

Универзитет у Београду
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА
Катедра за софтверско инжењерство

Семинарски рад из предмета Напредне софтверске технологије
**Тема: Развој софтверског система за продају чоколада у
Јава програмском окружењу**

Професор: Др Милош Милић

Студент: Даница Здравковић 3709/22

Београд, 2023.

1. Вербални опис

Неопходно је креирати апликацију за вођење евиденције о продаји чоколада преко интернета. Апликацијом је између осталог омогућено чување података о админу, купцима, наруџбинама, ставкама наруџбине, чоколадама и категоријама чоколади.

Креирање налога купца заснива се на уношењу података о имену, презимену, имену купца и лозинци.

Креирање ставке наруџбине се заснива на избору чоколаде и броју комада исте. Купац може да креира више ставки наруџбине. Креирањем наруџбине, памте се ставке наруџбине, датум креирања наруџбине и статус наруџбине.

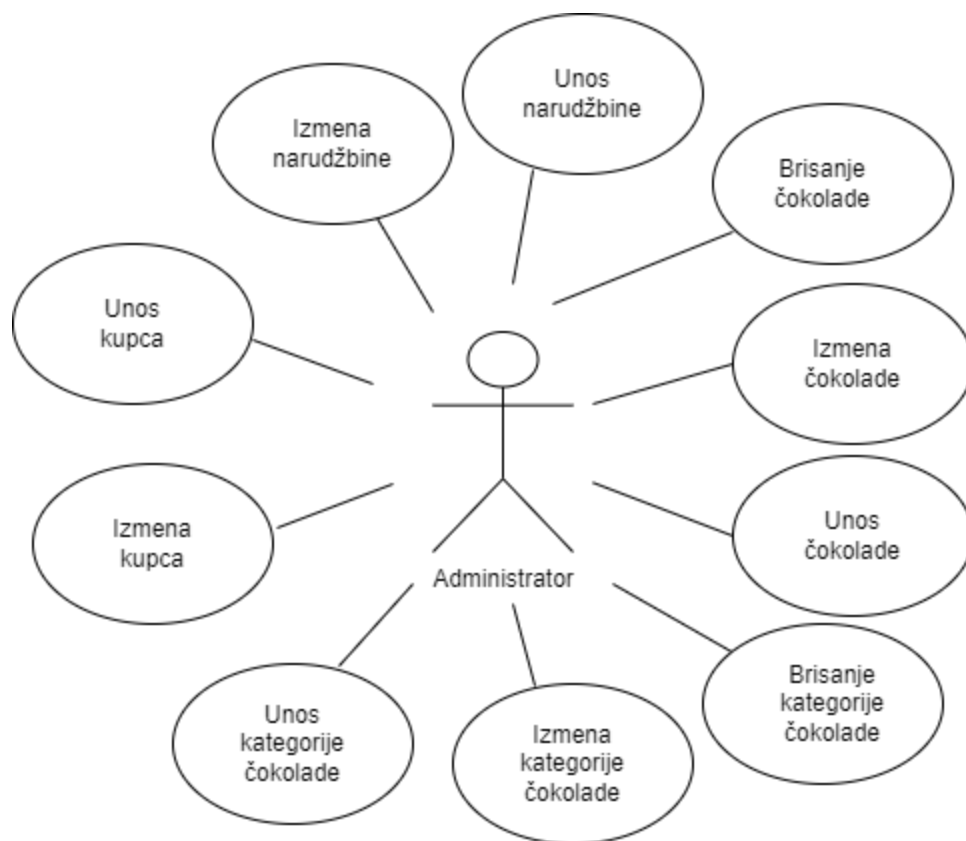
Креирање категорије чоколаде заснива се на уношењу податка о називу и опису категорије.

Креирање чоколаде се састоји од података о називу чоколаде, цени, попусту, опису, линку ка слици, као и категорији којој чоколада припада.

2. Модел случајева коришћења

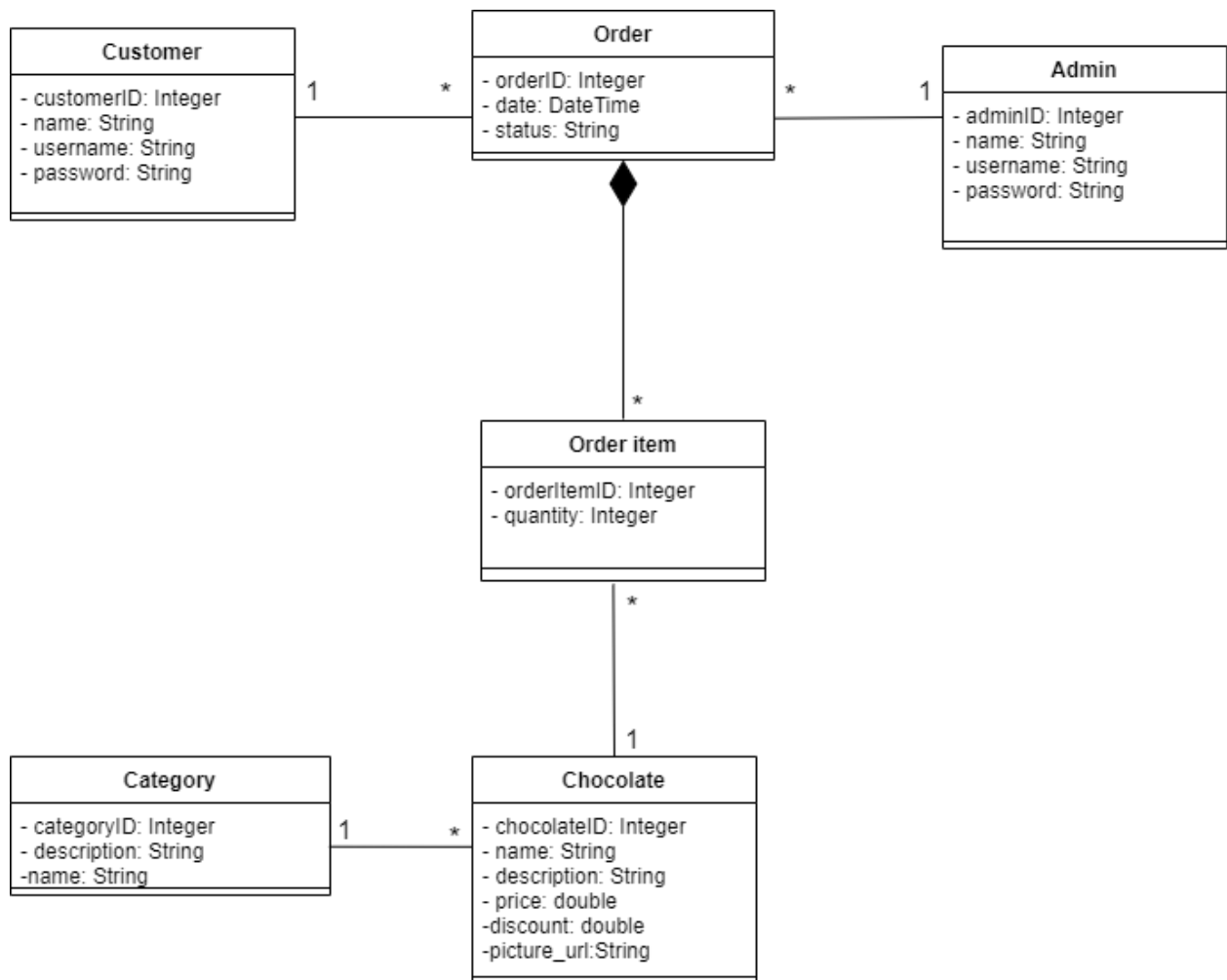
Могу се препознати следећи случајеви коришћења (Слика 1):

1. Унос купца
2. Измена података о купцу
3. Унос чоколаде
4. Измена података о чоколади
5. Брисање чоколаде
6. Унос категорије чоколаде
7. Измена категорије чоколаде
8. Брисање категорије чоколаде
- 9. Унос наруџбине-сложен случај коришћења**
- 10. Измена наруџбине-сложен случај коришћења**



Слика 1-Модел случајева коришћења

3. Концептуални модел



Слика 2-Концептуални модел

4. Релациони модел

На основу концептуалног модела, добија се следећи релациони модел.

Customer(CustomerID, Name, Username, Password)

Admin(AdminID, Name, Username, Password)

Order(OrderID, Date, Status, CustomerID, AdminID)

OrderItem(OrderItemID, Quantity, ChocolateID, OrderID)

Category(CategoryID, Description, Name)

Chocolate(ChocolateID, Name, Description, Price, Discount, Picture_url, CategoryID)

Табела Customer		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADES Order DELETE RESTRICTED Order
	CustomerID	Integer	not null and > 0			
	Name	String	not null			
	Username	String	not null			
	Password	String	not null			

Табела 1-Табела Customer

Табела Admin		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADES Order DELETE RESTRICTED Order
	AdminID	Integer	not null and > 0			
	Name	String	not null			
	Username	String	not null			
	Password	String	not null			

Табела 2-Табела Admin

Табела Category		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT /

	VrstaCokoladeID	Long	not null and > 0			UPDATE CASCADES Chocolate
	CategoryID	Integer	not null and >0			DELETE RESTRICTED Chocolate
	Name	String	not null			
	Description	String	not null			

Табела 3-Табела Category

Табела Chocolate		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	
	ChocolateID	Integer	not null and > 0			INSERT RESTRICTED Category
	Name	String	not null			UPDATE RESTRICTED Category
	Description	String	not null			DELETE /
	Price	double	>0			
	Picture_url	String	not null			
	Discount	double	> 0			
	CatgeoryID	Integer	not null and >0			

Табела 4-Табела Chocolate

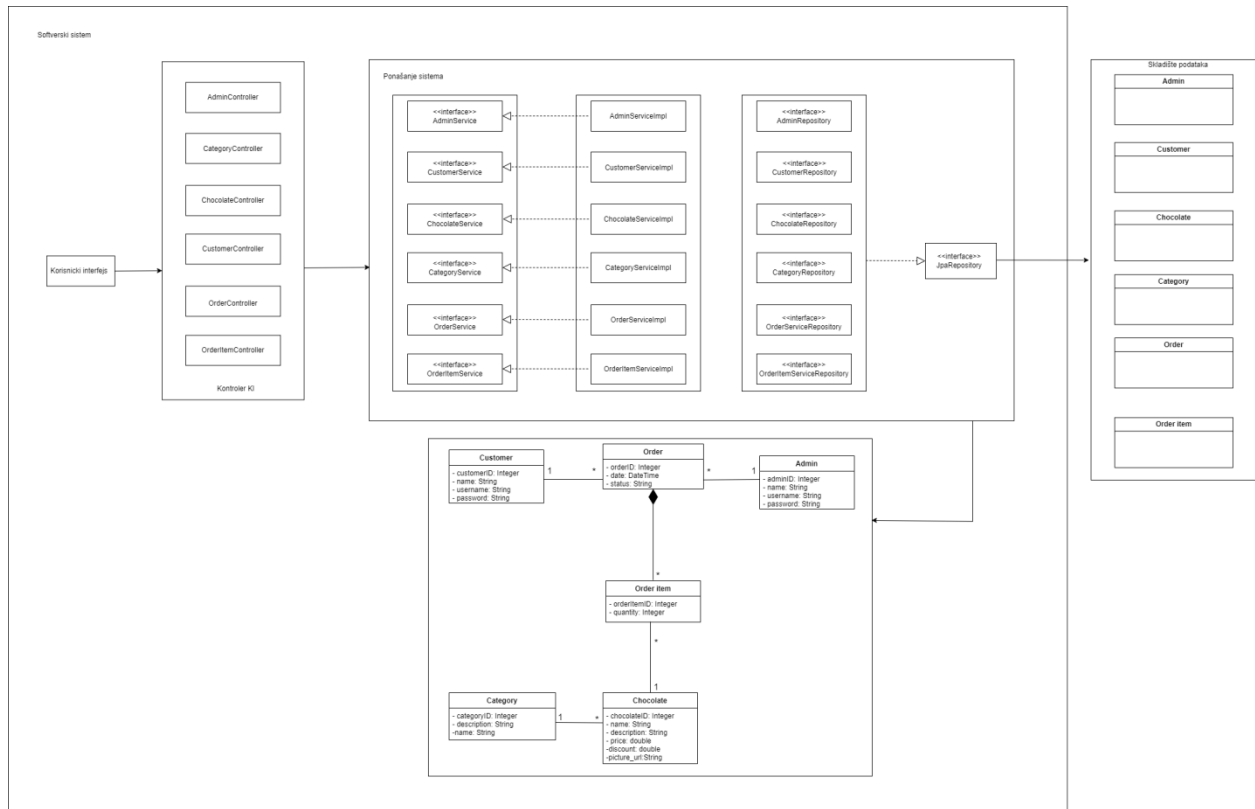
Табела Order		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Admin, Customer UPDATE RESTRICTED Admin, Customer CASCADES Order item DELETE CASCADES Order item
	OrderID	Integer	not null and > 0			
	date	Date				
	Status	String				
	CustomerID	Integer	not null and > 0			

Табела 5-Табела Order

Табела Order item		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Order, Chocolate UPDATE RESTRICTED Order, Chocolate DELETE /
	OrderItemID	Integer	not null and > 0			
	OrderID	Integer	not null and > 0			
	Quantity	Integer	not null >0			
	ChocolateID	Integer	not null and > 0			

Табела 6-Табела Order item

5. Коначна архитектура



Слика 3-Коначан дијаграм класа

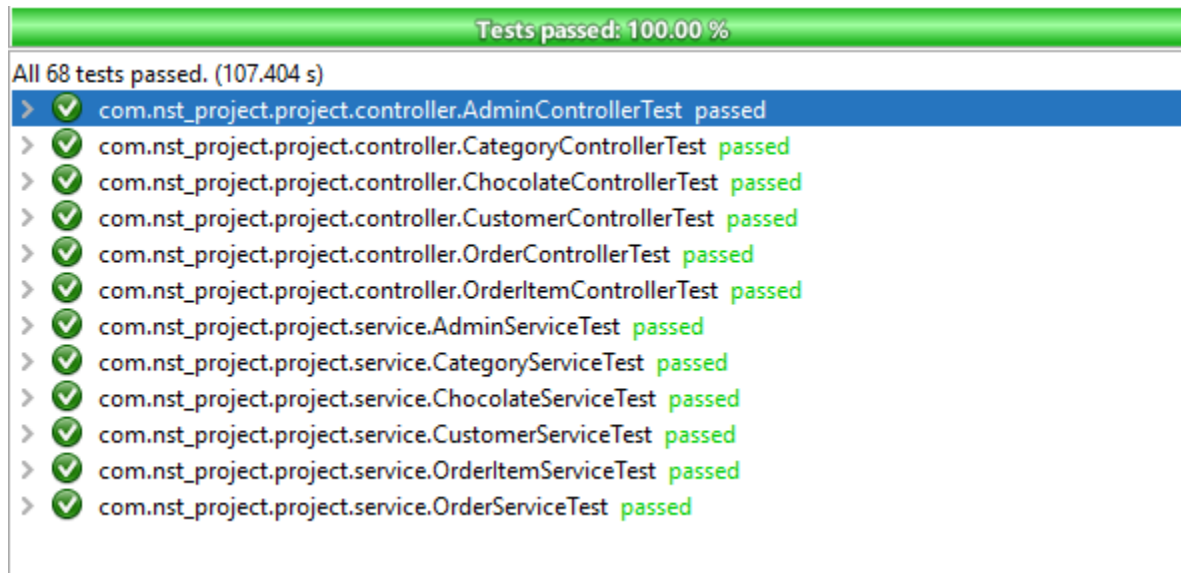
6. Тестирање софтвера

За тестирање софтвера коришћени су JUnit тестови. Да би ови тестови могли да се користе неопходно је додати нови *dependency*:

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.9.2</version>
  <scope>test</scope>
</dependency>
```

Слика 4-Dependency за JUnit тестове

Резултат тестирања је 68 успешних тестова.



За израду извештаја о успешности тестова коришћен је *plug-in* JaCoCo.

```

<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.8.8</version>
  <executions>
    <execution>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <!-- attached to Maven test phase -->
    <execution>
      <id>report</id>
      <phase>test</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

Слика 5-Додавање JaCoCo *plug-in*

Извештаји се креирају у оквиру фолдера `.\target\site\jacoco.index.html` и овом случају изгледају овако.

chocolates_back

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.nst_project.project.mapper	<div><div></div></div>	16%		n/a	12	18	20	30	12	18	0	6
com.nst_project.project.model	<div><div></div></div>	60%		n/a	35	64	48	128	35	64	0	6
com.nst_project.project.serviceImpl	<div><div></div></div>	81%	<div><div></div></div>	71%	10	43	16	98	6	36	0	6
com.nst_project.project.dtos	<div><div></div></div>	85%	<div><div></div></div>	65%	13	83	6	144	4	70	0	6
com.nst_project.project.controller	<div><div></div></div>	87%		n/a	1	32	13	102	1	32	0	6
com.nst_project.project.exception	<div><div></div></div>	48%		n/a	4	10	9	20	4	10	2	8
com.nst_project.project	<div><div></div></div>	37%		n/a	1	2	2	3	1	2	0	1
Total	549 of 1,943	71%	13 of 40	67%	76	252	114	525	63	232	2	39

Слика 6-Резултати извештаја

Од значаја су тестови пакета `com.nst_project.project.serviceImpl` и `com.nst_project.project.controller`. Црвеном бојом су означени делови који нису покривени тестовима, док су зеленом бојом означени делови које тестови покривају.

com.nst_project.project.serviceImpl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
OrderServiceImpl		74%		50%	3	9	4	18	2	8	0	1
OrderItemServiceImpl		76%		100%	1	5	2	14	1	4	0	1
CustomerServiceImpl		86%		75%	2	10	4	23	1	8	0	1
ChocolateServiceImpl		81%		50%	2	8	3	17	1	7	0	1
CategoryServiceImpl		81%		50%	2	8	3	17	1	7	0	1
AdminServiceImpl		100%		100%	0	3	0	9	0	2	0	1
Total	98 of 541	81%	4 of 14	71%	10	43	16	98	6	36	0	6

Слика 7-Извештај тестова пакета com.nst_project.project.serviceImpl

com.nst_project.project.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
OrderItemController		68%	n/a	n/a	0	4	4	12	0	4	0	1
OrderController		83%	n/a	n/a	1	7	3	19	1	7	0	1
CustomerController		92%	n/a	n/a	0	7	2	24	0	7	0	1
ChocolateController		90%	n/a	n/a	0	6	2	20	0	6	0	1
CategoryController		90%	n/a	n/a	0	6	2	20	0	6	0	1
AdminController		100%	n/a	n/a	0	2	0	7	0	2	0	1
Total	47 of 380	87%	0 of 0	n/a	1	32	13	102	1	32	0	6

Слика 8-Извештај тестова пакета com.nst_project.project.controller

7. Верзионисање кода

За верзионисање кода се користе *Git* акције које се чувају у фолдеру `.github/workflows/` са екстензијом `.yaml`. Њихова улога је да се аутоматски покрећу на неки догађај на *Git Hub*-у (*push*, *issue*, *pull*, *merge*...). У овом фолдеру се може наћи више фајлова, који се независно покрећу сваки пут када се окине догађај на који се односе. Елементи `.yaml` фајла су:

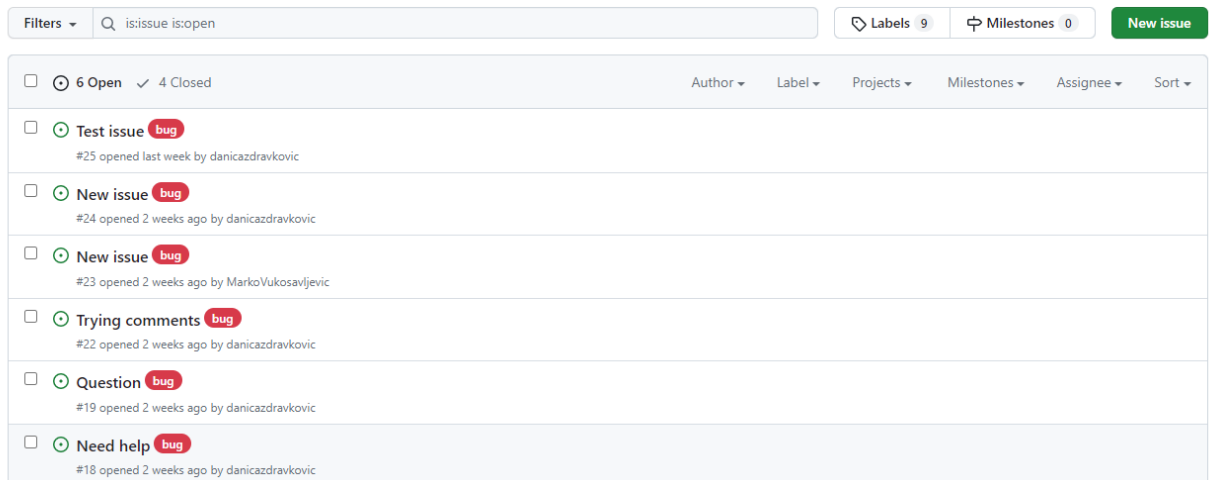
- `name`-назив акције која се покреће на *Git Hub*-у
- `on`-догађај који покреће извршавање `.yaml` фајла
- `jobs`-послови који се извршавају, може их бити више, могу бити зависни и независни. Уколико су послови зависни користи се кључна реч `needs` где се наводи други посао од кога зависи извршење тренутног посла. Послови могу да се извршавају преко акција или скрипти.
- `runs-on`-оперативни систем на коме се извршава посао, може их бити више
- `steps`-кораци посла, може их бити више
- `uses`-кључна реч која се користи када се за извршење *Git* акција користе већ унапред написане акције
- `run`-кључна реч која се користи када се за извршење *Git* акција користе скрипте.

adding labels.yml

```
.github > workflows > ! adding labels.yml
1  name: Apply label
2  on:
3    issues:
4      types: [opened]
5
6  jobs:
7    apply-label:
8      runs-on: ubuntu-latest
9      permissions:
10       issues: write-all
11      steps:
12        - uses: actions/github-script@v4
13          with:
14            script: |
15              github.rest.issues.addLabels({
16                issue_number: context.issue.number,
17                owner: context.repo.owner,
18                repo: context.repo.repo,
19                labels: ['bug']
20              })
```

Слика 9-adding labels.yml

Назив ове кације је Apply label. Користи се за догађај када се креира *issue* као догађај на Git Hub-у и када се отвори. За извршење ове акције неопходна је већ креирана акција *actions/github-script@v4* која служи за постављање лабеле на сваки креирани *issue*. Лабела која ће се поставити у овом случају је 'bug'. На Git Hub-у то изгледа овако:



Слика 10-Акција додавања лабеле

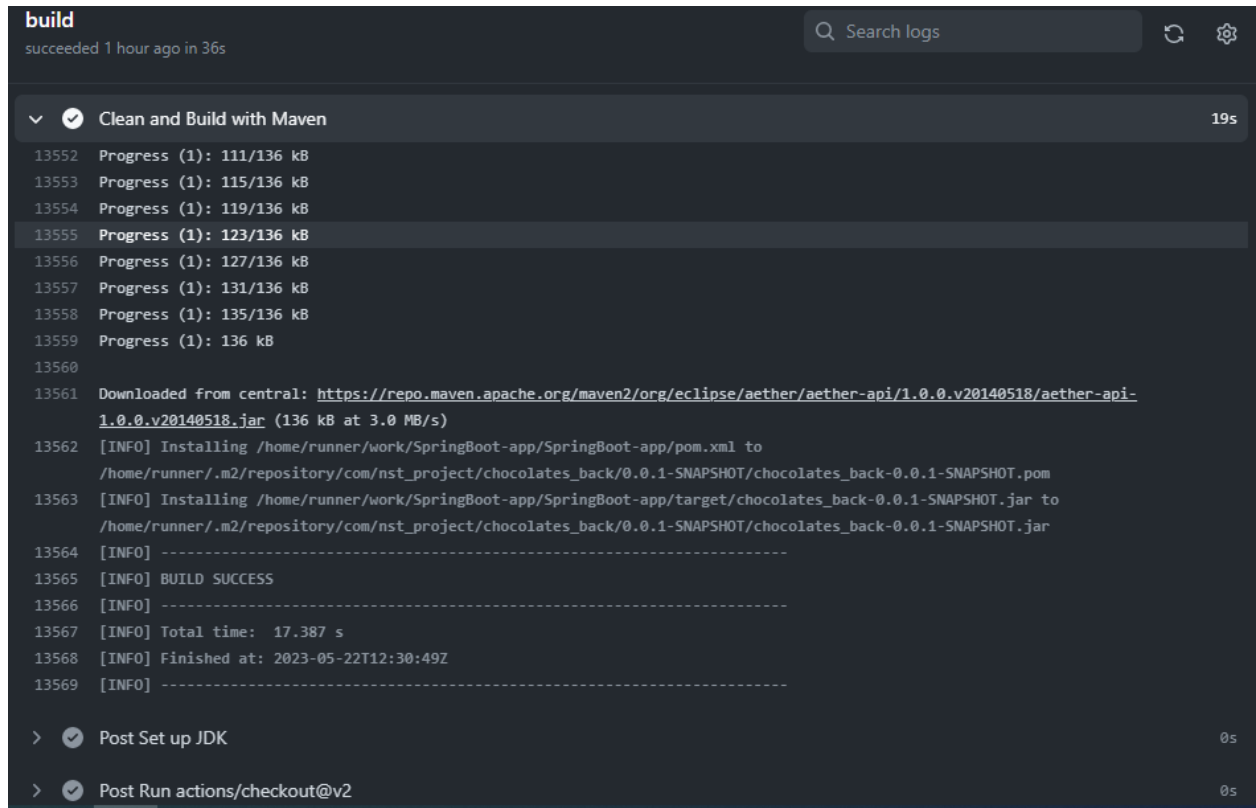
Clean_build.yml

```
.github > workflows > ! clean_build.yml
1  name: Clean and Build Maven Spring Boot Project
2
3  on: [push]
4
5  jobs:
6    build:
7      runs-on: ubuntu-latest
8
9      steps:
10     - uses: actions/checkout@v2
11
12     - name: Set up JDK
13       uses: actions/setup-java@v1
14       with:
15         java-version: '17'
16
17     - name: Clean and Build with Maven
18       run: |
19         chmod +x mvnw
20         ./mvnw clean install -DskipTests
```

Слика 11-Clean_build.yml

Назив ове кације је Clean and Build Maven Spring Boot Project. Користи се за догађај push. За извршење ове акције неопходна је већ креирана акција *actions/setup-java@v1* која служи за покретање команди *mvn clean install*. На овај начин се аутоматски при

сваком push догађају пројекат чисти(clean) и израђује(build). Резултат ове акције је успешно чишћење и израда Maven пројекта.



Слика 12-Акција clean and build

push action.yml

```
.github > workflows > ! push action.yml
1  name: Push action
2  on: [push]
3  jobs:
4  Start:
5    runs-on: ubuntu-latest
6    steps:
7      - run: echo "Event that starts this action is ${github.event_name}"
8      - run: echo "Actor's name is ${github.actor}"
9    show_message:
10     runs-on: ubuntu-latest
11     steps:
12       - name: Show message
13         run: echo "${github.event_name} event detected on ${github.ref_name} branch."
14
```

Слика 13-push action.yml

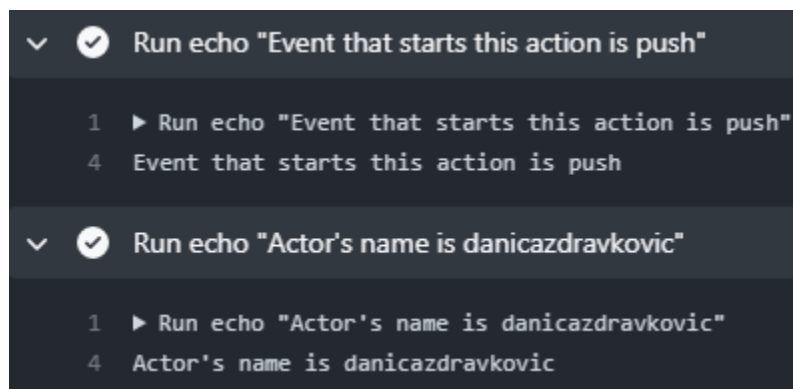
Назив ове кације је Clean Push action. Користи се за догађај push. За извршење ове акције неопходне су скрипте које покрећу echo команду која служи за испис неких контекстних варијавли као што су

`${github.event_name}` -назив догађаја који покреће акцију, у овоом случају ће то увек бити push

`${github.actor}` -корисничко име под којим сам улогована на Git Hub-у (danicazdravkovic)

`${github.ref_name}` -назив гране на којој се десио догађај

Овај фајл у себи садржи два посла, па када се покрене push догађај и активира ова акција, резултат је следећи испис.



Слика 14-Резултат посла Start

```

1 ▶ Run echo "push event detected on main branch."
4 push event detected on main branch.

```

Слика 15-Резултат посла show_message

docker.yml

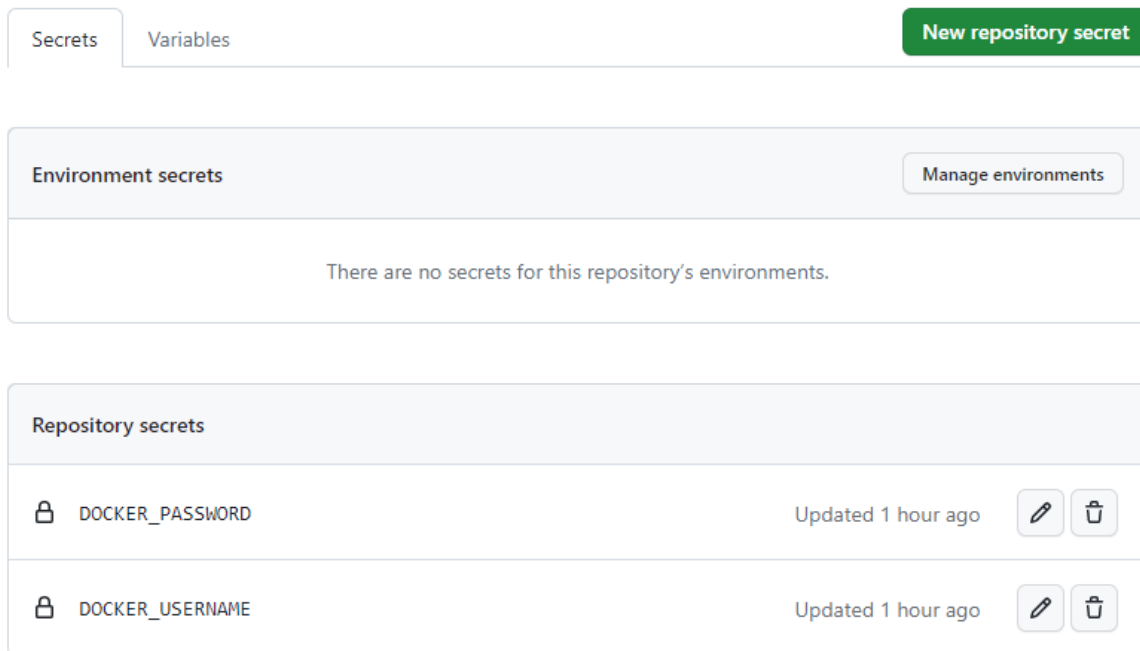
```

.github > workflows > ! docker.yml
1  name: Publish Docker image
2
3  on:
4  push:
5    branches:
6      - main
7  jobs:
8    push_to_registry:
9      name: Push Docker image to Docker Hub
10     runs-on: ubuntu-latest
11     steps:
12       - name: Check out the repo
13         uses: actions/checkout@v3
14
15       - name: Log in to Docker Hub
16         uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
17         with:
18           username: ${ secrets.DOCKER_USERNAME }
19           password: ${ secrets.DOCKER_PASSWORD }
20       - name: Build Docker image
21         run: docker build -t danicazdravkovic/empolyee-jdb:1.0 .
22
23       - name: Push Docker image
24         run: docker push danicazdravkovic/empolyee-jdb:1.0

```

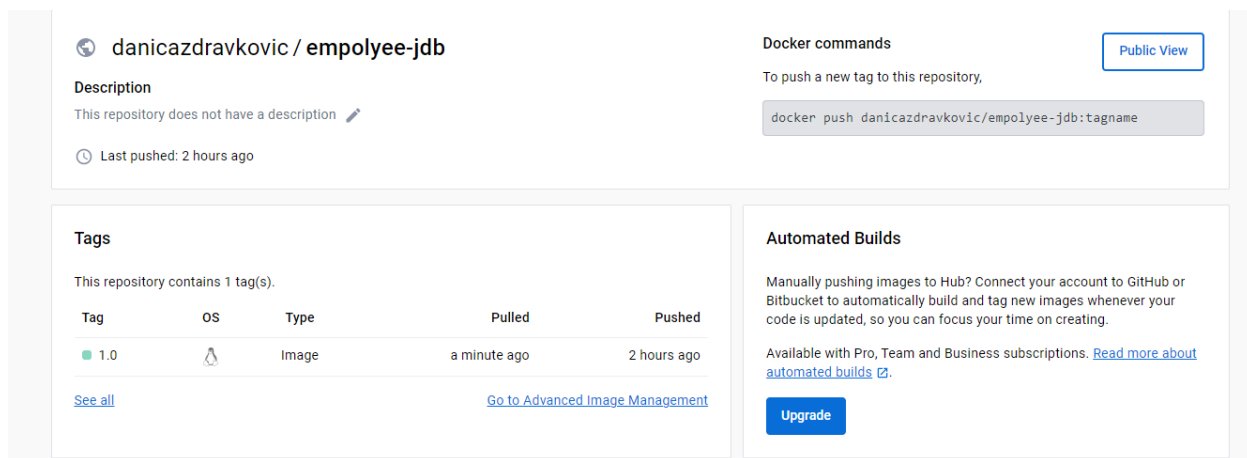
Слика 16-docker.yml

Назив ове кације је Publish docker image. Користи се за догађај *push* и то само за *main* грану. За извршење ове акције неопходна је већ креирана акција *docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a*. Циљ ове *Git Hub* акције је да сваки пут када се пројекат *push*-ује на *main* грану, да се **креира** одговарајући *Docker image* преко команде `run: docker build -t danicazdravkovic/empolyee-jdb:1.0 .`, а да се након тога он **постави** на *Docker Hub* комадном `run: docker push danicazdravkovic/empolyee-jdb:1.0`. Да би се *image* успешно поставио, неопходно је да се наведе корисничко име и таг као саставни део назива *image*-а. У ову сврху било је неопходно направити скривене *Git* варијабле које представљају корисничко име и лозинку за *Docker Hub* (`${ secrets.DOCKER_USERNAME }`), (`${ secrets.DOCKER_PASSWORD }`).



Слика 17-Креирање secrets варијабли које се користе као корисничко име и лозинка за логовање на Docker Hub

Резултат ове акције након *push* догађаја на *main* грани је постаљен *image* на *Docker Hub*-у:



Слика 18-Аутоматски постављен image на Docker Hub-у

8. Распоређивање софтвера

Доступност апликације проверавамо преко Spring Boot Actuator-а преко endpoint-а /actuator/health. За омогућавање рада Актуатора, неопходно је додати следећи dependency.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Слика 19-Dependency неопходан за Актуатор

Уколико је апликација доступна резултат уношења овог endpoint-а је:

The screenshot shows a REST client interface with the following details:

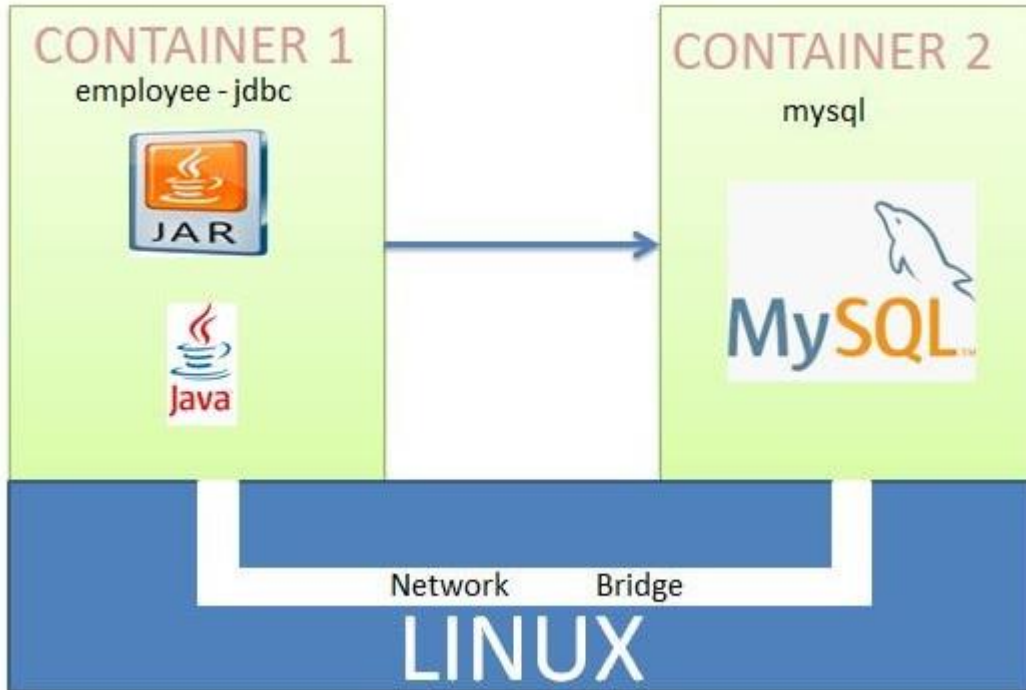
- Method:** GET
- URL:** http://localhost:8080/actuator/health
- Status:** 200 OK
- Time:** 317 ms
- Size:** 207 B
- Response:** {"status": "UP"}

KEY	VALUE	DESCRIPTION
Key	Value	Description

Слика 20-Доступност апликације

9. Паковање апликације

За паковање апликације коришћен је Docker. Docker је апликација која се састоји из контејнера и image-а. Контејнер је image који се извршава. Сваки контејнер представља изоловано окружење, па њихова међусобна комуникација није могућа. У те сврхе се креира мрежа (docker network).



Слика 21-Изглед Docker мреже

Мрежа може да се користи тако што ће се у њу појединачно убацити један контејнер за базу података(mysql) и један за саму апликацију(employee-jdbc-container).

Креирање employee-mysql мреже:

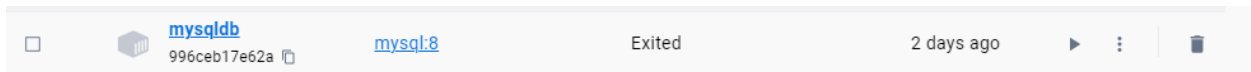
```
docker network create employee-mysql
```

Креирање и покретање mysqlpdb контејнера:

```
docker container run --name mysqlpdb --network employee-mysql -e  
MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=bootdb -d mysql:81
```

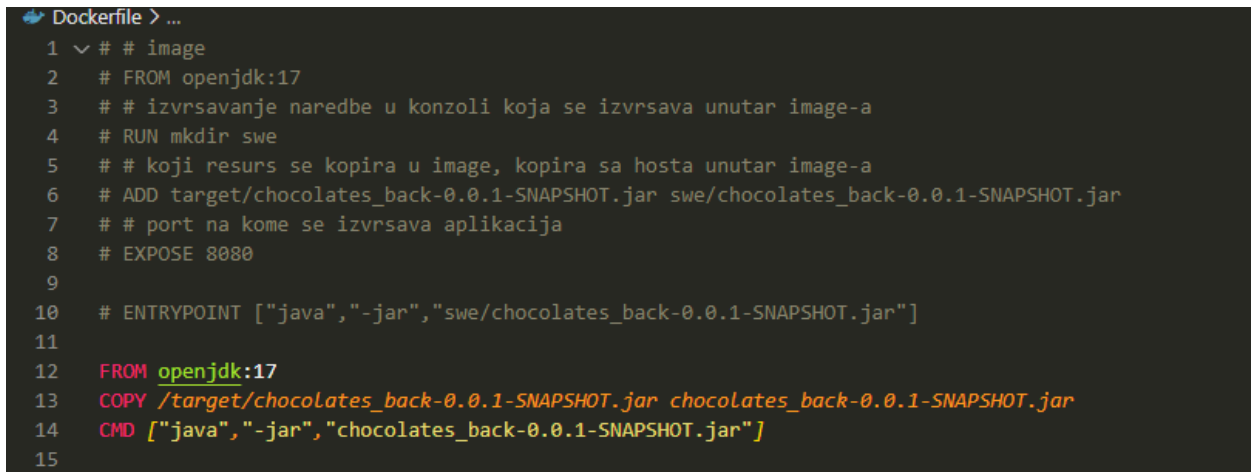
¹ Након команде name наводи се назив контејнера. Након команде --network наводи се мрежа којој ће контејнер припадати. Не мора да припада ниједној мрежи, али у нашем случају да би контејнери могли да комуницирају морају припадати истој мрежи. MYSQL_ROOT_PASSWORD вредност која се користе као

На овај начин је креиран mysql контејнер.



Креирање employee-jdbc-container контејнера:

За креирање овог контејнера неопходно је прво креирати Docker фајл.



Слика 22-Docker фајл

Верзија јаве која се користи је 17. Команда COPY служи за копирање .jar фајла из локалног фолдера у Docker контејнер. Команда CMD дефинише који фајл треба да се изврши када се креира контејнер, а то је управо .jar који ће се копирати у Docker контејнер.

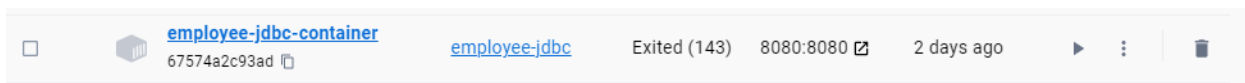
Након команде mvn clean install, где се креира .jar фајл /target/chocolates_back-0.0.1-SNAPSHOT.jar у фолдеру target, неопходно је покренути креирање image-а комадном *docker image build -t employee-jdbc*. Креирање и покретање инстанце image-а остварује се следећом наредбом:

```
docker container run --network employee-mysql --name employee-jdbc-container -p 8080:8080 -d employee-jdbc2.
```

На овај начин креиран је employee-jdbc-container

лозинка за повезивање на mysql базу. MYSQL_DATABASE је назив базе која ће се креирати у оквиру контејнера где ће се креирати све неопходне табеле за Spring Boot апликацију. mysql:8 је назив image-а који користимо за креирање овог контејнера.

² Мрежа у којој је контејнер креиран се наводи након наредбе --network. Назив контејнера се наводи након наредбе --name. Повезивање портова (Port binding) се остварује наредном -p 8080:8080 -d. На крају се наводи image на основу ког се контејнер креира employee-jdbc.



Како се ова два контејнера налазе у истој мрежи они сада могу и да комуницирају.

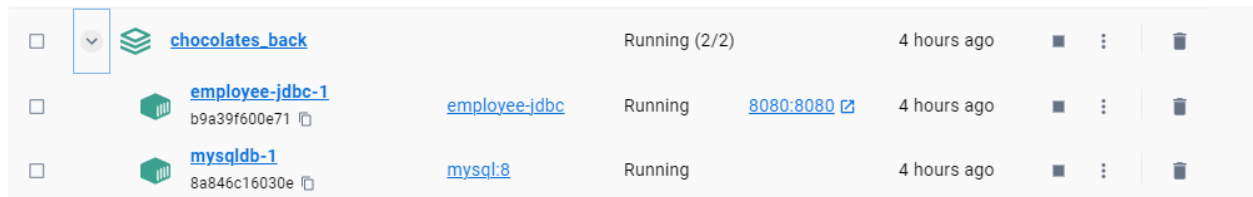
Исти овај ефекат ће се постићи креирањем docker compose фајла.













```
docker-compose.yml
27  version: "3"
28  services:
29    employee-jdbc:
30      image: employee-jdbc
31      ports:
32        - "8080:8080"
33      networks:
34        - employee-mysql
35      depends_on:
36        - mysql
37      environment:
38        - SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/bootdb
39        - SPRING_DATASOURCE_USERNAME=root
40        - SPRING_DATASOURCE_PASSWORD=root
41
42
43    mysql:
44      image: mysql:8
45      networks:
46        - employee-mysql
47      environment:
48        - MYSQL_ROOT_PASSWORD=root
49        - MYSQL_DATABASE=bootdb
50
51  networks:
52    employee-mysql:
53
```

Слика 23-Docker compose фајл

У оквиру services команде се наводе сервиси-контејнери које је неопходно креирати то су mysql и employee-jdbc. За сваки сервис-контејнер се командом image наводи image који се користи за њихово креирање, а то су employee-jdbc и mysql:8. Port командом се

назначава на ком порту се подиже контејнер у оквиру Docker-а. Networks означава мрежу којој се контејнер придружује. Depends_on означава контејнер са којим је тренутни контејнер повезан. Environment су варијабле које су неопходне да се контејнер креира и покрене. Крајња команда networks означава мреже које се користе у овом docker compose фајлу. На команду docker compose up у локалном фолдеру пројекта, покреће се овај фајл и креирају ова два контејнера у оквиру мреже employee-mysql и повезују се.



<input type="checkbox"/>		chocolates_back		Running (2/2)		4 hours ago			
<input type="checkbox"/>		employee-jdbc-1 b9a39f600e71	employee-jdbc	Running	8080:8080	4 hours ago			
<input type="checkbox"/>		mysql-1 8a846c16030e	mysql:8	Running		4 hours ago			

Слика 24-Резултат docker compose фајла

10. Закључак

Аутоматизација развоја софтвера омогућава олакшано и брзо креирање и распоређивање неопходних фајлова. Креирањем тестова омогућава се посебан *test driven* приступ развоја софтвера где уопште није неопходно покретати само апликацију, већ покретањем тестова вршити њено кориговање све до финалне фазе.

Git акције обезбеђују аутоматско покретање кода који ће се извршити сваки пут када се одговарајући догађај окине и на тај начин одмеђује посредство човека у обављању тих послова.

Docker помаже развој софтвера тиме што целу апликацију, са свим њеним конфигурационим фајловима, оперативним системом и неопходним фајловима пакује у један контејнер и спрема је за коришћење. Docker Hub је користан јер обезбеђује размену креираних image-а и на тај начин повезује програмере широм света.