PA1489 Övningsuppgifter

Mikael Svahnberg*

2024-04-19

1 Introduktion

Det här dokumentet sammanfattar de *icke betygsgivande* övningsuppgifter som introduceras under varje föreläsning.

2 Kom Igång med Git

2.1 Registrera ett Konto

- Måste tyvärr börja med att registrera ett konto på någon server.
  - `https://github.com/signup` eller `https://education.github.com/pack`
  - `https://gitlab.com/users/sign_up`
  - `https://www.atlassian.com/software/bitbucket/bundle`
  - `https://codeberg.org/`
- Github är fortfarande väldigt stort för open source-projekt
  - Lite i blåsväder för hur de använder koden som du laddar upp dit
- Många migrerade över till Gitlab när Microsoft köpte Github
- Atlassian och Bitbucket har bra integration med deras övriga produkter.
  - Brukade vara väldigt generösa för studenter och universitet (numera vet jag inte)
- Codeberg.org är specifikt fokuserat på open-source-projekt

2.2 Skapa och klona ett Repository

- Lättast att börja i webgränssnittet
- Döp projektet till något kreativt, t.ex. `gitexempel`
- När du är klar bör du kunna hitta en länk, t.ex. under `<> Code` som du kan använda för att klona projektet
  - till exempel: `git clone https://codeberg.org/mickesv/gitex.git`
  - Det här sätter up `remote/origin` mm. åt dig.

*Mikael.Svahnberg@bth.se

2.3 Skapa lite git-historia

1. Skapa några filer
2. Lägg till dem till stashen och committa
3. Ändra någon av filerna; lägg till och committa igen
4. Upprepa några gånger
5. Skapa en branch
6. Skapa några filer, lägg till och committa.
7. Redigera någon av dina första filer, committa.
8. Kolla loggen
9. Kolla status
10. Pusha till servern
11. Kolla status

2.4 Forka en kollegas repository

1. Leta rätt på en kollegas konto (på samma server)
2. Välj ett repository och forka det (Lämpligen exempel-kontot som ni nyss skapade)
3. Klona ner det till din dator och skapa lite mer git-historia
4. När du har pushat allt till din fork, skapa en `pull request` hos deras repository (via webben)

2.5 Hantera en pull request

När din kollega har skapat en pull request mot ditt repo, hantera den:

- Inspektera commit för commit vad som är ändrat
- Går den att merga automatiskt? Det borde stå någonstans.
- Skapa en merge commit.

Skapa några fler commits i era respektive forkar

- Skapa en ny pull request
- Den här gången skall ni *neka* pull requesten.

2.6 Fler deltagare i samma projekt

- Dela in er i grupper om ca 5 personer
- Välj en kollegas repository
- Gå till `Settings/Collaborators` och lägg till fler av er på samma projekt.
- Klona repot

Nu får ni bara arbeta i en viss fil `charlie-foxtrot.txt`

- Ni får skriva ny text
- Ni får redigera texten som finns där
- Ni får stoppa in text: mellan två rader, och mitt i en rad.
- Ni får ta bort text

Committa regelbundet (max 2-3 ändringar per commit) Pusha efter varje commit

- Ni kan behöva göra en `fetch/merge` för att få göra en `push`

hantera merge-konflikterna
Diskutera i små grupper: Hur skall ni göra för att få färre konflikter?

3 Testning och Debugging

3.1 Introduktion till SorterTool

- `https://codeberg.org/mickesv/SorterTool.git`
- *SorterTool* implementerar och testar ett par olika sorteringsalgoritmer.
- Det finns många olika sätt att sortera listor på som är olika snabba.
  - Sök på "Sorting out Sorting" för en gammal film (30 min) från 1980 som vi tvingades titta på i Datastruktur-kursen.
  - "Big-O" notation beskriver komplexiteten hos algoritmer.
- Exempel:
  - Insertion Sort: `O(n²)` Flytta element ur vägen och stoppa in elementet på rätt plats
  - Selection Sort: `O(n²)` Hitta det minsta värdet och stoppa in det först; börja om på element 2
  - Merge Sort: `O(n*log n)` Se till att varje par är ordnade, kombinera par `n` med `n+1`; upprepa.
  - QuickSort: `O(n*log n)`
    1. Välj ett element i mitten
    2. se till att alla till vänster är mindre och alla till höger större
    3. upprepa för vänster och höger.
  - Bubble Sort: `O(n²)` jämför varje element med alla andra och byt plats på alla element som inte redan är ordnade.

3.2 Kom igång med övningen

1. Klona SorterTool till din dator: `https://codeberg.org/mickesv/SorterTool.git`
2. Öppna projektet i din IDE (IntelliJ IDEA). Studera följande filer:
   - `src/Main.java` för att snabbt kunna köra programmet
   - `src/Sorter.java` implementerar de olika sorteringsalgoritmerna.
   - `Tests/SorterTest.java` testar `Sorter`.
3. Kör alla tester. Vad händer?
4. Kör specifikt testet för `bubbleSort()`

3.3 Skaffa mer information

Läs meddelandet
Nej, allvarligt! **Läs meddelandet!**

- Vad kan `AssertionFailedError` betyda?
- I vilken fil och på vilken rad händer det?
- Vad står det på den raden?
  - Räcker detta för att förstå vad som gick fel?
  - Hur kan du ta reda på mer informaiton?

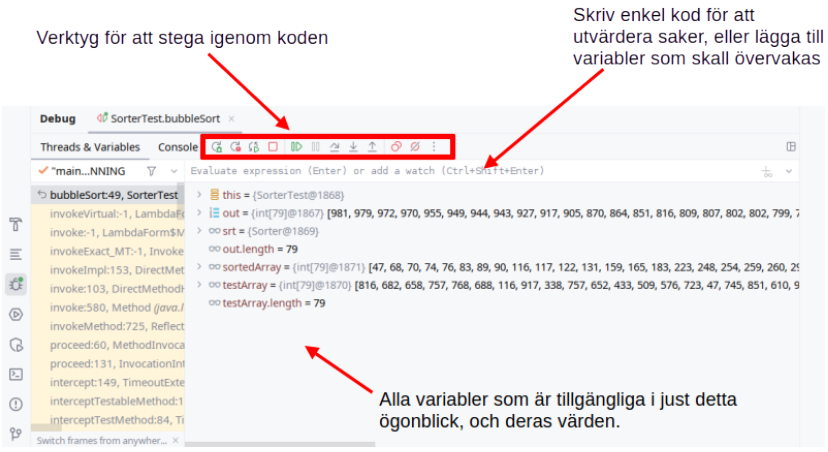
3.4 Använd Debuggern

1. I filen `SorterTest.java`, klicka på radnummret `49`; Det borde bytas till en liten stopp-skylt
2. Kör nu testet för `bubbleSort()` i "Debug"-läge.

Vad ser du nu i:

- kodfönstret?
- fönstret nedanför koden?

3.5 Debug-fönstret



- Här kan vi se varför testet misslyckas, men inte vad i koden som orsakade felet.
- *Ledtråd:* Titta på arrayen `out` och alla värden där. Kan du se något mönster?

3.6 Stega genom koden

1. Flytta break-punkten fråmn rad `49` till rad `47`.
2. Kör testet för `bubbleSort()` igen i Debug-läge (Starta om testet när du får frågan).

Viktiga verktyg för att stega genom koden:

Continue/Resume Kör på till nästa breakpoint

Step over Kör nästa instruktion, och stanna när du kommer tillbaka

Step in Följ med in i nästa instruktion (oftast in i en metod)

Step out Kör klart metoden du är i nu, och stanna när du kommer tillbaka.

Använd **Step in** nu för att stega in i anropet till `srt.bubbleSort()` .

- Notera att variabelfönstret ändras. Vad ser du där nu? Varför?

Stega ett par varv i den inre for-loopen.

- raderna med `for` och `if` körs i varje varv.
- När fortsätter koden in i if-satsen?
  - Vid vilka värden på `out[outer]` respektive `out[inner]` ?

Stämmer detta? (Det gör det inte. Vad borde hända?)
Rätta till felet och fortsätt debugga.

- Som du märker verkar inte ändringen fungera.
- Du måste *starta om* testet för att det skall ta effekt.

3.7 Kör om alla tester

- När du nu har fixat felet så borde alla tester bli gröna.
- Varför blir både `sort()` och `bubbleSort()` gröna? Du har ju bara fixat `bubbleSort()`...

3.8 Lek vidare på egen hand

- Kan du använda debuggern för att förstå hur sorteringsalgoritmerna fungerar?
- Skriv din egen sorteringsalgoritm och testa den.
- Just nu finns det ett test för varje metod. Kan du ha fler tester för varje metod?
  - Vilka fler tester kan vara relevanta?

4 Dokumentation

4.1 Introduktion till JavaPonies

- *Desktop Ponies* är en urgammal mono-applikation (liknar Visual Basic) som låter My Little Ponies springa runt på skärmen.
- *Java Ponies* är "min" version av detta program
  - `https://codeberg.org/mickesv/JavaPonies.git`
  - Varning:
    - Det är långt ifrån färdigt
    - Det är inte fulständigt dokumenterat
    - Det är långsamt och säkert buggigt
  - Men:
    - PONIES

4.2 Kom igång med övningen

1. Klona projektet till din dator: `https://codeberg.org/mickesv/JavaPonies.git`
2. Öppna projektet i din IDE och studera programmet så att du förstår vad det gör.
   - `src/JavaPonies.java` startar programmet
   - `src/model/Pony.java` Implementerar en klass som instantieras för varje ponny
   - `src/model/PonyBehaviour.java` representerar ett enskilt beteende som en ponny kan ha
   - `src/view/PonyWindow.java` Sköter visning och uppdatering av en viss ponny som är aktiv på skärmen.

4.3 Skapa dokumentationen

- Från en terminal: `javadoc src/*.java src/model/*.java src/view/*.java -d doc`
- Från IntelliJ: `Tools/Generate JavaDoc`, fyll i att dokumentationen skall hamna i katalogen `doc` (Resultatet öppnas i din webläsare)

Inspektera dokumentationen:

- Jämför med vad du ser i java-filerna
- Vad finns med? Vad finns inte med?
- När du skapade dokumentationen fick du många varningar. Vad beror de på?

4.4 Uppdatera JavaDoc

Filen `src/model/PonyBehaviour.java` saknar JavaDoc-kommentarer.

1. Skriv dessa kommentarer så att du inte längre får några javadoc-varningar från `PonyBehaviour.java`.
2. Ökade detta användbarheten av dokumentationen? Varför / Varför inte?
3. Ökade detta läsbarheten av koden? Varför/Varför inte?

4.5 Skapa en Issue

Notera:

- Om du har ett konto på Codeberg.org kan du skapa en issue direkt mot JavaPonies-projektet
- Om du inte har eller inte vill ha ett konto kan du skriva din issue direkt i en textfil.

Att Göra:

1. Hitta någonting att åtgärda i projektet. Det kan vara en:
    - Bug – något som inte fungerar som förväntat
    - Enhancement – en ny feature
2. Skriv din issue. Den skall innehålla:
    - Kort men innehållsrik titel
    - Beskrivande text
    - Steg för att provocera fram buggen, eller steg till där förbättringsförslaget skulle kunna vara lämpligt
    - Förväntat resultat
    - Faktiskt resultat
    - Övrig information (om relevant)

4.6 Förbättra en metod

1. Välj en metod eller attribyt som du tycker är otydlig och genomför en *Refactoring* så att den blir tydligare.
    - IntelliJ har en hel meny för Refactor; utforska den för att se vad som finns där och hur det fungerar

Fundera på:

- Vilket stöd har du av din IDE för Refactoring?
- Hjälper detta stödet?
- Hur vet du vad som är en bra refactoring?

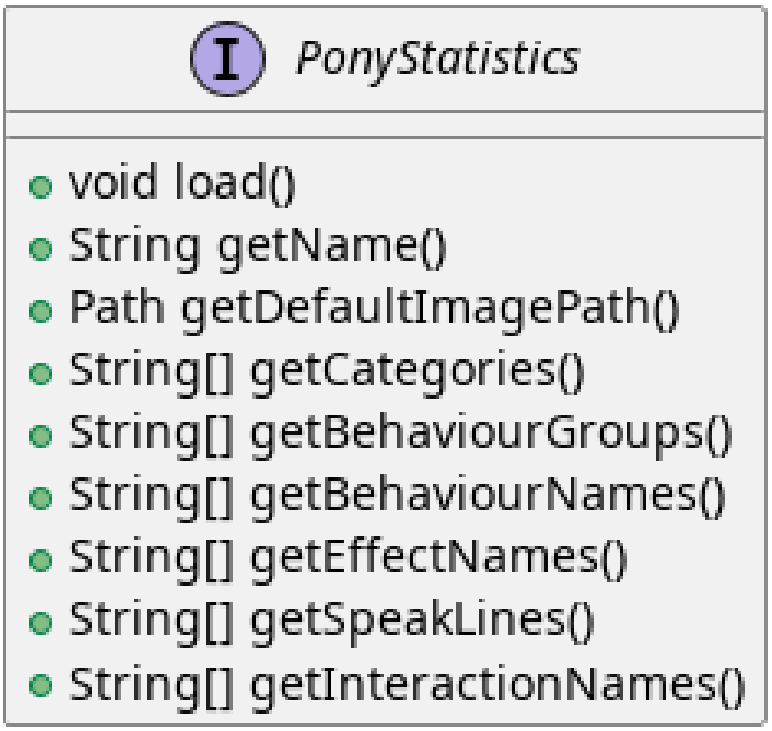
5 Implementation i Java

5.1 Mera Ponies

- Vi fortsätter med JavaPonies.
- MLP-fansen har beställt ett utökat gränssnitt där man kan få reda på mer data om varje Ponny.
- Vi skall dessutom se till att påbörja implementationen av *Interactions*, att en ponny byter beteende för att de är nära någon annan.

5.2 MLP-Data

1. Skriv ett interface `src/model/PonyStatistics.java` enligt nedan.
2. Se till att `model.Pony` implementerar detta interface. Notera att
   - Några metoder redan finns, men kan behöva utökas
   - Några metoder kommer kanske anropas flera gånger; särskilt `load()` kommer behöva ta hänsyn till detta.
   - Nya klasser kan behöva skapas t.ex. för att innehålla en Interaction
   - Några av get-metoderna kan behöva iterera över en `ArrayList<>` av t.ex. `Behaviours` för att plocka fram deras namn och spara i en String-array.
   - Vi inte har något sätt att använda dessa metoder ännu. *Skriv gärna enhetstester i stället.*



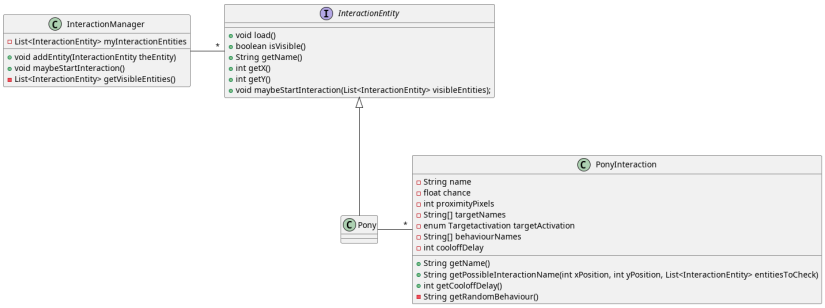
5.3 Interactions

- För att en Ponny skall kunna interagera med en annan Ponny, så krävs det att de vet att de står i närheten av varandra.
- Detta kan inte en enskild Ponny veta. Vilka andra alternativ har vi?
  - `view.PonyWindow` vet var en viss Ponny är (genom att fråga den), men inte de andra.

- `view.MainWindow` har, när den skapat `PonyCard` för varje `Pony`, inte ens koll på vilka Ponnys som finns.
- `model.PonyContainer` kan veta. Men då får den två ansvarsområden: Underhålla samlingen av Ponnys *och* sköta interaktioner.

$\sum$

1. Vi behöver skapa en ny klass `model.InteractionManager`, som har en samling med `InteractionEntity`
   - Metoden `maybeStartInteraction()` behöver anropas regelbundet
2. Vi behöver skapa ett interface `model.InteractionEntity` som `Pony` implementerar
   - Särskilt viktig är metoden `maybeStartInteraction()`
3. Vi behöver skapa en klass `model.PonyInteraction` som representerar en specifik möjlig interaktion.
4. Klassen `JavaPonies` behöver "sätta igång" ett `InteractionManager` - objekt.
5. Klassen `view.PonyWindow` behöver samarbeta med `model.Pony` så att `model.Pony` vet om den är synlig eller inte.



6 Grafiska Gränssnitt i Java

6.1 Mera Pony-Statistik

- I projektet JavaPonies finns en branch `PonyStatistics` där interfacet från tidigare föreläsningar implementeras.
- Checka ut JavaPonies på ett nytt ställe (om du vill spara din implementation) och byt branch:
  - `git clone https://codeberg.org/mickesv/JavaPonies.git`
  - `cd JavaPonies && git checkout PonyStatistics`
- Kontrollera vad som ändrats: `git diff origin/main`

6.2 En Ny Main

1. Skriv en ny klass `JavaPonyStatistics extends JavaPonies` (att ärva från JavaPonies gör att du kan spara mycket av uppstarten från JavaPonies).

2. Skriv en ny `main()` - funktion i `JavaPonyStatistics` :

```
public static void main(String[] args) {
  JavaPonyStatistics ps = new JavaPonyStatistics();
  ps.printStatistics();
}
```

3. Implementera metoden ~JavaPonyStatistics.printStatistics() så att den:

   - itererar över alla Ponies (du hittar dem via `myPonies.findAll()`), och
   - skriver ut Categories, Behaviour Groups, Behaviours, Effects, Interactions, och Speakig Lines:

   Printing Statistics for Apple Bloom
   Categories:
   Behaviour Groups:
   Behaviours: stand, walk, $\text{follow}_{\text{aj}}$, $\text{spin}_{\text{merightround}}$, workout, aww, CMC, dance
   Effects:
   Interactions:
   Speaking lines:
   - CUTIE MARK CRUSADER DESKTOP PONIES!!!
   - Did I get my cutie mark? Did I? Did I!?
   - Scoot-Scootalooo!
   - Aww!
   - Aren't you gonna stay for brunch?
   - But I want it now!
   - I am a big pony!
   - I'm not a baby, I can take care of myself!
   - Likely story.
   - Not the cupcakes!
   - Some pony needs to put this thing out of its misery.
   - You're not using power tools, are you?
   - Scootaloo! Scoot-Scootaloo!
   - Trust me.
   - What a thing to say!

6.3 Bara en enda Pony

Notera signaturen för main: `public static void main(String [] args)`

public så att man kommer åt den utanför klassen

static så att man inte behöver först skapa ett objekt

void man kan inte returnera något

main så att runtime-java vet vilken metod den skall leta efter

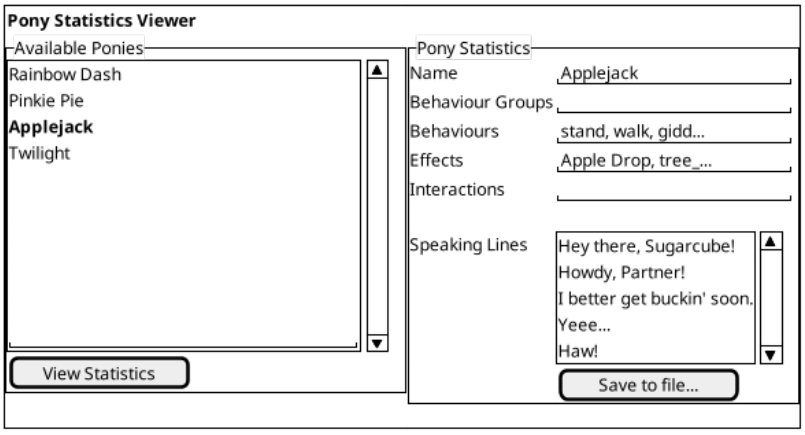
String [] args Här kommer alla kommandorads-parametrar.

Dags att lägga till lite interaktivitet:

1. Om `(0 == args.length)` , lista statistik för alla ponies (som tidigare)
2. Annars, hitta alla ponies som innehåller `arg[0]`.
   - Du kommer vilja se till att allting är antingen stora eller små bokstäver: `String::toLowerCase()` .
   - Det räcker att veta om ponnyns namn *innehåller* strängen, använd `String::contains()` .
   - `pony.getName().toLowerCase().contains(arg[0].toLowerCase())`

6.4 Ett Grafiskt Gränssnitt

1. Skriv en ny klass `view/PonyStatisticsViewer` som skapar en JFrame enligt nedan.
2. Skriv en funktion för att fylla listan med namnen på alla tillgängliga Ponies.
3. Skriv kod så att när man har valt en Pony och trycker på knappen "View Statistics", så visas statistiken till höger.
4. Vänta med "Save to file..." - knappen.



6.5 Save to File...

Nu är det dags att implementera "Save to file...":

1. När man trycker på knappen skall en `javax.swing.JFileChooser` öppnas.
2. Statistiken om den valda Ponyn skall sedan skrivas till den angivna filen.

3. Kontrollera att filen har rätt innehåll genom att öppna den (eller visa den i din terminal)

1. MWE for Save to File

```
import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.filechooser.FileSystemView;

public class FiCH {

public static void main(String [] args) {
  JFrame f = new JFrame("FiCH");
  f.setSize(500, 500);
  f.setVisible(true);
  JLabel l = new JLabel("no file selected");

  JButton button1 = new JButton("save");
  button1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        JFileChooser j = new JFileChooser(FileSystemView.getFileSystemView().getHomeDi
        int result = j.showSaveDialog(f);
        if (result == JFileChooser.APPROVE_OPTION) {
          l.setText(j.getSelectedFile().getAbsolutePath());
        } else {
          l.setText("the user cancelled the operation");
        }
      } });

  JPanel p = new JPanel();
  p.add(button1);
  p.add(l);
  f.add(p);
}
}
```

7 Virtuella Maskiner och Containers

7.1 Kom igång med några tutorials

1. Docker `https://docs.docker.com/get-started/`
2. Docker with node.js `https://docs.docker.com/language/nodejs/`

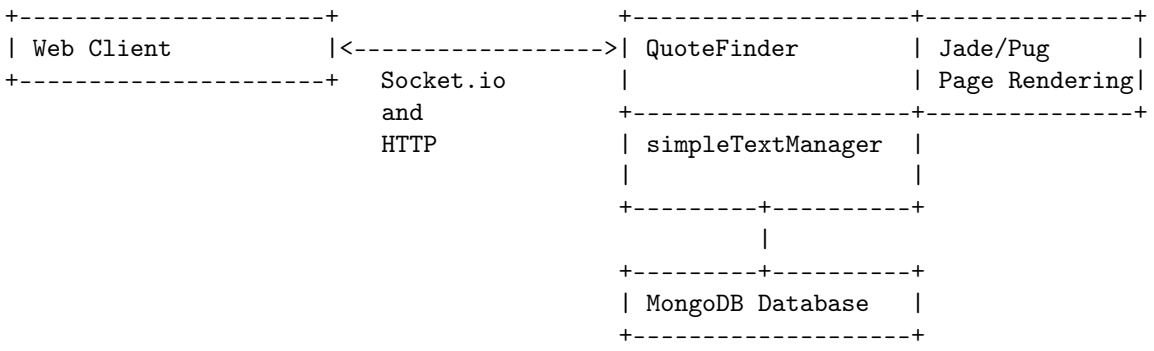
7.2 Om Projektet: QuoteFinder

- Ladda ner Projektet: `https://github.com/mickesv/ProvisioningDeployment.git`

- Är egentligen en del av en kurs i *Applied Cloud Computing and Big Data*
- Var lugn, vi kommer inte använda allt i den här kursen.

- *QuoteFinder* letar efter citat i texter.
  - Tre versioner
    * Version 1, Letar efter en hel sträng
    * Version 2 && 3 letar efter orden nära varandra.

  Version 1 Enklare, går att köra nästan ensamt.
    * Version 2 && 3 består av ett antal kommunicerande microservices.
  - Skrivet i Javascript / node.js `https://nodejs.org/` (ett tolkat språk)
  - Express web app `http://expressjs.com/`
  - Använder också socket.io `https://socket.io/`
  - Kopplar upp sig mot en MongoDB-databas `https://www.mongodb.com/`
  - Exponerar tre websidor: `/` , `/add` , och `/list` .



7.3 Kolla koden: `Containers/Version1/QFStandalone/src/`

- Läs igenom `index.js`
  - Hur skapas en "route"?
  - Vad händer när man t.ex. begär sidan `/add` ? Vilken funktion anropas?

- **Ni behöver *inte* fördjupa er om följande**:
  - `res.render()` använder sig av *Jade/Pug* för att skapa en websida: `https://pugjs.org/`
    * Vanligt arbetssätt; "lättare" att skapa en websida med dynamiskt innehåll från något mall-system.
  - *Promises* är ett sätt att länka saker som skall hända asynkront så att de ändå sker i ordning.

        * En ledtråd är när man hittar kod `.then()`; då arbetar man troligen med en *Promise*.

- Läs igenom `simpleTextManager.js`
  - Vad gör klassen / vilka metoder / vilka ansvarsområden har den?
  - Titta lite närmre på metoden `addText()`
    * vad gör den?
    * varför tror du att den sparar texterna på det här viset?

7.4 Bygg en image

- gå till katalogen där `Dockerfile` ligger, `Containers/Version1/QFStandalone/`
- Titta på `Dockerfile`, förstår du hur den är uppbyggd och vad som kommer hända?
- Bygg en image: `docker build -t qfstandalone .`
  - Vad händer?
  - Notera hur den bygger upp lager efter lager.
- Kontrollera efteråt att den faktiskt byggdes `docker image ls`
  - Vilka fler images har du? Varför tror du att de finns där?

7.5 Starta applikationen: podman/docker

1. Applikationen använder `MongoDB`, så vi behöver hämta den: `docker pull mongo`
2. Vi behöver ett nätverk för att qfstandalone skall kunna prata med databasen:
   - `docker network create qfstandalone-net`
3. Starta databasen: `docker run -d --network qfstandalone-net --network-alias textstore --name textstore mongo`
4. Starta applikationen: `docker run -it --network qfstandalone-net -e TEXTSTORE_HOST=textstore -w /app -v ./src:/app/src --name qfstandalone -p 8080:3000 qfstandalone`

Förklaring: Starta Databasen

```
docker run                  # Start a Container
-d                          # In detached mode (in the background)
--network qfstandalone-net  # Connect to the virtual network we just created
--network-alias textstore   # Make this container accessible
                            # on the network using this name
--name textstore            # Use this name when we access
                            # the container with docker
mongo                       # Use this image as base for the container
```

Förklaring: Starta Applikationen

```
docker run                      # Start a container
-it                             # In interactive mode, and attach
                                # a terminal so we can also type into it
--network qfstandalone-net      # Same virtual network
-e TEXTSTORE_HOST=textstore     # Set the environment variable to the
                                # network alias of our MongoDB database
-w /app                         # Set the working directory inside the container
-v ./src:/app/src               # Attach the host directory ./src
                                # to the guest under /app/src
--name qfstandalone             # Container name
-p 8080:3000                    # Connect host port 8080 to
                                # port 3000 in the container
qfstandalone                    # Use this image (the tag we previously set)
```

7.6 Testa

1. Lägg till en bok, gå till: `http://localhost:8080/add`
   - Använd förslagsvis en bok från Gutenberg-projektet `https://www.gutenberg.org/`
   - Om du inte skriver in något så kommer du lägga till en textversion av Leo Tolstoy's *Krig och Fred*
2. Gå till `http://localhost:8080/` och sök efter något, till exempel 'prince'.

Att göra:

- Håll ett öga på din terminal. Vad skrivs ut? Vad händer?
- Eftersom vi startade med flaggorna `-it` så kan vi kontrollera appen i terminalen:
  - Prova skriv `rs` och tryck på `<enter>` , vad händer?
  - Det här är för att vi kör programmet med hjälp av `nodemon` : `https://nodemon.io/`
- Vi startade också programmet med en *bind mount* : `-v ./src:/app/src`
  - Öppna filen `src/index.js` och leta rätt på metoden `startPage()`
  - Byt ut return-raden mot `return listTextsPage(req, res);`
  - Vad händer i terminalen?
  - Ladda om startsidan i webläsaren; du bör nu också se en lista med alla tillgängliga texterna.

7.7 Avbryt, Stoppa, och Städa upp

- Avbryt den körande applikationen genom att trycka `Ctrl-C` i terminalen.
  - Det här stoppar den körande containern `qfstandalone`
  - Databas-containern `textstore` fortsätter köra i bakgrunden
  - Nätverket finns fortfarande tillgängligt

  - Kolla vad som finns kvar: `docker ps -a`

- Dags att rensa:

```
docker rm -f textstore qfstandalone
docker network rm qfstandalone-net
docker network prune -f
```

7.8 Starta applikationen: podman/docker compose

- Vi har redan introducerat en `docker compose` -fil för att starta applikationen.
- Öppna och studera filen `docker-compose-v1.yml`
- Starta applikationen med `docker compose -f docker-compose-v1.yml up`
- Testa som innan med `http://localhost:8080/` och `http://localhost:8080/add`

Att göra:

- Notera hur utskrifterna i terminalen skiljer sig.
- Vad händer om du skriver `rs` som innan i terminalen?
- Vad händer när du avbryter med `Ctrl-C` ? Kontrollera med `docker ps -a`

1. Överkurs: kommunicera med applikationen
   (a) Uppdatera docker compose-filen (Se nedan)
   (b) I en separat terminal, koppla på dig på den körande containern: `docker compose -f docker-compose-v1.yml attach app`

```
version: "3.8"
services:
  app:
    image: qfstandalone
    stdin_open: true # docker run -i
    tty: true        # docker run -t
    ports:
      - 8080:3000
    volumes:
      - ./Containers/Version1/QFStandalone/src:/app/src
    environment:
      TEXTSTORE_HOST: textstore
  textstore:
    image: mongo
    command: --quiet --syslog
    expose:
      - "27017"
```

7.9 Sammanfattning

1. Bygg en `Image`
2. Starta en `Container`
    - Starta endast en container åt gången
    - Starta flera containrar med ett enda kommando
3. Redigera filer lokalt och se dem ändras i en körande container

Fördelar:

- Kan köra vilka program och programspråk du vill i en container.
- Upprepningsbar deployment.

Nackdelar:

- Kan köra vilka program och programspråk du vill i en container; inklusive malware.
- Det borde vara, men är inte, helt transparent att ta nästa steg ut på molnet.
- Databasen är inte persistent än...

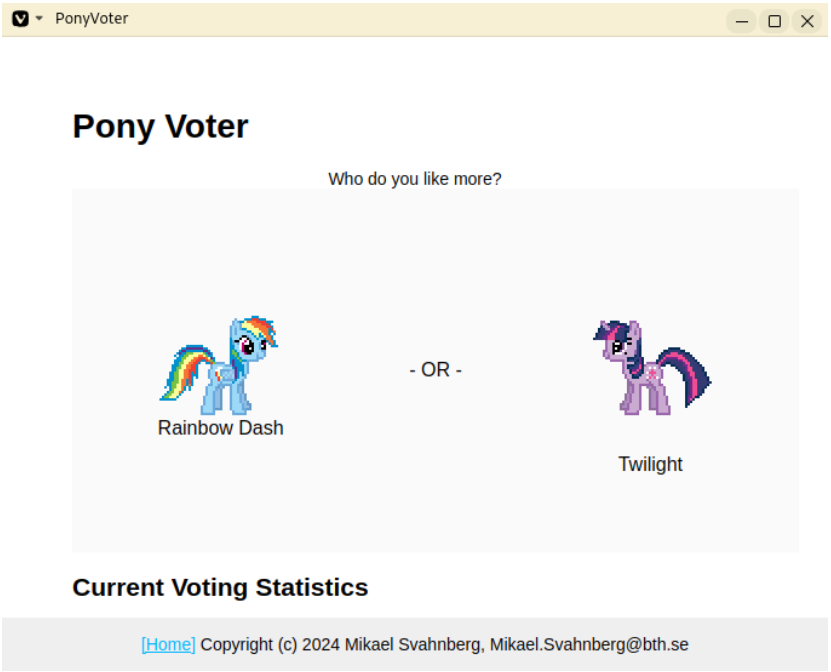
8 Utveckla med Microservices

8.1 Introduktion: PonyVoter

- Vi stannar i Equestria, men med en enkel röstningsapp den här gången.
- PonyVoter presenterar två alternativ åt gången, och man röstar genom att klicka på en av dem.
- Rösterna räknas i en databas, så att man över tid kan se vilken ponny som är mest populär.

Kom igång:

1. Ladda ner projektet: `https://codeberg.org/mickesv/PonyVoter.git`
2. Studera filerna, försök bilda din egen uppfattning om vad du har laddat ner.



8.2 Teknisk Översikt

- PonyVoter består av tre containers och en databas:

  PonyVoter "Framsidan" på applikationen som servar webbsidor till användarna

  VoteCounter Registrerar röster och sparar dem till databasen

  StatsPresenter Räknar ihop hur många röster respektive ponny har och sammanfattar detta

  MongoDB Databasen där rösterna lagras

- PonyVoter är hopplöst överdesignat och samtidigt underimplementerat
  - `VoteCounter` och `StatsPresenter` är extremt enkla, och hade antagligen inte behövt ha egna Containers i nuläget.
  - Mycket är hårdkodat.
  - För att inte kräva för många extra resurser används ingen renderingsmotor (såsom `Pug`) för att generera HTML-koden.
  - För att hålla projektet litet finns det bara sex ponnies att välja mellan.
  - Fullständighet? Bara det allra nödvändigaste finns implementerat.
  - Skalbarhet, vad händer när det totala antalet röster ökar?
  - Buggar! De finns såklart.
  - Säkerhet?

Fundera på:

1. Vilka containers skall vara tillgängliga för användaren?
2. Hur ser du till att bara dessa blir tillgängliga?
3. Hur kan du starta alla containers med ett enda kommando?

8.3 Starta och Testa

1. Filen `ponyvoter.yaml` används av `docker compose` för att bygga och starta applikationen.
   - Hur är den uppbyggd?
   - Vad finns angivet för varje container?
   - Är `volumes` - blocken nödvändiga? Vad gör de?
   - Kan du se hur man kommer åt respektive container?
2. Starta applikationen: `docker compose -f ponyvoter.yaml up`
3. Gå in på `http://localhost:8080` och testa applikationen
   - Håll ett öga på terminalen när du kör; vad skrivs ut?
4. Avbryt genom att trycka `Ctrl-C` i terminalen.
   - Vad händer?
   - Kolla med `docker images` vilka images som du har
   - Kolla med `docker ps -a` vilka containers som körs respektive inte längre är igång
5. Starta igen (samma kommando)
   - Vad händer?
   - Notera att statistiken nollställs inte, trots att alla containrar startats om.
     - Varför?
     - Hur kan du ta reda på mer om detta?

8.4 Hitta Databasen

1. Kontrollera vilka volymer som docker har skapat `docker volume ls`
   - Det borde finnas två med långa icke-namn, ex. `aa5972d833f74bc8085bafdc32aa279e45c8d29cf63`
   - Kan det vara dessa som är databasen?
   - går det få mer information? `docker volume inspect aa5972d833f74bc8085bafdc32aa279e45c8`
2. Gå bakvägen. `docker ps -a` visar att databasen heter `ponyvoter-mongodb-1`
   - Vad får du för information från `docker inspect ponyvoter-mongodb-1`?
   - Leta efter "Mounts" i utskriften, eller filtrera lite först: `docker inspect -f '{{.Mounts}}' ponyvoter-mongodb-1`

Vår misstanke stämmer alltså. MondoDB använder alltså två volymer:

- `/data/configdb` och `/data/db` .

Uppgift:

1. Läs på om *Volumes* i dokumentationen till docker compose.
2. Modifiera `ponyvoter.yaml` så att `mongodb` använder två *namngivna* volymer; `db-data` och `db-config`.
3. Rensa bort de båda gamla med `docker volume prune` .

8.5 Skala applikationen

- Eftersom alla containers håller sig till REST-principerna, så går det enkelt att skala.
- I `ponyvoter.yaml` kan man ange hur många `replicas` en viss service skall ha i en viss driftsättning.
  - Det är lite mer invecklat än så; läs på i den officiella dokumentationen först.

Att göra

1. Uppdatera `ponyvoter.yaml` så att den driftsätter 3 st replicas av `votecounter`.
2. Starta om applikationen.
3. Rösta på ett antal ponnys och håll koll på terminalen: vad händer?
   - Finns det någon ordning i hur dina tre replicas används?
4. Fundera på:
   - Kan du ändra till 5 replicas *utan* att starta om applikationen? Hur? Prova!
   - Tips 1: Räcker det att ändra i yaml-filen?
   - Tips 2: `up` tar flaggan `--detach`
   - Tips 3: Du kanske inte ens behöver ändra i yaml-filen...
     - kolla vad du kan göra med `docker compose --help`

8.6 Erbjuda och Använda REST

Det är dags att titta inuti applikationen också.
`Containers/StatsPresenter`

- Har en enda kodfil: `src/index.js`
- Det finns i huvudsak fyra delar:
  1. Skapa en express-webserver
  2. Koppla upp mot databasen
  3. Ställ in och sätt igång alla REST-ändpunkter som applikationen skall lyssna på

4. Funktioner för varje ändpunkt

`Containers/VoteCounter`

- Ser i princip likadan ut.

`Containers/PonyVoter`

- Lite fler funktioner, men i stort sett samma struktur.

Att göra:

- Vilka REST-ändpunkter erbjuder respektive container?
- Är de GET, POST, PUT, eller DELETE? Vad borde de vara?
- Vilka typer av svar ger respektive ändpunkt?
- Hur kan du testa det?

8.7 Testa API:et

- Bara `PonyVoter` är tillgänglig från värd-datorn...
- hur kan vi testa de andra containrarna?

Att göra

1. Studera `Containers/APITester` så att du vet vad den gör.
   - Studera även `test.yaml`.
2. Starta `PonyVoter` - applikationen
3. Kör `docker compose -f test.yaml up` och se vad som händer.
   - Notera att du har tre olika typer av svar, med olika `Content-Type`.
   - Hur kan du använda detta när du bygger ett REST-API?

8.8 Fundera på / Ta reda på

- Kan du kontrollera om en container är frisk?
  - Hur skriver du en sådan *healthcheck* i din docker compose-fil?
  - Måste du alltid ha en särskild ändpunkt i ditt REST-API för detta?
    - När måste du definitivt ha en särkskild ändpunkt?
    - Finns det andra lösningar?
- Vissa driftsättningsplatformar har begreppet *Init Containers*
  - Vad använder man init containers till?
  - Hur kan du åstadkomma detta med docker compose?
- Vad är docker compose *Secrets* ?
  - När skall du använda dem?
  - Hur?
- Vad behöver du göra för att din docker compose-file skall bli färdig att driftsättas (*Production Ready*)?

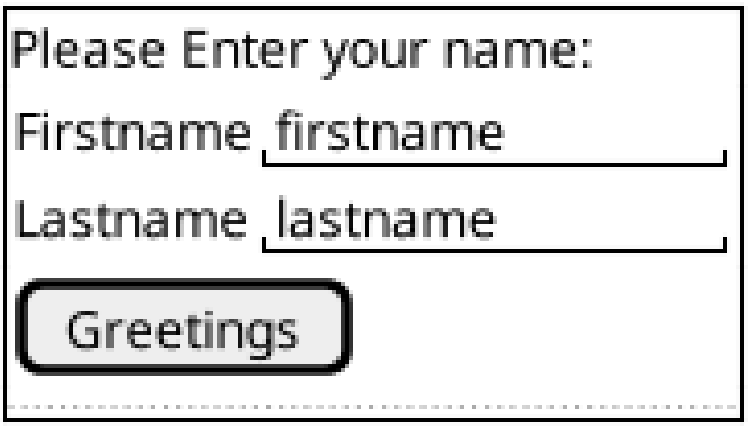
8.9 Sammanfattning

- Du har nu arbetat med en *microservice* - applikation
- Varje komponent (Container) har sitt eget *REST-api*
- Du har använt flera olika programspråk (`JavaScript/Node.js` och `bash`)
- Du har skalat delar av din applikation upp och ner

9 Kom igång med JavaScript

9.1 Introduktion till Övningen

- Den här gången skall vi börja från grunden med ett helt nytt projekt.
- Du kanske vill skapa projektet på din git-server först och klona det därifrån
  - Om inte, starta åtminstone i en ny katalog med `git init`
- Projektet går ut på att man får fylla i förnamn och efternamn på en websida, och få en hälsning tillbaka.
- Som en del av projektet skall du *minst* skapa följande
  1. En `Dockerfile` som
     - installerar `nodemon`,
     - installerar alla övriga beroenden från `package.json`, och
     - startar applikationen med `ENTRYPOINT ["npm", "run", "dev"]`.
  2. En `package.json` som minst:
     - deklarerar `Express ^4.19.1` som ett beroende
     - har ett `dev` - script som startar applikationen med hjälp av nodemon.
  3. En fil `src/index.js` som startar upp en express-webapplikation med två routes
     - `GET /` som levererar en sida enligt nedan
     - `GET /greet` som lägger till en rad `Hello, Firstname Lastname!`.
  4. En fil och en klass `src/person.js` som
     - representerar en person med `firstname()`, `lastname()`, och `fullname()`.
     - Lagrar namnen med versal första bokstav (ex lagras “john” som “John”)
     - Har en metod `greet()` som returnerar `this.fullname()`.
     - Glöm inte `module.exports = Person`
  5. (gärna) en `makefile` med två regler:
     - en `build` (`docker build .  -t namegreeter`)
     - en `run` (`docker run -it -p8080:3000 -w /app -v ./src:/app/src namegreeter`)



9.2 Spara Hälsningar

- Nästa steg är att lägga till en sida till `GET /list` som visar alla personer man tidigare hälsat på

Att göra

1. Lägg till en array `previousGreetings` i index.js, som du sparar dina `Person` - objekt i.
2. Lägg till en route till `GET /list` i `index.js` som visar alla personer från din `previousGreetings`.

9.3 Räkna Hälsningar

- För varje person man hälsar på, kolla i `previousGrettings` om du har hälsat på den personen innan (antag att om "Firstname Lastname" är samma så är det samma person)
- Lägg till en räknare i Person-klassen som ökas varje gång man hälsar på den personen.

9.4 Familjerelationer

- Om bara efternamnet stämmer, (och inte förnamnet) så är det en släkting.
- Lägg till metoden `addRelative(aPerson)` i din Person-klass.
  - Dubbelkolla (för säkerhets skull) så att personen inte redan är listad som en släkting
- Fixa `GET /list` så att alla släktingar listas för varje person.
- Fixa `GET /list` så att namnen skrivs ut i bokstavsordning baserat på efternamnet.

9.5 Sammanfattning

- Du har nu:
  - skrivit en enkel web-applikation i en container
  - skrivit en klass i Javascript.
  - Lagt till metoder i klassen.
  - Sparat objekt i samlingar och hämtat dem därifrån.

10 Applikationsutveckling med JavaScript

10.1 Introduktion: Craic – ett enkelt chat-program

- I den här uppgiften skall vi arbeta med ett enkelt chat-program: *Craic*.
  - *Craic* är ett irländskt ord för skvaller.
  - Applikationen går ut på att man skriver små korta meddelanden till varandra.
  - `https://codeberg.org/mickesv/craic.git`

10.2 Utmaningar

Sätta sig in i en existerande kodbas
- Hellre än att jag går igenom systemet och systemarkitekturen så får *ni* glädjen att göra det.
- Hur kör man programmet?
- Vilka huvudsakliga komponenter finns?
- vad gör respektive modul?

Användargränssnitt från inuti en container
- Vi hade kunnat skriva en web-klient, men vi vill ha något annat
- Textbaserat UI, så kallat *TUI*

10.3 Kom igång med projektet

1. Ladda ner projektet: `https://codeberg.org/mickesv/craic.git`
2. Sätt dig in i projektet
   - Hur kör man programmet?
   - Vilka komponenter finns?
   - Vad gör respektive modul?
3. Testkör
   - Skriv några inlägg
   - Lägg datormusen upp-och-ner och försök använda bara tangentbordet
     - (tips: Man behöver trycka `<escape>` för att lämna ett textfält)
4. Kan du koppla upp dig mot en kollegas server? Hur?

10.4 Uppdatera Klienten

1. Lägg till ett textfält med namnet på servern som skall användas
2. Se till att den angivna servern faktiskt används
3. Testkör tillsammans med en kollega

10.5 Uppdatera Servern

1. Lägg till fler otillåtna ord och namn (notera att en del är angivna som *Reguljära Uttryck*)
2. Lägg till en modul som gör att man kan `#tagga` nyckelord och `@nämna` andra användare
   - Skall de bara sparas tillfälligt i servern eller skall de lagras i databasen?
   - Hur söker man efter en viss `#tag`? Lägg till det till serverns REST-API.
3. Lägg till stöd för att hämta flera sidor av inlägg
   - Man behöver lägga till `page=xxx` till frågan
   - Man behöver lägga till `page=xxx`, och `nextPage: yyy` i svaret.
   - Extrapoäng om ni inte använder sidnummer rakt av utan räknar fram en nyckel i stället.
     - (Att man kan lista ut sidnummer är ett säkerhetshål)

Fundera på:

- Vad händer om man använder ett förbjudet ord men stavar det annorlunda, t.ex. “belGIUm”?
- Hur kan du testa dina API-förändringar?
- Blir det en ny major version av produkten när du lägger till nya REST-ändpunkter?
- Blir det en ny major version när du lägger till stöd för att hämta fler sidor?
- Kan du förenkla servern så att den har en konfigurerbar lista med filter att tillämpa, snarare än att de är hårdkodade?
  - Fundera på hur du skulle implementera detta.

10.6 Skriv Tester

1. Planera och skriv Mocha/Chai-tester för servern
2. Planera och skriv Mocha/Chai-tester för klienten

10.7 Skapa en web-klient

1. Skapa en ny Container som kör en webklient på samma vis som den TUI-baserade klienten.
2. Kör programmet med både TUI-klienten och webklienten igång samtidigt.
3. Kan du återanvända dina tester från TUI-klienten?

10.8 Fundera på REST-API:et

- Hur vet klienterna om det finns nya meddelanden?
  - Vad innebär detta för servern?
- Hur kan du göra det annorlunda / snällare för servern?
  - Försök!

10.9 Sammanfattning

- Med containers och moduler blir varje del av programmet ganska fristående och lätt att anpassa.
- REST-API är inte lika enkelt som ett vanligt metodanrop, men nästan.
- Skalbarhet:
  - En egen container för att hantera `#taggar` och `@omnämnanden`?
  - Flera server-containrar med lastbalanserare?
- Olika typer av klienter

11 Kom igång med Databaser

11.1 Docker Compose-fil

- Vi behöver inget git-repo den här gången, utan börjar med en enkel docker-compose-fil (se nedan)
  - Vad gör den här filen?
  - Default-användaren heter `postgres`, men man måste ange lösenordet.
- Starta med `docker compose` som vanligt.

```
version: '3.9'
services:
  db:
    image: postgres
    restart: always
    shm_size: 128mb
    environment:
      POSTGRES_PASSWORD: hunter2
  adminer:
```

```
image: adminer
restart: always
ports:
  - 8080:8080
```

11.2 Översikt om Adminer

- Öppna en webläsare mot `http://localhost:8080`
- Vid inlogget behöver du ange

  System PostreSQL

  Server db (eftersom det är vad servern heter i docker compose-filen vi skapade)

  Username postgres

  Password hunter2 (eller vad du nu ändrade det till i docker compose-filen)

- Adminer ger dig möjlighet att klicka dig fram i webläsaren för att skapa databaser, tabeller, och värden
- Du kan också skriva SQL-kommandon direkt
  - Det här är tacksamt när man skall göra större eller upprepade operationer.

Tips Om du är God Vän™ med din editor kan du säkert koppla upp dig mot databasen därifrån.

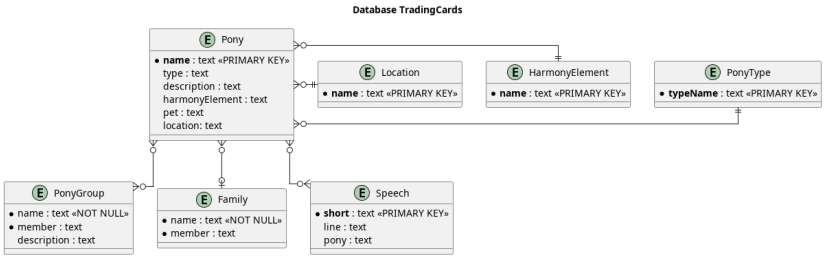
- Du behöver då se till att databasen är tillgänglig på port 5432 även utanför docker-compose-klustret.
- Glöm inte att "stänga in" den igen när du har utvecklat färdigt så att inte någon utifrån kan hacka din databas.

11.3 Skapa En Databas

- Vi fortsätter med pony-extravagansan.
- Den här gången vill vi bygga en databas för att kunna generera "Trading Cards"
- Vi bortser för stunden från bilder, cutie marks, och liknande och håller oss till ren text.
- Databasdesignen är förenklad; det är här ni behöver en hel kurs om databaser bara för att förstå hur och varför.
- **Att göra** Skapa en databas `TradingCards`.

11.4 Skapa Tabeller och Kolumner

1. Skapa följande tabeller och kolumner:



11.5 Fyll på med Data

1. Ladda ner filen `https://codeberg.org/mickesv/gists/raw/branch/main/TradingCards_insert.sql`
    - Titta igenom filen så att du förstår vad den gör.
2. Leta rätt på sidan `"SQL command"` I adminer-gränssnittet.
3. Klistra in filen och tryck på execute.
    - Om du skapade databasen korrekt skall alla `INSERT` fungera.

- Annars, *läs felmeddelandet*, åtgärda och försök igen.

4. Fyll på med några fler Ponnys, t.ex. härifrån:
   - `https://mlp.fandom.com/wiki/My_Little_Pony_Friendship_is_Magic_Wiki`

11.6 Enkla Sökningar

1. Använd adminer-gränssnittet och gör några enklare sökningar.
   - Till vänster finns det länkar `select` och `tabellnamn` för varje tabell.
   - Välj "select" för rätt tabell, och fyll i fälten för "Select" och "Search" så att du kan hitta:

Visa följande fält	från tabellen	som matchar vilkoret
name, type	pony	name är exakt (=) "Rarity"
name, type	pony	name innehåller (~) "Twilight"
member	family	name är exakt "Apple"
pony, line	speech	pony innehåller 'Rainbow'

11.7 Kombinerade Sökningar

1. Fundera på hur du skulle uttrycka följande frågor:
   - Hitta alla Pony.name och Pony.type för Ponies som är med i en Family.
   - Vad kan alla Pony som befinner sig i Ponyville tänkas säga?
   - Vad kan alla Pony som *inte* befinner sig i Ponyville tänkas säga, och vad heter de?
   - Hitta namnen på alla Ponies som nämns i en PonyGroup men som inte finns i tabellen Pony än.
2. Försök ställa dessa frågor i adminer.
   - Du kan behöva använda "SQL Command" för att lyckas.

11.8 Hantera Sökningar och Resultat från Datorprogram

1. Skapa en Container 'PonyTradingCard' som söker i databasen och listar alla Ponys enligt nedanstående mall.
   - Du väljer själv programspråk. I node.js behövs paketet `"pg"` för PostgreSQL.
2. Uppdatera din docker-compose-fil så att den här containern också körs.

```
--------------------
Pony: Fluttershy
Type: Pegasus
Element of Harmony: Kindness
Pet: Angel
```

Description: Very shy and scared of dragons.
Location: Everfree Forest

Family Members:
- Mr. Shy
- Mrs. Shy
- Zephyr Breeze

Groups:
- Gen 4
- Main Character

Speech:
- "Oh, my."
- "I don’t wanna talk about it."
- "I’d like to be a tree."