavimas iš SQL ir DB lentelių



# NUORODOS

- JPA 2.1 Specification PDF
- EE7/Persistence Tutorial
- JPQL pavyzdžiai čia ir čia
- JPQL sintaksė
- Vidinės užklausos ir Criteria
- NamedEntityGraph čia ir čia
- Dinamiškai formuojamos vardinės užklausos
- JSR-338 Java Persistence 2.1 Oficiali dokumentacija
- JPA 2.x specification
- Hibernate Community Documentation, EntityManager



## UŽDUOTIS #2 - PAVELDĘJIMAS

- prekę pakeisti į vienos lentelės paveldimumo tipo esybę su diskriminatoriumi
- praplėsti prekę keliomis prekių grupėmis, pvz. Clothes ir Toys
- įsitikinti, kad tokias prekes iš lentelės įmanoma nuskaityti
  - kad jas įmanoma išsaugoti
  - kad jas galima pridėti ir išimti iš krepšelio



# KITOJE PASKAITOJE

Spring Security. Live coding



# IŠ PRAEITOS PASKAITOS

- React atributų perdavimas event metu
- perduoti per funkciją

```
<input onClick={ (event) => this.onClick(event, papildomas) } />
```

- įsidėti į papildomą komponentą ir perduoti per props

```
class Komponentas extends Component {  
    handleClick = (event) => {  
        this.props.onClick(this.props.papildomas);  
    }  
    render() { return (  
        <input onClick={this.handleClick} />  
    ); }  
}  
// tuomet panaudojimas būtu toks  
<Komponentas papildomas={papildomas}  
    onClick={this.onKomponentoClick} />
```





# AKADEMIJA.IT

INFOBALT IR TECH CITY

## SPRING SECURITY. LIVE CODING. PRAKTIKA

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# TURINYS

- Spring Security
- Live Coding
- Praktika



# TURINYS

- Kas yra Spring Security
- Prijungimas prie projekto
- Prijungimas prie UI



# KAS YRA SPRING SECURITY

## KODĖL ATSIRADO?

- Saugumas - svarbus aspektas be išimties visoms aplikacijoms
- Saugumas stipriai įtakoja aplikacijos funkcionalumą
- Aplikacija saugumu pati rūpintis neturėtų
  - todėl norima atskirti saugumą nuo aplikacijos funkcijų



## TRUMPA ISTORIJA

- 2003 sukurtas Acegi Security
  - saugumo servisai Spring karkasui
- Nuo 1.1.0 Acegi tapo Spring moduliu



# KAS YRA SPRING SECURITY

*Spring Security is a powerful and highly customizable authentication and access-control framework. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.*

- Authentication - ar asmuo dedasi tuo, kuo sako
  - pvz. per slaptažodj
- Authorization - taisyklės, nusakančios, kas gali ką daryti
  - pvz. per roles (angl. ROLE)



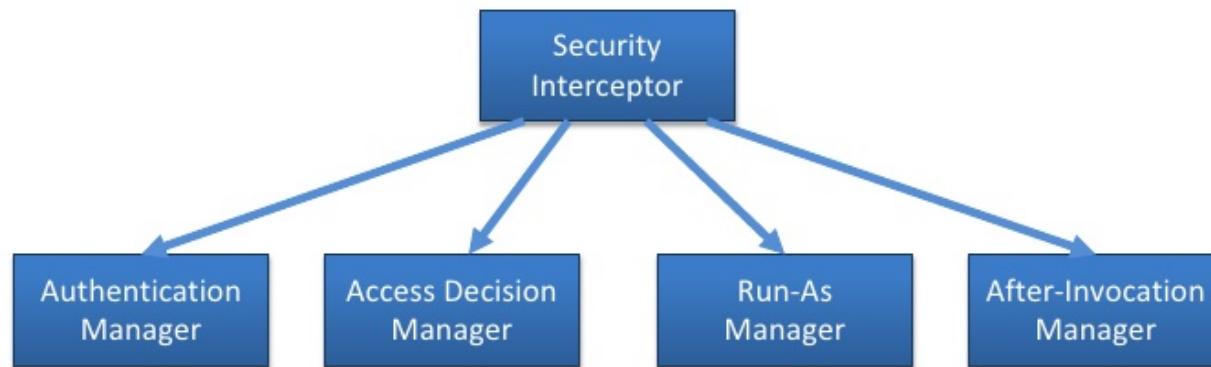
## PAGRINDINIAI ELEMENTAI

- Filtrai (Security Interceptor)
  - praktiškai patikrina username ir password
  - deleguoja užklausas Manager'iams
- Autentifikacija
- Autorizacija
  - Web
  - Metodai



# SECURITY INTERCEPTOR

*Fundamental elements of Spring Security*

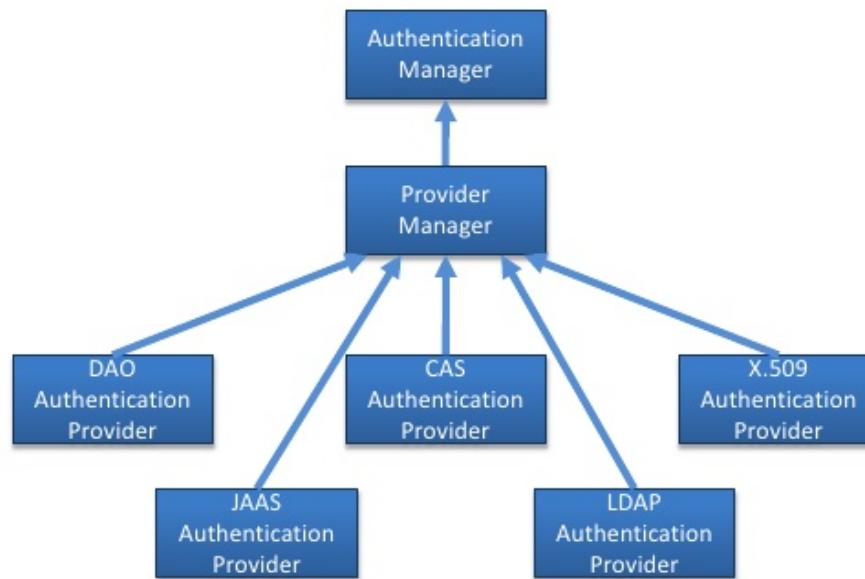


# AUTHENTICATION MANAGER

## Authentication Manager



- verifies *principal (typically a username) and credentials (typically a password)*
- Spring Security comes with a handful of flexible authentication managers that cover the most common authentication strategies



# PRIJUNGIMAS PRIE PROJEKTO

# PRIKLAUSOMYBĖ

- jungiame prie Spring Boot projekto

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

- application.properties

```
server.session.cookie.name = SECURITYID
```



# KLAIDOS IŠMETIMAS REST

- SecurityEntryPoint.java kad išmestų 401 klaidą

```
@Component("restAuthenticationEntryPoint")
public class SecurityEntryPoint implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException authException) throws IOException, ServletException
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
    }
}
```



# DUOMENŲ SAUGOJIMAS

- jau turimoje klasėje prisidedam slaptažodį

```
@Entity
@Table(name = "Naudotojas")
public class User {
    <..>
    @NotBlank
    private String password;
    <..>
    @Email
    @Size(min=6)
    private String email;
    @ManyToOne(cascade = {CascadeType.MERGE, CascadeType.DETACH})
    @JoinColumn(name = "ROLE_ID")
    private Role role;
}
```

- aišku vietoje email galima naudoti tiesiog pvz username



# NAUDOTOJŲ SERVISAS

- savo naudotojų servise implementinti UserDetailsService

```
@Service
public class UserService implements UserDetailsService { // <..>
    @Override
    public UserDetailsService loadUserByUsername(String username)
        throws UsernameNotFoundException {
        User user = findByEmail(username);
        if (user == null)
            throw new UsernameNotFoundException(username + " not found.");
        return new org.springframework.security.core.userdetails.User(
            user.getEmail(), user.getPassword(),
            AuthorityUtils.createAuthorityList(
                new String[] { "ROLE_" + user.getRole().getName() } ) );
    } // <..>
    @Transactional(readOnly = true)
    public User findByEmail(String email) {
        return userRepository.findByEmail(email); } }
```



# NAUDOTOJŲ SERVISAS

- kuriant naują naudotoją reikia užkoduoti slaptažodį su PasswordEncoder, šiuo atveju su numatytuoju Spring

```
User newUser = new User();
newUser.setUsername(username);
PasswordEncoder encoder =
    PasswordEncoderFactories.createDelegatingPasswordEncoder();
newUser.setPassword(encoder.encode(password));
Role r = new Role();
r.setName("CALC");
newUser.setRole(r);
User saved = userRepository.save(newUser);
```



# KONFIGŪRACIJA

- sukurti naują SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled=true, prePostEnabled=true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private SecurityEntryPoint securityEntryPoint;
    @Autowired
    private UserDetailsService userService;
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
        throws Exception {
        auth.userDetailsService(userService);
        //auth.inMemoryAuthentication().withUser("uu")
        //  .password("pp").roles("USER", "CALC");
    }
}
```



# KONFIGŪRACIJA

- taip pat SecurityConfig.java prijungti:

```
public class SecurityConfig <..> {  
    // <..>  
    @Override  
    protected void configure(HttpSecurity http) throws Exception {  
        http  
            .authorizeRequests()  
                // be saugomo UI dalis ir swaggeris  
                .antMatchers("/", "/swagger-ui.html").permitAll()  
                // visi /api/ saugus (dar galima .anyRequest() )  
                .antMatchers("/api/**", "/calc/**").authenticated()  
            .and()  
        // <..>
```



# KONFIGŪRACIJA

- taip pat SecurityConfig.java prijungti:

```
// <..>
    .formLogin() // leidziam login
    // prisijungus
    .successHandler(new SimpleUrlAuthenticationSuccessHandler())
    // esant blogiems user/pass
    .failureHandler(new SimpleUrlAuthenticationFailureHandler())
    .loginPage("/login").permitAll() // jis jau egzistuoja !
    .and()
// <..>
```



# KONFIGŪRACIJA

- taip pat SecurityConfig.java prijungti:

```
// <..>
    .logout().permitAll() // leidziam /logout
.and()
    .csrf().disable() // nenaudojam tokenu
        // toliau forbidden klaidai
    .exceptionHandling()
        .authenticationEntryPoint(securityEntryPoint)
.and()
    .headers().frameOptions().disable(); // H2 konsolei
}
}
```



# KONFIGŪRACIJA

- aišku prisijungus galima grąžinti username:

```
.successHandler(new AuthenticationSuccessHandler() {  
    @Override  
    public void onAuthenticationSuccess(HttpServletRequest request,  
                                         HttpServletResponse response, Authentication authentication)  
        throws IOException, ServletException {  
        response.setHeader("Access-Control-Allow-Credentials", "true");  
        response.setHeader("Access-Control-Allow-Origin",  
                          request.getHeader("Origin"));  
        response.setHeader("Content-Type",  
                          "application/json; charset=UTF-8");  
        response.getWriter().print("{\"username\": \"" +  
                                 SecurityContextHolder.getContext().getAuthentication().getName  
                               + "\"}");  
    }  
})
```



# SAUGUMO ĮJUNGIMAS SERVISE

- bet kuriame servise, pvz. kalkulatoriaus:

```
@RestController
public class HelloController {
    @RequestMapping(value = "calc", method = RequestMethod.GET)
    // Preauthorized galima or, secured - tik and
    @PreAuthorize("hasRole('CALC')") // @Secured("ROLE_CALC")
    public Result calc(@RequestParam int left, @RequestParam int right
        //<..>
    }
}
```

- Neprisijungus bus Forbidden
  - Prisijungus be CALC rolės - access denied
- O kaip prisijungti per UI?

# KAIP GAUTI PRISIJUNGUSĮ NAUDOTOJĄ

- bet kuriame servise, pvz. kalkulatoriaus:

```
@RestController
public class HelloController {
    @RequestMapping(path = "/loggedUsername", method = RequestMethod.GET)
    public String getLoggedInUsername() {
        Authentication authentication =
            SecurityContextHolder.getContext().getAuthentication();
        if (!(authentication instanceof AnonymousAuthenticationToken))
            String currentUserName = authentication.getName();
            return currentUserName;
    }
    return "not logged";
}
```



# PRIJUNGIMAS PRIE UI



# FORMA

- Login formos puz.:

```
const Forma = ({email, pass, onPassChange, onEmailChange, onSubmit},  
    context) => {  
  return <form onSubmit={onSubmit}>  
    <input type="text" value={email} onChange={onEmailChange} />  
    <input type="password" value={pass} onChange={onPassChange} />  
    <input type="submit"/>  
  </form>;  
}
```



# FORMACONTAINER

- Kaip siunčiame užklausas iš <http://localhost:3000/>:

```
import axios from 'axios';
axios.defaults.withCredentials = true; // leidzia dalintis cookies
class FormaContainer extends Component {
  onEmailChange=(event)=>{this.setState({email:event.target.value}) }
  onPassChange=(event)=>{this.setState({pass:event.target.value}) }
  onSubmit = (event) => {
    let userData = new URLSearchParams();
    userData.append('username', this.state.email);
    userData.append('password', this.state.pass);
    axios.post('http://localhost:8081/login', userData,
      {headers:{'Content-type':'application/x-www-form-urlencoded'}})
      .then((resp) => {
        console.log("user "+resp.data.username+" logged in") })
      .catch((e) => { console.log(e); });
    event.preventDefault();
  }
}
```



# FORMACONTAINER

```
class FormaContainer extends Component {  
    // <..>  
    render() {  
        return <Forma email={this.state.email} pass={this.state.pass}  
            onEmailChange={this.onEmailChange}  
            onPassChange={this.onPassChange}  
            onSubmit={this.onSubmit} />;  
    }  
    onCalc = (event) => {  
        axios.get('http://localhost:8081/calc?left=1&right=2')  
            .then((response) => { console.log(response); })  
            .catch((e) => { console.log(e); });  
        event.preventDefault();  
    }  
}
```



# KAIP PADIDINTI SAUGUMĄ?



AKADEMIJA.IT  
INFOBALT IR TECH CITY

## SLAPTAŽODŽIO KODAVIMAS

- Slaptažodžio kodavimas
- HTTPS tiek Rest, tiek UI
- CSRF tokenai formoms
- išorinė duomenų bazė (LDAP ar kitokia)
- ir kiti būdai



# SLAPTAŽODŽIO KODAVIMAS

```
public class SecurityConfig <..> {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(); // galima pakeisti
    }
    //<..>
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
        throws Exception {
        auth.userDetailsService(userService)
            .passwordEncoder(passwordEncoder());
    }
}
```



## UŽDUOTIS #1

- jei niekada nedarėte, nukopijuokite React/Npm aplikacijos katalogą ir pakeiskite aplikacijos pavadinimą
- nusikopijuokite Spring Boot/Maven katalogą ir pakeiskite pavadinimą, paketą ir versiją savo Java projektui
- įsitikinkite, kad pakeitus pavadinimus projektai kompiliuoja ir aplikacijos pasileidžia
  - egzamino metu bus nurodyta, kaip turi būti suformuoti aplikacijų pavadinimai, kokie paketai naudojami, kokia turi būti versija



## UŽDUOTIS #2

- jau esate susikūrė ManyToMany ryšį (Krepšelis<->Prekė)
- sukurkite Rest servisą, kurio pagalba galima viena Rest užklausa pridėti keletą prekių į krepšelį
- sukurkite Rest servisą, kuris galėtų priimti visas krepšelyje pakeistas Quantity reikšmes ir atnaujinti jas visiems produktams iškart viena užklausa
  - aišku turi būti patikra, ar kiekvienos prekės norimas pirkti kiekis quantity neviršija ant prekės nustatytojo (neviršija visų prekių sandėlyje skaičiaus)



## UŽDUOTIS #3

- taip pat patikrinimas turėtų tikrinti, kad į kitus krepšelius nėra pridėta prekių daugiau nei quantity, ir jeigu dabartiniam krepšeliui prašoma per daug prekių, - neleisti
- turi būti suformuotas klaidos pranešimas su sąrašu įvykusių klaidų
  - galiausiai pranešimas turi pasirodyti React aplikacijoje gražiai suformatuotu tekstiniu pavidalu
  - ir raudonai nuspalvinti atitinkami quantity langeliai
- jeigu tai įgyvendinsite Java priemonėmis, vietoje Java perrašykite į vieną ar kelias sudėtingesnes JPQL užklausas su JOIN sakiniais patikrinimams



## PAPILDOMA UŽDUOTIS #4

- vėl nusikopijuokite React ir Spring Boot aplikacijas
- įjunkite spring boot aplikacijoje saugumą taip, kaip aprašyta skaidrėse
- apsaugokite su anotacijomis egzistuojančius Rest servisus
- React susikurkite prisijungimo formą bei atsijungimo mygtuką
- prisdėkite keletą naudotojų su skirtinomis rolėmis
- pabandykite gauti forbidden ir accesss denied klaidas
- egzamino praktinėje užduotyje Spring Security nebus



# KITOJE PASKAITOJE

Praktika



AKADEMIJA.LT  
INFOBALT IR TECH CITY

# IŠ PRAEITŲ PASKAITŲ

- Norėdami 100% būti tikri, kad jūsų pakeitimai veikia
  - React: išjungti aplikaciją, npm install ir npm start
  - Spring: išjungti aplikaciją, mvn clean install ir paleisti aplikaciją iš naujo
    - Java atveju, jeigu leidžiate iš IDE, tuomet po pakeitimų dažniausiai reikia atnaujinti ir projektą
    - mvn eclipse:eclipse, mvn idea:idea Java projekto atveju ir Refresh/Clean Project
    - Update... ar kitoks jeigu tai Maven IDE tipo projektas



# IŠ PRAEITŪ PASKAITŪ

- `.gitignore`
  - ignoruokite eclipse ar idea sukurtus failus
  - ignoruokite build ir target katalogus
- paprasta Spring Boot aplikacija su target katalogu virtualioje mašinoje:
  - git pull gali užtrukti ir 3 valandas
  - 20 mokinių po 3val = 60 val = per ilgai
  - o dar siunčiant įvykus klaidai reikia pakartoti iš naujo ..
  - jei kada nors jidėjote target/ tai repozitoriją jau reikia kurti iš naujo



## IŠ PRAEITŲ PASKAITŲ

- grįžkite prie React
  - pasižiūrėkite, ar dar prisimente, nes kai kurie jau primiršo



# IŠ PRAEITŪ PASKAITŪ

- Spring Boot galima įjungti actuators, pvz /loggers

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

- application.properties

```
management.endpoints.web.exposure.include=loggers
management.endpoint.loggers.enabled=true
```

- Actuators [dokumentacija](#)
- Galima pakeisti log lygi veikimo metu, pvz.:

```
POST /actuator/loggers/jdbc.sqlonly
{"configuredLevel": "DEBUG"}
```





**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

## JPA PRAKTIKA

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

# TURINYS

- JPA Praktika



## UŽDUOTIS #1

- būtinai pabaikite parduotuvę
  - tiek React, tiek Spring Boot
- jei baigėte - parodykite kodą ir veikimą
  - duosiu individualių patarimų



## UŽDUOTIS #2

- pakartokite skaidres
  - sugalvokite klausimų, ypač jei kažko nežinote ar neatsimenate iš paskaitų
    - nes kitą kartą konsultacija
- peržiūrėkite visų skaidrių visas užduotis
  - jeigu kurios nors nedarėte - atlikite



# KITĄ KARTĄ

Konsultacija



**AKADEMIJA.LT**  
INFOBALT IR TECH CITY

# IŠ PRAEITŲ PASKAITŲ

- JPA leidžia valdyti susijusius Entity patiembs ARBA automatiškai
  - jei patys valdome, nenaudojame Cascade ir orphanRemoval
    - tuomet patiembs reikia eiti Entity medžiu iš apačios į viršų ir rūpintis, kas atsitiks su Entities
    - iš esmės galima sutaupyti konfigūravimo, pvz. daug kur nebūtini atgaliniai ryšiai
  - jei norime, kad valdymas vyktų automatiškai
    - reikia pagalvoti ir sukonfigūruoti iš visų pusų tinkamus operacijų kaskadinimus
    - tinkamai sukonfigūravus, neberekia niekuo rūpintis - užtenka vykdyti operaciją vienam Entity
  - jei norime, galime naudoti abu metodus kartu



# IŠ PRAEITOS PASKAITOS

- Pabandykite susikurti BitBucket Public Repository
  - arba jei nekūrėt - Github Repository

Create a new repository

Name\* Tut2

Description

Access level  This is a private repository

Repository type  Git  
 Mercurial

Project management  Issue tracking  
 Wiki

Language Java

Repository integrations

HipChat  Enable HipChat notifications

**Create repository** Cancel



**AKADEMIJA.IT**  
INFOBALT IR TECH CITY

## KONSULTACIJA

Java Technologijos ir Priemonės

Andrius Stašauskas

andrius@stasauskas.lt

<http://stasauskas.lt/itpro2018/>

## KONTROLINIŲ UŽDUOČIŲ TIKSLAI

- Patikrinti, ką sužinojote
- Patikrinti, kaip greitai galite surasti informaciją
- Užduotys bus vertinamos balais
- Rekomendacijomis bus vertinamos kokybiškai atliktos kontrolinės užduotys



## KONTROLINĖS UŽDUOTYS

- Laikas yra ribojamas
- Po nurodyto laiko sprendimai nepriimami
- Taupykite laiką kiekvienam atsakymui
- Negaiškite laiko prie detalių - pirmiausia atlikite bazines užduotis, tada “gražinkite” kodą (švaresnė projekto struktūra, validacija, gražus kodo išdėstymas, dar gražiau atrodantis puslapis naršyklėje ir pan.)
  - javadoc ir unit testai nebus atskirai vertinami, nebent jų reikės suprasti kodą vertinant kažkurį iš kriterijų



## LAIKAS

- Testas
  - 2018-12-19 Trečiadienis
  - 8:30 - 10:00
  - 1 val. 30 min.
- Praktinė užduotis
  - 2018-12-20 Ketvirtadienis
  - 8:55 - 12:25
  - 4 akademinės valandos + pertraukos
  - ateikite 8:00



# TESTAS

- Testas 30% galutinio balo
  - 50 klausimų per 90 min
    - galima baigti anksčiau
  - klausimai tik su vienu teisingu atsakymu
- Testo pradžioje
  - el. paštas
  - vardas ir pavardė
  - KODAS - **TIK LOTYNIŠKOS RAIDĖS**
    - asmeninis kodas, naudojamas praktinėje užduotyje, taip pat jis bus naudojamas skelbti rezultatams



## TESTO METU

- Nespauskite F12 - google formos gali lūžti. Jeigu reikia ką nors pasibandyti naršyklėje, geriau tai daryti kitoje kortelėje ar puslapyje
- Forma pildoma ilgai, todėl kai tik užbaigsite, nebijokite ir spauskite Submit/Siųsti
  - nuspaudus Submit/Siųsti atsiradusiam lange matysis nuoroda “Edit your response”, arba redaguoti atsakymus. Ją nuspaudę, galėsite grįžti ir atsakymus peržiūrėti bei pakeisti



Jūsų atsakymai buvo išsaugoti

Edit your response



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# PRAKTINĖ UŽDUOTIS

- Praktinė užduotis 70% galutinio įvertinimo
  - užduotims skiriamos keturios pamokos, tai yra 210 min
  - pertraukos nebus
  - priimami daliniai sprendimai (bet kokie sprendimai)
  - KODAS - jūsų asmeninis kodas, įrašytas testo pradžioje
- Balas: 70% įvertinimo
- Užduotis pasibaigus laikui pasiūmama komanda

```
git clone
```

- leidžiama išvalymo komanda

```
mvn eclipse:clean idea:clean
```

- tada

```
mvn clean install
```

- tuomet startuojamas serveris

```
mvn clean install org.codehaus.cargo:cargo-maven2-plugin:1.7.0:run \
-Dcargo.maven.containerId=tomcat8x -Dcargo.servlet.port=8081 \
-Dcargo.maven.containerUrl=http://repo.maven.org/maven2/org/apache/tomcat/tomcat/8.5.35/tomcat-8.5.35.zip
```



## PRAKTINĖS UŽDUOTIES PATEIKIMAS

- GitHub ar BitBucket Public Repository
  - public repository
  - git
  - repository taip pat turi būti naujas, nes kitu atveju bus labai daug metadata ir git clone komanda užtriks ilgai
    - nebent užtikrinsit, kad prieš tai ten nebuvo daug metadata
  - projektas turi būti naujame kataloge sukurtame egzamino dieną



# GALUTINIS VERTINIMAS

- Testo ir kontrolinės užduoties balai sudedami
  - jeigu reikės, apvalinama matematiškai (.5 ir į viršų)
  - formulė: (30% + 70%)
- Užduotis vertinama pagal kriterijus tam tikroje srityje (jų yra apie 20):
  - nėra/blogai/pagal kriterijų labai mažai įgyvendinta
  - pusiau/kažkiek teisingai/pusiau teisingai/pusė padaryta
  - yra/gerai/viskas gerai padaryta/klaidų nėra/viskas įgyvendinta ar panaudota
- Rezultatai paskelbiami vėliausiai iki sausio vidurio
- Rezultatai bus pateikiami viešai <http://stasauskas.lt/itpro2018> pagal jūsų įrašytą kodą, vėliau ir TAMO
- Jeigu prieiks, sutarsime laiką, kurio metu galėsime peržiūrėti rezultatus
  - taip pat galima rašyti el. paštu su klausimais



# PAGRINDINĖS TEMOS

- Git
- Maven ir NPM
- Sistemų architektūros
- JS ir programavimo kalbų istorija
  - be datų, tik React/JS/Java
- Tomcat ir Spring Boot
- React
  - React Router, Axios, AJAX
- Spring ir Spring Boot / Security
- JPA ir Spring Data
- Visos temos, kurios buvo minimos skaidrėse



## PRAKTINĖ UŽDUOTIS

- Sukurti app'są (war failą su Spring Boot REST + javascript React UI) pagal pateiktus reikalavimus
- Galima iš anksto atsinešti savo war failo maven projekta, bet mokėti pakoreguoti projekta ir struktūrą pagal užduotį
- Galima iš anksto atsinešti savo react/js npm projekta, bet mokėti pakoreguoti projekta ir struktūrą pagal užduotį
- Galima naudotis viskuo, išskyrus draugų pagalba
  - Turėkite skaidres kompiuteryje/usb drive
  - Dar kartą pereiname per skaidres
  - Dar kartą peržvelgiame visas praktines užduotis



# ARTĒJA KOLIS



# ARTĒJA KOLIS



AKADEMIJA.IT  
INFOBALT IR TECH CITY

# SĒKMĒS RUOŠIANTIS!



AKADEMIJA.LT  
INFOBALT IR TECH CITY