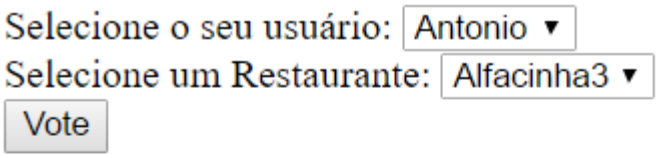
HungerCracy – by Antonio Souto (antoniosouto@gmail.com)

Hungercracy is a fully functional implementation of the 3 US (see Appendix) plus data persistence.

- To build: gradllew build
- To run: java -jar build/libs/hc-0-1-0.jar
- Open http://localhost:8080 and click on the link, or go directly to http://localhost:8080/voting

If you are running it before 11:30 AM you will get:



Nome do Restaurante	Votos	Eleito
Alfacinha3	4	false
Alfacinha2	3	false
Alfacinha	2	false

Technologies used:

- Java, including here and there some new cool Java 8 stuff, like lambda expressions and the great new implementation of dates handling of java.tim.LocalDate
- MVC: Spring Boot + ThymeLeaf
- View: HTML JavaScript + JQuery
- Repository: Git
- Micro Services: Spring Boot
- Data persistence: JPA + H2 database
- Tests: Junit and Mockito for unit tests and for test the application with Spring MVC including the Web Layer.
- Build: Gradle

Some highlights about this implementation:

- **Using Thymeleaf as a server side template engine along with Spring enables a very clear separation between Control and View, and the model implanted over HTML is very straightforward to use.**
- **However, as the template engine does not exclude the need for some JavaScript, maybe it should be better to switch for good to JavaScript, with a RESTful service talking to a jQuery client through JSON, and enjoy the gain in liberty with a much less restrictive framework. Other benefit is that JavaScript allows you to run away from the flickering of full HTML pages swap of the old plain GET method receiving a**

complete page with the new info. It may look like that heavily using JavaScript transfers part of the control to the View, blurring the MVC separation. However, you still keep things separating, relying on JavaScript to render your page and to do some data manipulation while heavily relying on he backend for heavy computation and database queries, and transferring only the data trough web.

Improvements Roadmap

- **Raise the unit test coverage (There is plenty of room to increase it, but I run out of time) (high priority)**
- **Switch to REST/JSON/jQuery (medium priority)**
- **Persist data in disk to prevent losing data in case of a crash (low priority)**
- **Login (medium priority)**
- **Notify users about the voting progress and results (high priority)**
- **Enhance Ui Design – add some good CSS and JavaScript to make it nice and easy to use (medium)**
- **Handle some special cases like duplicate entries for User and Restaurant in the database**
- **Localization – replace hardcoded messages in Portuguese for message keys (low)**

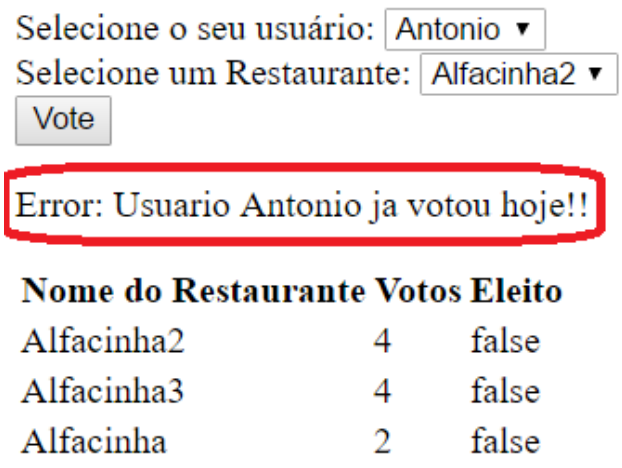
DEMO

US 1 Acceptance Criteria: Each user can only vote in one restaurant per day.

Note: in the Agile sense of delivering value to the user instead of a full stack of background work, there is no Login, and each user is in charge of selecting his/her name to vote. The PO has probably considered much better to end the sprint with a voting feature then with a login to a nonfunctional system.

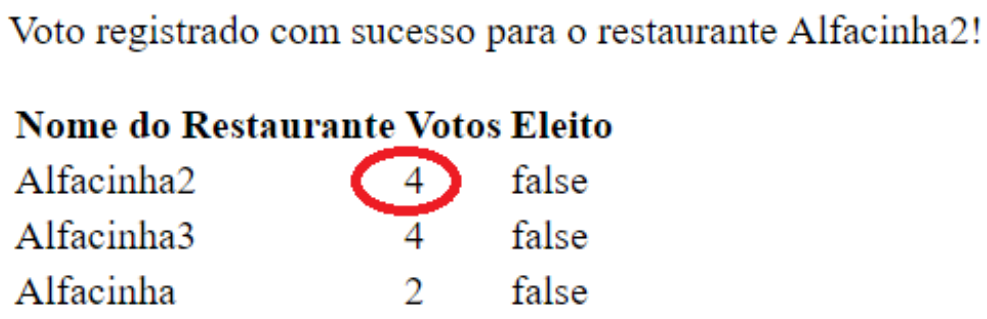
The preloaded users Antonio, Marcia and Vinicius have voted for the last time respectively today, yesterday and never.

If you try to Vote using user Antonio, you get the error below:

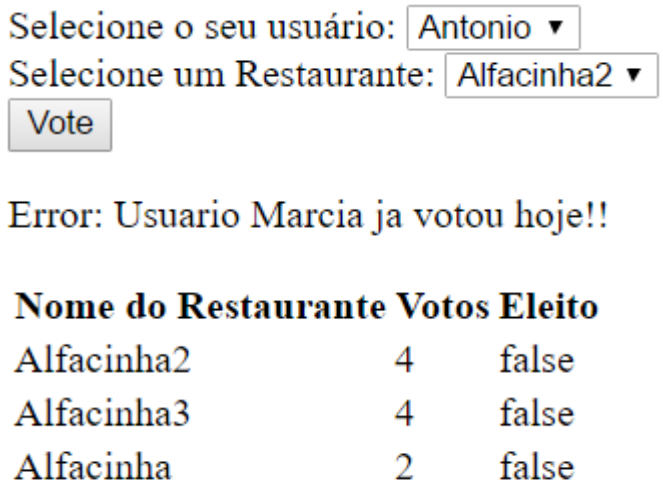


Both Marcia and Vinicius will be able to vote once and get the success message below, including the list of restaurants sorted by their

votes:



But if they try it again they will get the same error message that Antonio got:



US 2 Acceptance criteria: The same restaurant cannot be chosen more than once a week.

To implement this the system simply filter out the restaurants that have been chosen the last time during the current week.

For this demo we preloaded four restaurants:

```
: Restaurants found with findAll():
: -------------------------------
: [id=1, name=Churrascaria Sangue na Blusa, votes=125, won=false, last time won=2017-02-20]
: [id=2, name=Alfacinha, votes=2, won=false, last time won=2017-02-19]
: [id=3, name=Alfacinha2, votes=3, won=false, last time won=2017-02-16]
: [id=4, name=Alfacinha3, votes=4, won=false, last time won=2017-02-13]
```

But the system only list the of them, the Alfacinha* ones, which where not chosen by vote during the course of the current week.

See hungercracy.RestaurantServiceImpl.getAllRestaurantsNotYetChoosenThisWeekSortedByNameAndByVotes()

US 3 Acceptance criteria: Show the result of the voting before noon.

For this I created two concepts: Open and Closed Voting.

The system opens the voting every week day at 8:30 AM, and closes the voting at 11:30 AM.

Close Voting:

Controller:

- Every weekday at 11:30.
- Set the winner (flag it and update the lastDayWon timestamp.
- Uses Spring Boot Scheduler
    - See hungercracy.ScheduledTasks class.

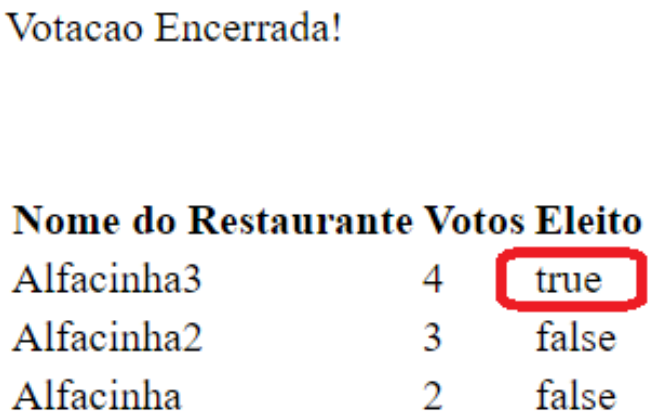
Model:

- Set attribute flag

View:

- Hide voting form and show table with all restaurants and indicate the winner.

For demo and testing purpose, type http://localhost:8080/voting.html?test=closeVoting and the closeVoting Method will be manually triggered disregarding the scheduled time.



Open Voting

Controller:

- Every weekday at 08:30
- Clear winner flags and votes
- Uses Spring Boot Scheduler
    - See hungercracy.ScheduledTasks class.

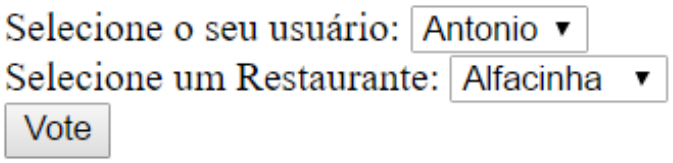
Model

- unset attribute flag

View

- Show Voting form and table with all restaurants excluding the ones that were chosen during the current week voting.

Note: the restaurant chosen on the last voting is fielterd out of the dropbbpx.



Nome do Restaurante	Votos	Eleito
Alfacinha	0	false
Alfacinha2	0	false

APPENDIX

Introdução

Os times da DBServer enfrentam um grande problema. Como eles são muito democráticos, todos os dias eles gastam 30 minutos decidindo onde eles irão almoçar.

Vamos fazer um pequeno sistema que auxilie essa tomada de decisão!

Instruções:

- O sistema deve implementar todas as histórias descritas nesse documento.
- Desenvolva o código na linguagem e arquitetura em que você se sentir mais à vontade.
- Não precisa implementar a persistência em banco de dados. Você pode retornar dados fakes.
- Todas as histórias devem ser implementadas com um nível aceitável de testes automatizados.

- Nos mande o link para um arquivo zip contendo todo o código. Pode ser via DropBox ou algo do gênero. Se ao invés disso você preferir enviar o link de um repositório público no GitHub ou BitBucket, também serve ;)

- Crie um readme que inclua:
    - O que vale destacar no código implementado?
    - O que poderia ser feito para melhorar o sistema?
    - Algo a mais que você tenha a dizer ☺

Histórias

História 1

Eu como profissional faminto
Quero votar no meu restaurante favorito
Para que eu consiga democraticamente levar meus colegas a comer onde eu gosto.

Critério de Aceitação

- Um professional só pode votar em um restaurante por dia.

História 2

Eu como facilitador do processo de votação
Quero que um restaurante não possa ser repetido durante a semana
Para que não precise ouvir reclamações infinitas!

Critério de Aceitação

- O mesmo restaurante não pode ser escolhido mais de uma vez durante a semana.

História 3

Eu como profissional faminto
Quero saber antes do meio dia qual foi o restaurante escolhido
Para que eu possa me despir de preconceitos e preparar o psicológico.

Critério de Aceitação

- Mostrar de alguma forma o resultado da votação.

Obrigado por dedicar parte do seu tempo! Esperamos ansiosamente a sua resposta!