

# 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 skirtinės 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>



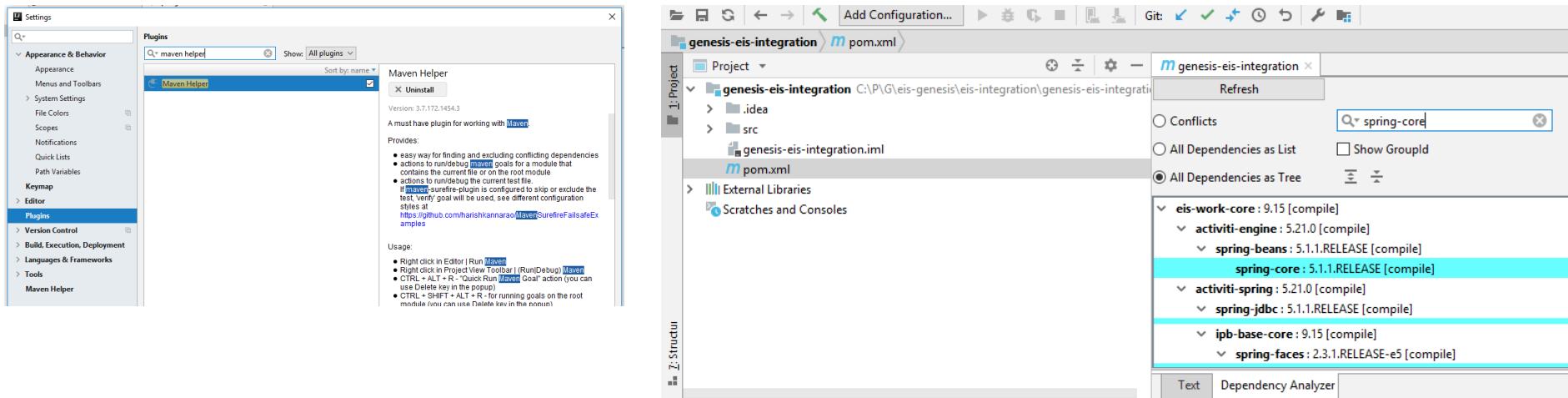
# IDE

- 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>



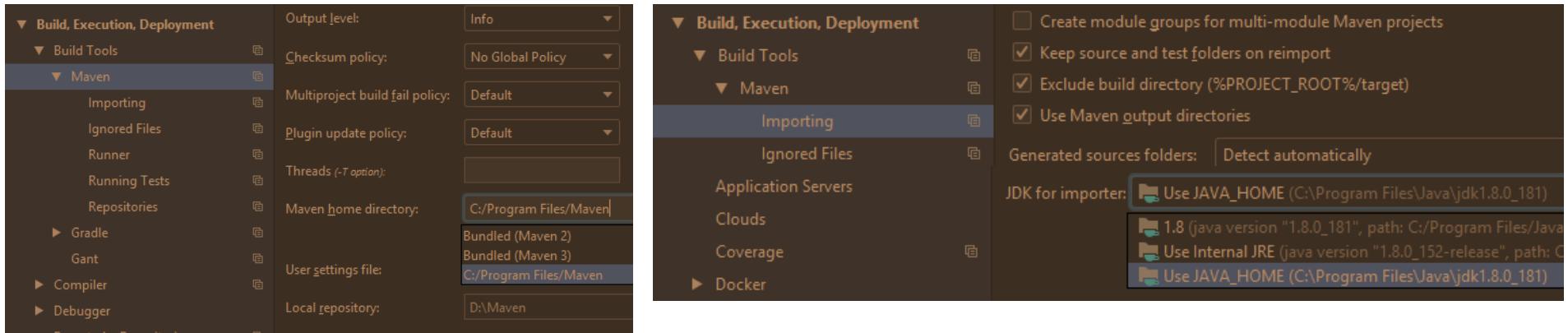
# IŠ PRAEITOS PASKAITOS - IDEA DEPENDENCIES

- 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 parametrum, 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

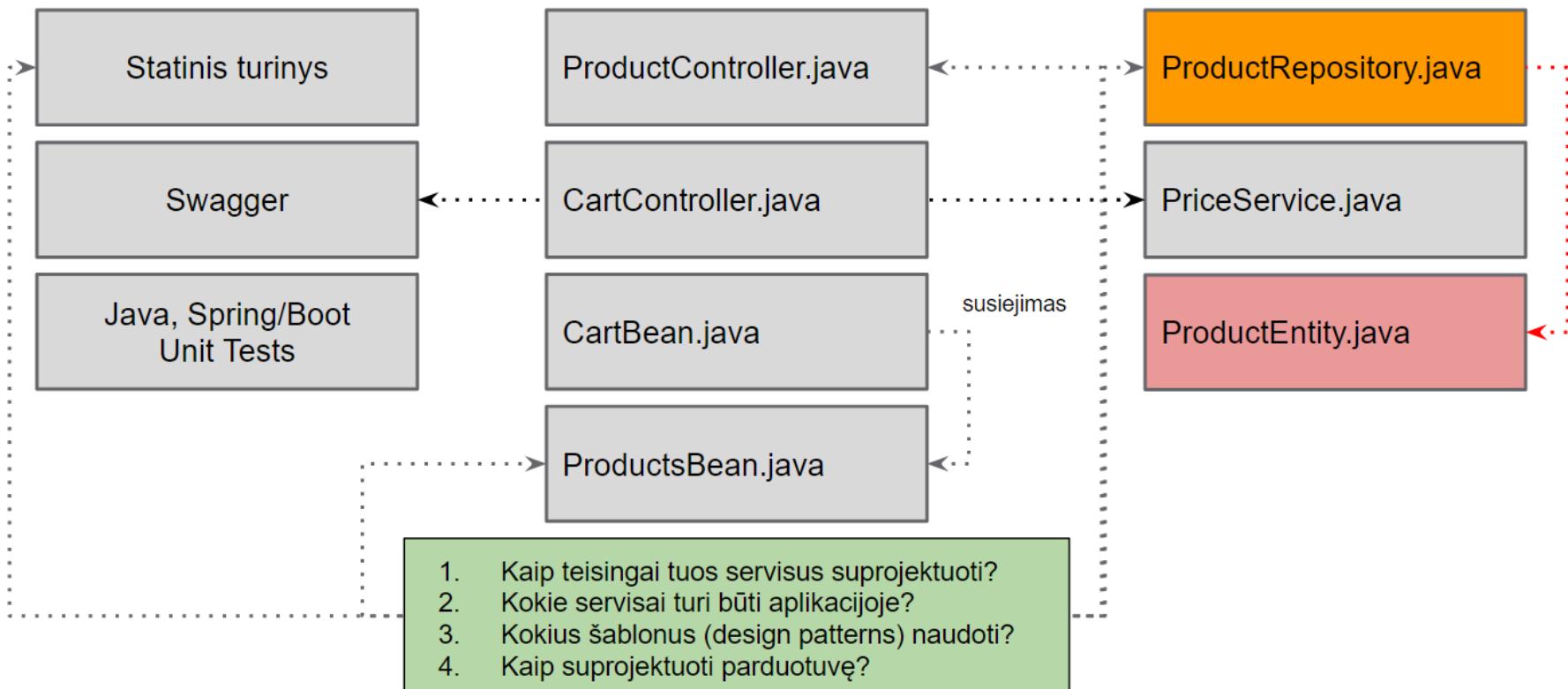
## APLIKACIJOS PROJEKTAVIMAS. PROGRAMAVIMO ISTORIJA

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

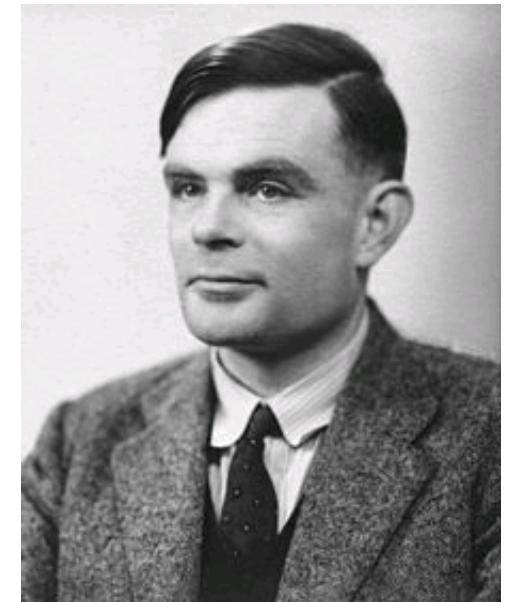


# PROGRAMAVIMO ISTORIJA

# 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

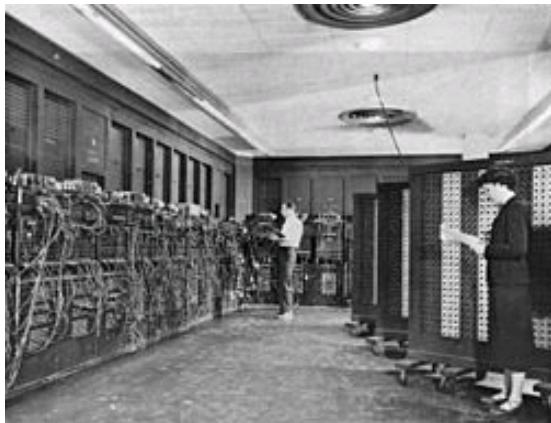


1910-1995



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



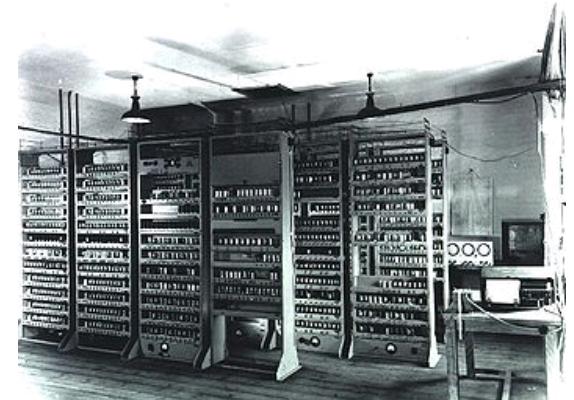
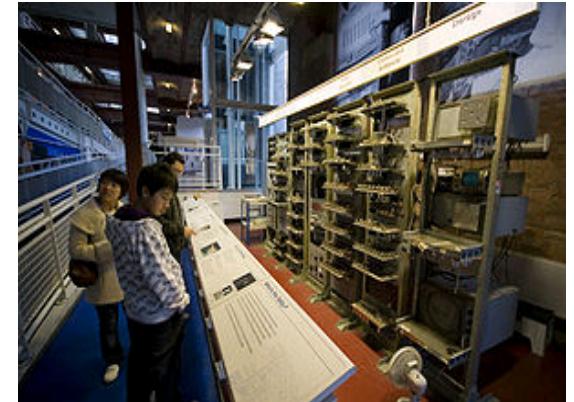
## VON NEUMANNO ARCHITEKTŪRA

- 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 paradigmų
- 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 kiekį:
      - 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}

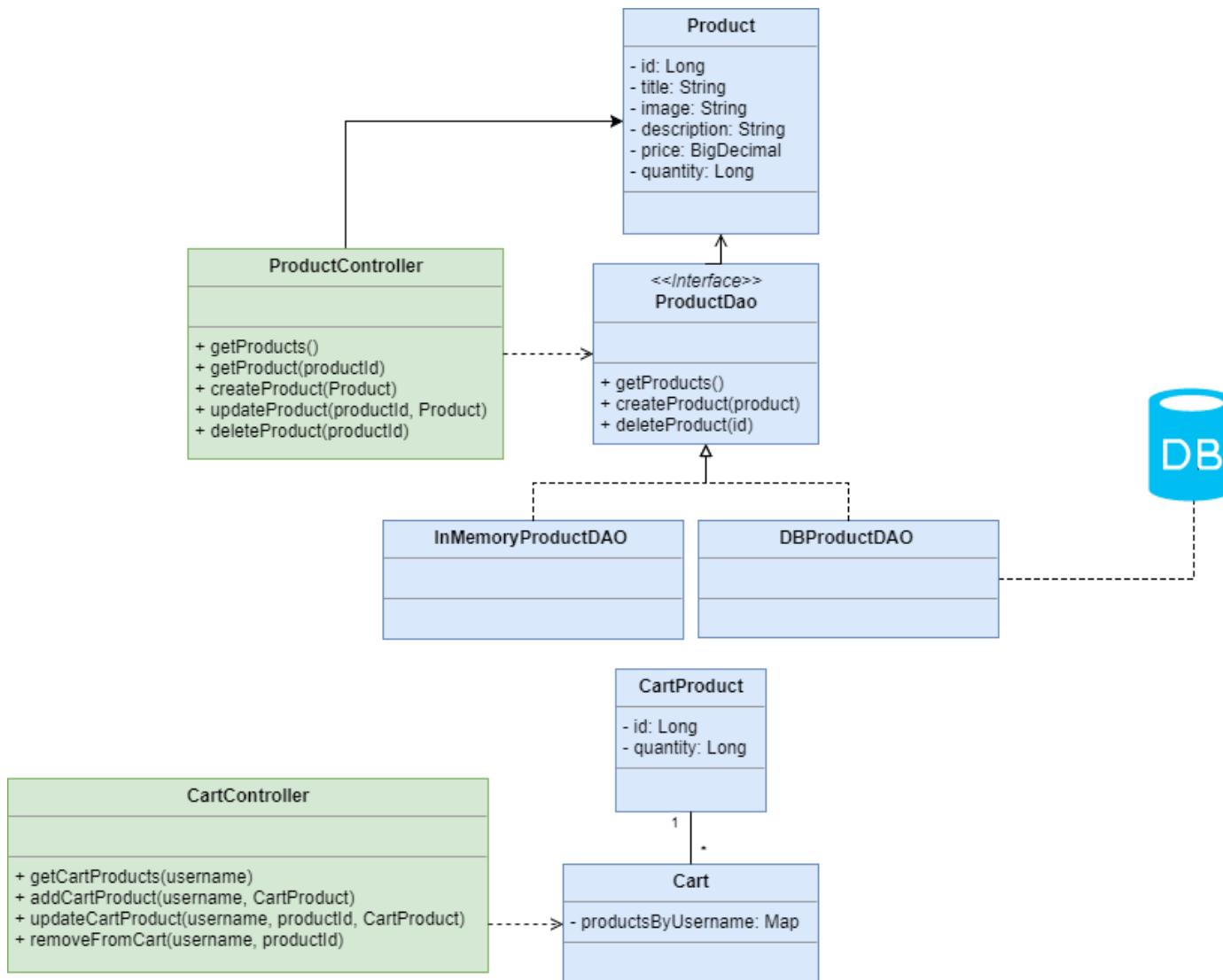


## PLANAS - NARŠYKLĖS URI-LANGAI

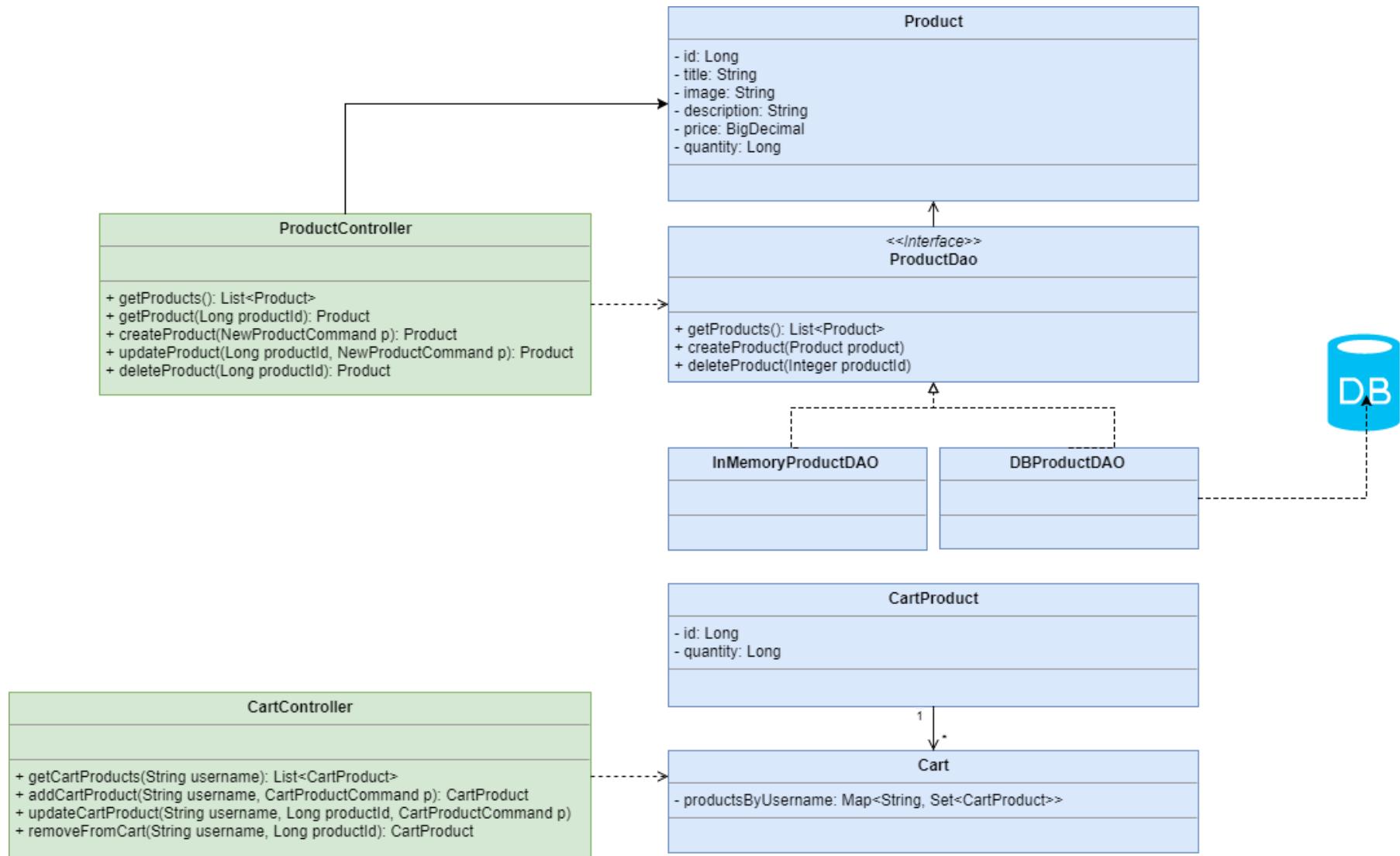
- /products - visi produktai
- /products/:productId - vienas produktas (su mygtuku įdėti į krepšelį)
- /cart - krepšelis (su kieku lauku ir galimybe keisti kiekį)
- /admin - administruojamų produktų sąrašas-lentelė
- /admin/products/new - naujo produkto forma
- /admin/products/:productId - vieno produkto keitimas



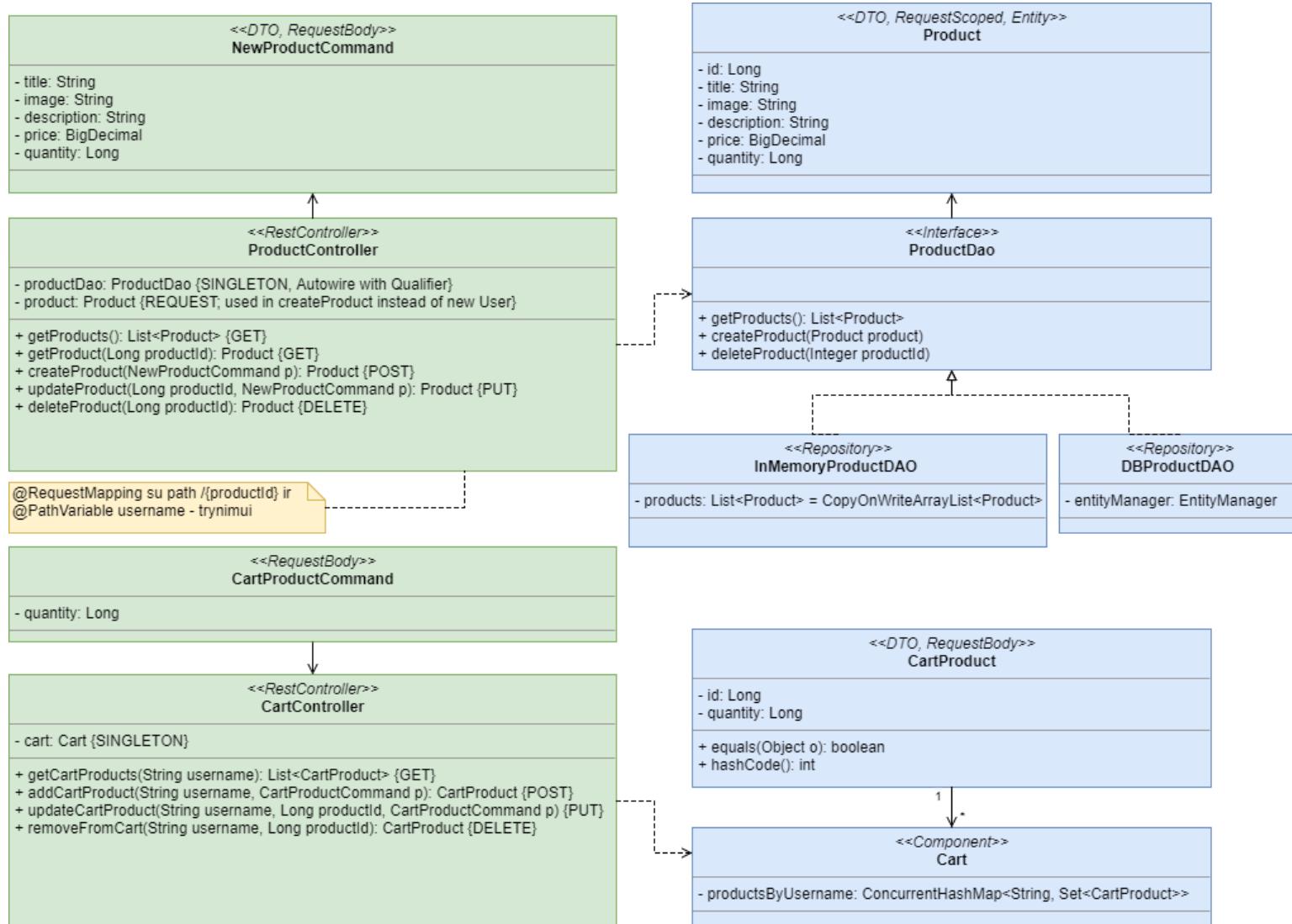
# PLANAS - SPRING JAVA



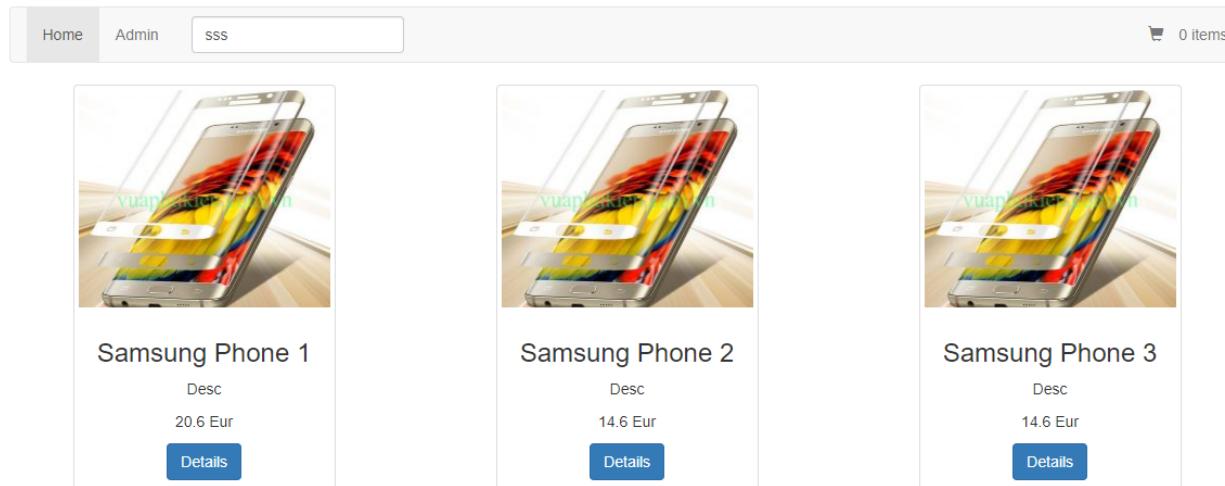
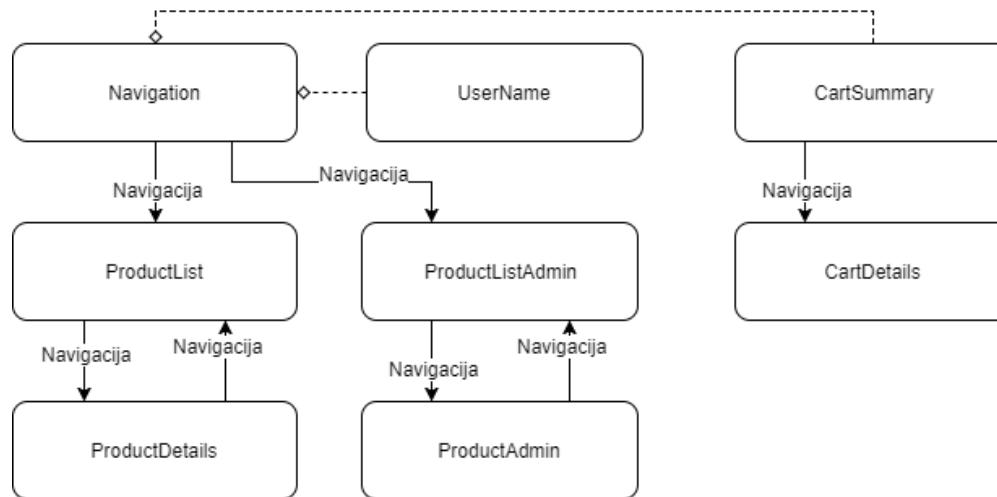
# PLANAS - SPRING JAVA



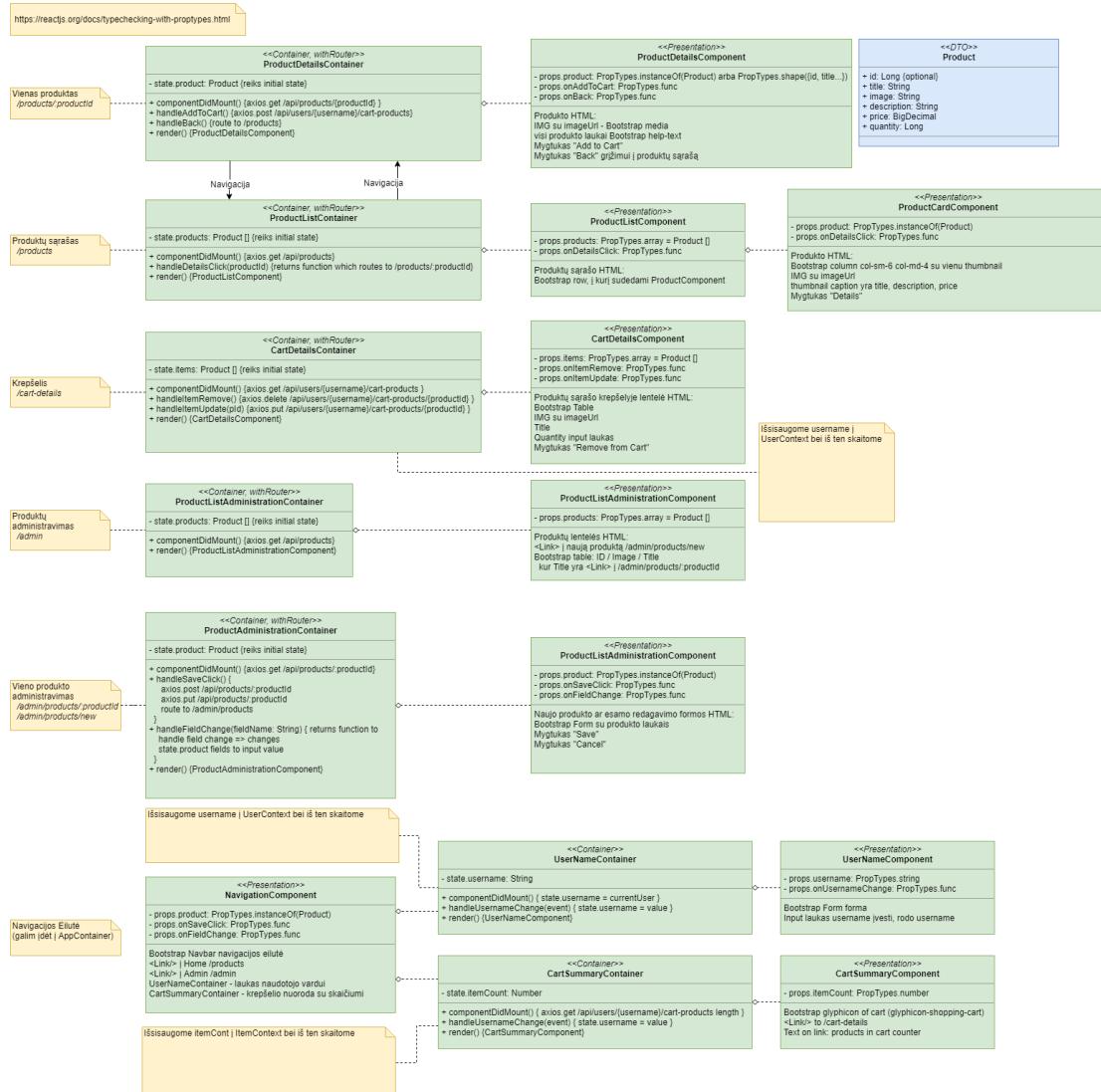
# PLANAS - SPRING JAVA



# 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 naršyklėje 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

