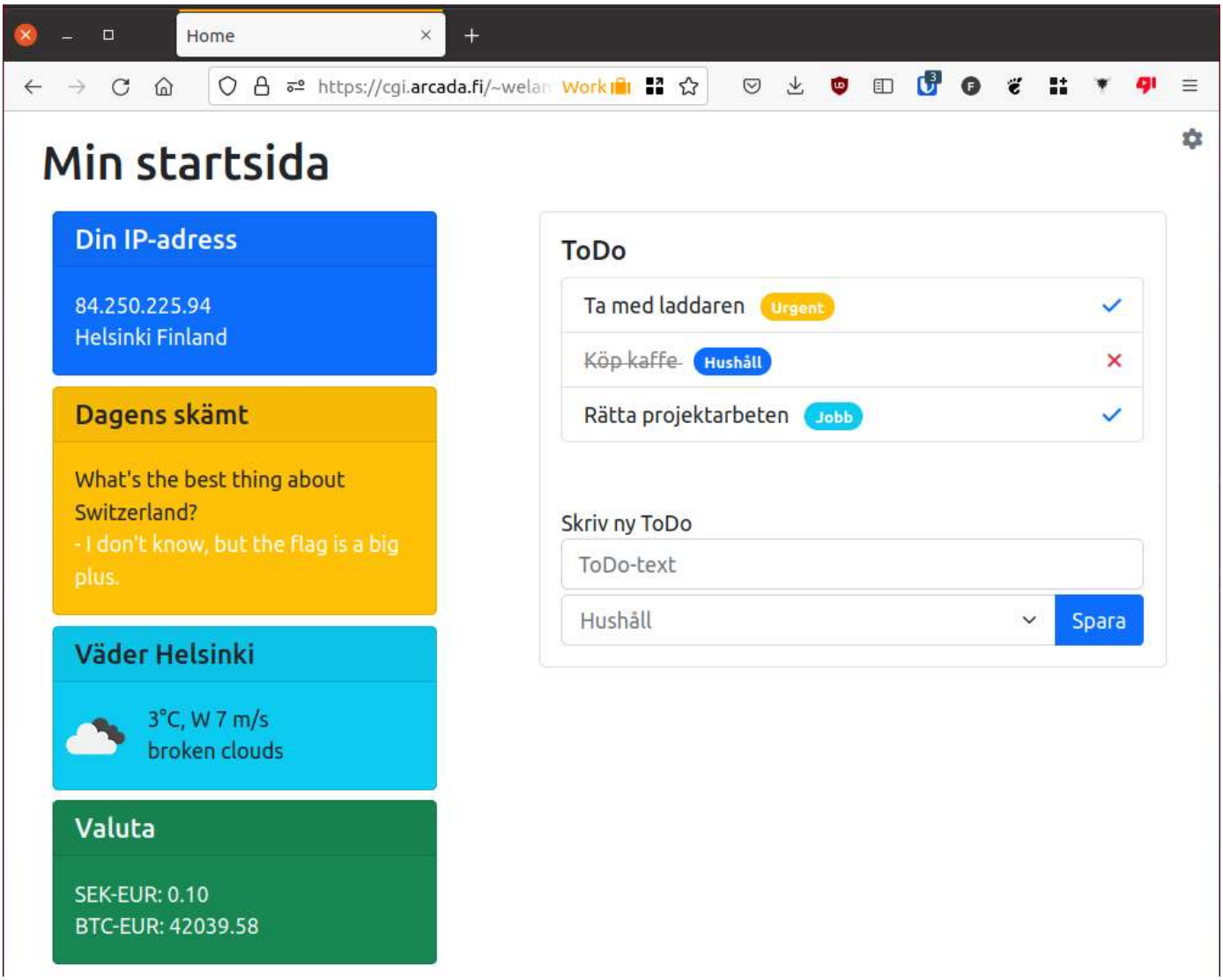
Detta är en gruppuppgift. Skapa först en grupp

Inlämningsuppgift

Projekt 1 - Min startsida

Projektet går ut på att skapa en egen startsida (att använda som "Home " i browsern)! Det här projektet kräver kunskap i framför allt webbkommunikation (API) och databaser. Använd gärna något CSS-ramverk (t.ex. bootstrap eller foundation) för att snygga till det hela!

- Detta projekt utförs i första hand som ett ***pararbete*** (samma parindelning kommer att gälla även för projekt 2).
- Projektet nås via GitHub Classroom på länken nedan. Den första i arbetsparet skapar ett nytt Team, den andra *joinar* teamet. Klona sedan ner projekt-repon och jobba vidare med den som grund för projektet.
- **Länken till projektet:** https://classroom.github.com/a/dbMvwBiG OBS det är ***OBLIGATORISKT*** att använda denna repo. Du får dock fritt skriva om boilerplate-koden om du vill.

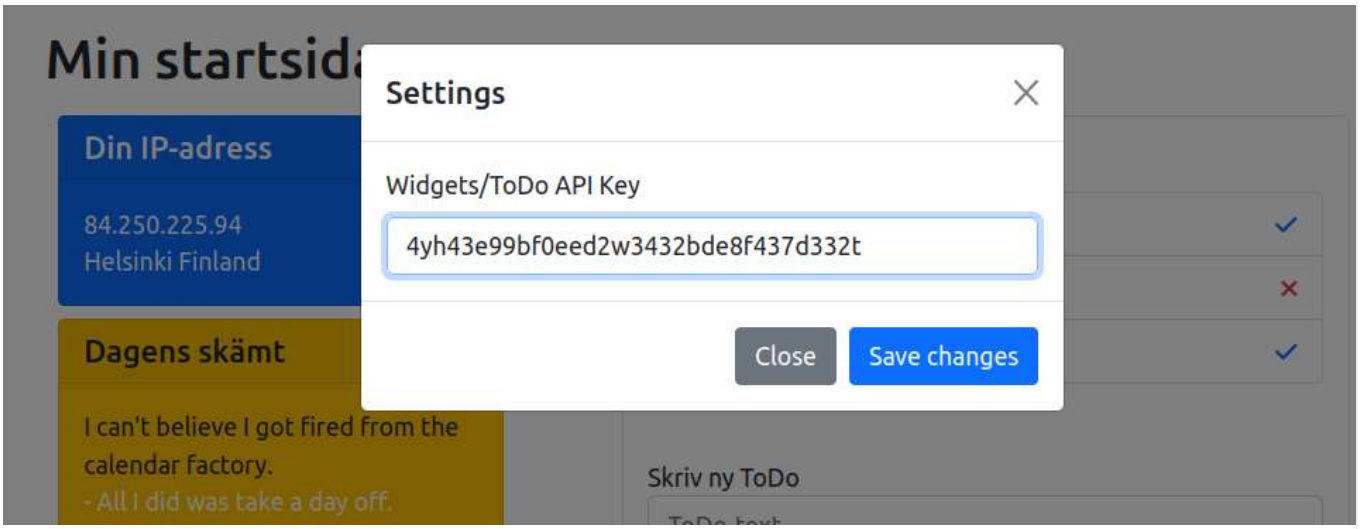


Del 1 – (10p) Widgets som konsumerar externa API:er

- Ha minst fyra widgets med nyttig information från externa API:er
- Widgetarna ska funka asynkront, och sidan ska funka trots att någon widget inte svarar
- Det är OK att köra det hela på cgi.arcada.fi (alltså bakom SSO-inloggning), men du kan fritt flytta din kod till någon annan hosting om du har tillgång.
- Om du använder API:er som kräver inloggningsinformation, var noga så du inte pushar hemligheter till GitHub! ***Se nästa del för en bättre lösning!***
- Förslag:
    - Väder
    - IP-adress och Geolocation
    - Lunchmenyer
    - VPN-info
    - Aktiekurser
    - Valutakurser

  - Nyhetsflöde
  - Vitsar

- Skapa någon form av Settings-meny som kan spara en variabel som cookie eller i localStorage. Kan vara en vanlig Javascript-prompt, men förstås är det ju bättre med något snyggare.
- Den här kommer att användas senare för att spara API-Key, men i det här skedet räcker det om det funkar att spara ett värde. Kan t.ex. se ut så här (man öppnar den genom att klicka på kugghjuls-ikonen i övre högra hörnet):



- Snygg kod, bra logik och UI/UX påverkar bedömningen. Det är ok att använda externa JS-bibliotek (jQuery, Vue, etc) men var konsekvent!
- TIPS: Du får gärna använda något CSS-ramverk för att snygga till det hela! Exemplen här är gjorda med *Bootstrap 5*, men du kan också kolla in *Zurb Foundation* eller t.o.m. gamla goda jQuery-UI.

Del 2 (10p) – Widget-API med databas

För att både du och ditt arbetspar (men ingen annan) ska kunna använda er startsida behöver ni någon form av autentisering, och det simplaste möjliga är att ha en api-key som man sedan skickar med i varjer request.

2a) Skapa en databas i PostgreSQL (du kan använda *Freddes DBaaS*-tjänst om du vill). Det är också OK att använd MySQL.

- Databasen behöver i det här skedet ha bara en tabell (det kommer fler senare).
- Lägg till användare med INSERT-queries, varje användare ska ha ett **namn** och en **api_key** samt ett JSON-fält (**widgets**) för widget-inställningarna. Hitta någon lämplig metod för att generera en lagom lång slumpmässig api_key (minst 32 tecken, kan förstås vara längre)

- I PostgreSQL kan du använda varchar (eg. character varying)-typen utan att ange längd.
- "serial" är i PostgreSQL en shorthand-datatyp för integer auto-increment

Det behöver inte se ut exakt så här, och de fält som finns i det här exemplet räcker knappast, men för att illustrera ungefär hur man kan göra:

users
id (serial, PRIMARY KEY)
username (varchar)
api_key (varchar)
widgets (json)
...

- Du får förstås lägga till egna nya kolumner vid behov!

2b) Koda ett enkelt API för era widget-inställingar.

- Skapa en endpoint som tar emot api_key som url-parameter och, om den finns i databasen, returnerar den användaren som ett objekt.
- Om du vet hur man gör är det förstås bättre att spara en *hash* av api_key:n i databasen (kolla in php password_hash()-funktionen) i stället för hela i klartext, men klartext duger i det här projektet.
- Funktionsprincipen kunde vara följande:
    - REQUEST-urlen någonting i stil med (OBS: Alltid http**s** när du skickar hemligheter!): https://cgi.arcada.fi/~username/api/widgets/?api_key=83gf264g2984fgq08374gtf243gt807g54
    - RESPONSE (json), kunde se ut ungefär så här:

```
{
    "username": "Fredrik Welander",
    "widgets": {
        "ip": {
            "url": "https:\/\/fw-teaching.fi\/ip\/"
        },
        "weather": {
            "url": "https:\/\/api.openweathermap.org\/data\/2.5\/weather?appid=472384a3(
        },
```

- Du behöver INTE ha något uppdateringsgränssnitt för widgets-inställningarna – det är OK att uppdatera direkt med queries i databasen

- Tanken är att man mot att skicka *api_key* får konfigurationen för de widgets man har på sidan!
- Uppdatera din Widgets-sida (Del 1) så att alla widgetar får sina urlar och eventuella andra inställningar från widgets-API
- Använd Settings-menyn (Del 1) och cookie/localStorage för att lagra din API-key

- Snygg kod och bra logik påverkar bedömningen för alla delmoment!

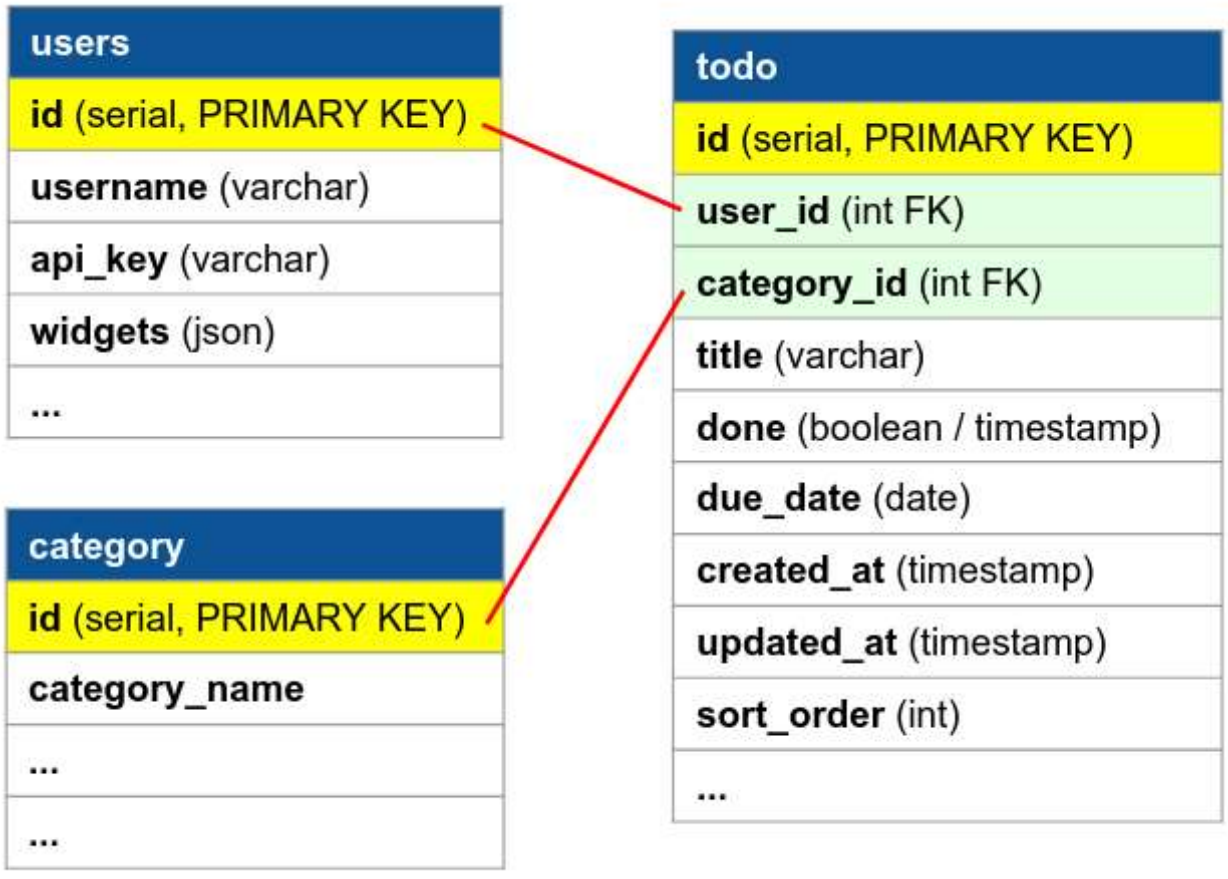
Del 3 (25p) – ToDo Widget med API och frontend

Utöka widget-sidan med en egen ToDo-widget! För att den här ska funka behövs databas och serverkod – skapa ett eget API!

3a) ToDo-Databas

- Använd samma databas som för widget-API, lägg helt enkelt till två tabeller, en för ToDo-noteringarna och en för kategorier.
- Använd kolumner med *Primary Keys* och *Foreign Keys* för att skapa relationer

Du behöver inte namnge alla fält exakt så här, och de fält som finns i det här exemplet kanske inte räcker till, men för att illustrera ungefär hur det kan se ut (relationerna ska dock fungera som i exemplet!):



3b) ToDo-API med CR(U)D

- Skapa ett nytt API på samma server som widgets-API, endpointen kan då vara t.ex. https://cgi.arcada.fi/~username/api/todo
- Alla requests ska kräva samma api_key som för widgets, du ska bara själv komma åt din ToDo-lista, och om ni jobbar i par ska ni kunna ha era egna på samma API!
- Du behöver INTE jobba enligt REST-principerna för fulla poäng (se bonus nedan), men serverkoden ska inte vara på samma url som frontend utan som en separat tjänst.
- API ska klara av **C**reate, **R**ead, **U**pdate och **D**elete av CRUD-helheten, men update-biten kan vara lite enklare (se bonus nedan för en mer fullständig version)
    - **GET /todo** – returnerar listan på alla ToDo-noteringar (för den användare vars api_key används)
        - Använd en JOIN för att få med kategorin från kategoritabellen
    - **POST /todo** – Själva todo-texten samt kategori (kategori-id) för att skapa en ny notering.
    - **PUT /todo** (du kan använda GET eller POST också) – skicka variabeln done=true (som JSON-body eller bara som url-parameter), ändrar status till *done*
    - **DELETE /todo** – radera ToDo-notering
- Kategorierna behöver inte gå att ändra på via API, det är OK att manuellt skriva in dem i databasen. Men relationen ska fungera som på bilden ovan.
- TIPS: Det är lite krångligt att få alla REST-metoder att funka på cgi-arcada.fi. De ovannämnda går att få att funka, PATCH verkar funka sämre. Om allt annat misslyckas så är det ok att köra med endast GET och POST i detta projekt.
- Snygg kod och bra logik påverkar bedömningen. Tänk också på säkerhet, t.ex. Cross-Site Scripting.

3c) ToiDo-Widget (frontend)

Koda själva Widgeten för ToDo och sätt med den på din Startsida!

- Widgeten ska endast vara frontend för widget-API, den ska vara kodad på en html-sida med endast html, JS och CSS, helt utan egen serverkod!
- Listan på ToDo-noteringar ska hämtas när man öppnar sidan (använd API-key från cookie/localStorage), och på nytt varje gång man gör någon ändring
- Följande egenskaper bör finnas med:
    - Skriva in ny ToDo-notering (skickar POST-request)

    - Markera en ToDo-notering som gjord (done) (skickar POST/PUT eller GET-request)
    - Radera en ToDo-notering som tidigare markerats som gjord (skickar DELETE-request)

- Testa och fundera vad som bör finnas med för att den ska vara användbar!
- Snygg kod, bra logik och UI/UX påverkar bedömningen!

BONUS (max 10p)

- Lägg till t.ex. följande egenskaper till ToDo:

    - Lägg till ett datumfält så du kan ha deadlines i din ToDo-lista
    - Lägg till möjlighet att ändra ordningsföljd på dina ToDo-noteringar
    - Ha någon form av *tabs* eller *filter* så att du kan visa endast de som inte är Done och tvärtom.
    - Fullständig CRUD, dvs det ska gå att editera en ToDo-notering och ändra text, kategori osv
    - Följ RESTful-principer så gott det går

Allmänt om projektet:

- **Projektet lämnas in genom att**

    - **1. pusha till GitHub innan deadline.**
    - **2. lägg upp en länk (här) till webbsidan (t.ex. på cgi.arcada.fi) som projektet finns på.**

- **PLAGIAT (att kopiera någon annans kod) innebär automatiskt underkänt.** Om du hittar någon bra kodsnutt på nätet (t.ex. för styling, responsivitet etc) kan du använda den, bara du anger källa, ***men*** den kod som direkt hör till projektets krav bör du skriva själv!