Full Stack Web Development

# Week 1 theorie

* A diagram of a full stack

  Description automatically generated with medium confidenceIemand die zowel client als server software kan ontwikkelen
  + Front-end
    - Client
    - HTML
    - CSS
    - Javascript
  + Back-end
    - Webserver
    - PHP
    - Python
  + Database
    - Databaseserver
    - SQLServer
    - Msql
    - MariaDB

## Front-end

* Browser
  + Chrome
    - Browser Engine: Blink
    - Javascript engine: V8

## Back-end

* Webserver (WSGI = Web Server Gateway Interface)
  + Flask
    - Webserver: Werkzeug

## Database

* Msql

## Hoe werkt een webserver

### URL

* Uniform Resource Locator
* Opbouw:
  + http://www.mct.be:80/map/opleiding.php?querystring=mct
  + Protocol: Welk protocol gebruiken we
  + Server: Naam of IP v/d webserver
  + Map/bestand.ext: Websites bestaan uit mappen en bestanden
  + Querystring: Extra variabelen
* Werking
  + Pakket komt aan op server
  + Wordt doorgestuurd naar poort 80 (standaard voor http)
  + Verbinding wordt gemaakt
  + A screenshot of a computer program

    Description automatically generated with medium confidencePakket levert request af
  + Bv een GET (Haldis)
  + Requests:
    - Get
    - Head
    - Post
    - Put
    - Delete
    - A screenshot of a computer code

      Description automatically generated with low confidenceConnect
    - Options
    - Trace
    - Path
  + Webserver zoekt gevraagde bestand
  + Voert eventuele code uit in het bestand
  + Indien gelukt: stuurt een response naar browser
  + Boodschap bestaat uit 1 of twee delen
    - Header altijd aanwezig
    - Body soms aanwezig

### HTTP Response Header

A picture containing text, font, screenshot, algebra

Description automatically generated

## HTTP

* Stateless protocol, geen constante verbinding
* Requests onafhankelijk van elkaar
* http kan onderweg niks onthouden

# Week 2 theorie

## Event Handling

### Events

* Gebeurtenissen
  + Gebruiker drukt op knop
  + Pagina is klaar met laden
  + Formulier wordt verstuurd
  + …

### Events opvangen en afhandelen

* Bv op knop klikken
* Standaard gebeurt er niks
* Verwittigen dat er moet geluisterd worden naar een bepaalde actie
* 2 problemen:
  + Hoe zeg ik iets aan de knop
  + Welke boodschap geef ik mee
* Button adresseren
  + const btn = document.querySelector(“.js-button”);
* boodschap meegeven
  + btn.addEventListener(welk\_event, wat\_moet\_er\_gebeuren);

## DOMContentLoaded

* JS uitvoeren voor HTML geladen is: foutmelding
  + Opl: scriptcode onderaan pagina
* Load-event
  + Wordt uitgevoerd wnr hele pagina geladen is (HTML, CSS, imgs, scripts, …)
* DOMContentLoaded-event
  + Wordt uitgevoerd wnr DOM geladen is (HTML)
  + Hierdoor kunnen we de init uitstellen en JS terug in head plaatsen

## CORS

* Cross-Origin Resource Sharing
* HTTP functie waarmee webtoepassing die onder het ene domein wordt uitgevoerd, toegang heeft tot bronnen in een ander domein
* Om JSON te kunnen ophalen moeten we toestemming hebben
  + Geregeld door CORS
* Browsers hebben een beveiligingsbeperkin
  + = same-origin-beleid
  + Voorkomt dat een webpagina api’s in een ander domein aanroepen
* Indien we JSON/data ophalen, wordt origin naar de server gestuurd
* Server antwoord met ACAO: Acces Control Allow Origin
* Hierbij kan server antwoorden dat:
  + Enkel jouw domein toegang heeft:
    - acces-control-allow-origin: http://www...
  + Iedereen toegang heeft:
    - acces-control-allow-origin: \*
  + Je geen toegang krijgt

## Fetch

* Externe gegevens ophalen op een asynchrone manier
* Bv: site maken met een weerbericht, maar om niet zelf telkens te moeten aanpassen: jsonbestand aangeleverd door hetKMI
  + Ophalen duurt soms lang: asynchroon ophalen
    - Hierdoor ziet de gebruiker een basisontwerp v/d site, dat aangevuld wordt met data zodra beschikbaar

### Promise

* A picture containing text, screenshot, font, line

  Description automatically generatedFetch werkt met promise
* Een JS-object dat 2 mogelijkheden heeft
  + Fulfilled: .then
  + Reject .catch

A screenshot of a computer code

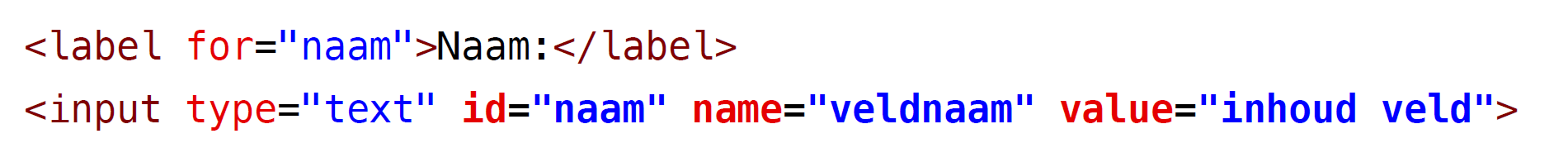
Description automatically generated with low confidence

# Week 3 theorie

## Recap

* Dienen om input v/d gebruiker op te vangen en te verwerken
* 2 delen
  + Front-end
  + Backend

## Label + tekstveld



* Attributen
  + Name: naam waarmee veld verstuurd wordt naar backend
  + Id: identificatie van veld in font-end
  + For: verwijzing naar id in frontend
  + Type: soort veld
  + Value: waarde van veld

## Formulieren verzenden

* Attributen:
  + Method – hoe gebeurt verwerking
  + Action – waar gebeurt verwerking

### Method

* Get: stuurt parameter naar server via URL
  + Gebruikt om documenten op te vragen
  + Maximum lengte
  + Ok om te cachen
  + Niet gebruiken voor wijzigingen op de server
* Post: stuurt parameter naar server via body
  + Gebruikt om data te wijzigen
  + Geen max lengte
  + Niet ok om te cachen
  + Ok om te gebruiken voor wijzigingen op de server

## Formvalidatie

* HTML5 validatie
* Javascript validatie

### HTML5 Validatie

* Required
* Min-/maxlength
* Min/max
* Type
* Pattern
* Samen met :valid en :invalid pseudoclass

### JS Validatie

* Meer controle
* Extra bovenop een pattern

# Week 4 theorie

## Back-end

### Flask

* Micro webframework in Python
  + Micro: bevat enkel basis (is uitbreidbaar)
  + Webfamework: web server gateway interface (WSGI)
* In staat om webrequests op te vangen en request terug te sturen
* 127.0.0.1**:5000 (i.p.v. 5500)**
* Flask in VSC
  + VENV (virtual environment)

# Week 5 theorie

* A close-up of a computer screen

  Description automatically generated with low confidenceZie DM
* Config.py
* Database.py
* Datarepository.py
* App.py

## Opvragen van alle klanten

A screenshot of a computer code

Description automatically generated with low confidence

## Opvragen 1 specifieke klant

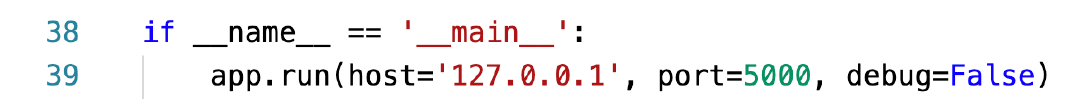
A picture containing text, screenshot, font

Description automatically generated

# Week 6 theorie

## SQLInjectioon

* Makkelijker bij een Flask project debuggen
  + Debug attribuut op False



* + Hierna kun je breakpoint zetten en runnen

A picture containing text, screenshot, font, line

Description automatically generated

* Bij SQL injectie wordt extra code toegevoegd aan input
  + Als in script geen voorzorgen:
    - Gevaarlijke situaties
      * Aanvaller kan:
        + Content in database tovoegen, wijzigen, lezen
        + Broncode lezen van bestanden op database server
        + Bestanden naar database server schrijven

E.g.: SQL = “select \* from tblUser where UserID = “ + txtUserID

Via code volgende doorgeven aan txtUserID:

“105 or 1=1”

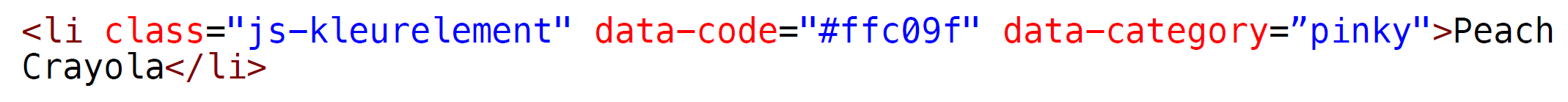
Or 1=1 is altijd true, waardoor alle userrecords getoond zullen worden

## Regions

* // #region een willekeurige naam
* // #endregion
* Goed voor overzicht

## Data attribuut

* Laat toe extra info op te slaan in HTML zonder dat deze gevisualiseerd wordt



* Kan opgevraagd worden vanuit JS:
  + Const gekozenCode = element.getAttribute(‘data-code’) ;
  + Const gekozenCat = element.dataset.category ;

## Dynamic Eventhandler

* Hoe eventhandler op element plaatsen dat is ingeladen via json
  + Ophalen met querySelector
  + Koppelen met Eventlistener
  + Waardes van huidige element ophalen met het keywoord ‘this’

const gekozenCategorie = this.dataset.category;

# week 7 theorie

## uitlezen v/d querystring

* hoe querystring van URL uitlezen in JS
  + URLSearchParams API

## Javascriptbestand voor 2 HTML’s

* Hoe app.js gebruiken in meerdere HTML bestanden
  + Voor grote projecten:
    - Gemeenschappelijke code in 1 bestand
    - Aparte code in aparte bestanden
    - Nadeel: code staat verspreid
  + Voor kleinere projecten
    - Alle code in 1 bestand
    - Gemeenschappelijke code bovenaan
    - Vanuit init verschillende code oproepen a.d.h.v. unieke selectoren

## Workflow

* 3 stappen
  + DOMContentLoaded
  + GET\_\_\_()
  + Show\_\_\_()
* Soms nog 3
  + ListenTo\_\_()
  + Get\_\_()
  + Show\_\_()
* HTML-elementen css klasse geven: js-x

## FAQ

* Wanneer noteer je .js-placeholder of js-placeholder
  + Met punt in javascript in queryselector
  + Sonder punt als class attribuut van html element
* Notatie #een-naam
  + Notatie met hashtag in JS in Queryselector ID aanspreken
  + In html zonder #
* Wat is innerHTML
  + “The Element property innerHTML gets or sets the HTML markup contained within the element”
  + In de DOM voeg je een extra html bloek toe binnen het element met de css klasse js-placeholder
* Kan ik innerHTML enkel voor het tonen van data gebruiken
  + Neen ook om gegevens op te vragen
    - Let element = document.querySelector(‘.js-stad’);
    - Let gekozenStad = element.innerHTML;
    - Console.log(gekozenStad);
  + Je vraagt de innerHTML op uit het element met de css klasse js-stad en toont die in de console
* Verschil tussen querySelector() en querySelectorAll()
  + querySelector() zal 1 html element teruggeven
  + querySelectorAll() zal een array van elementen teruggeven

# Realtime Communicatie

* Websockets
* Request (get, post,…) gevolgd door respons
  + Connectie wordt telkens verbroken
* We willen realtime data

## Wat is een websocket

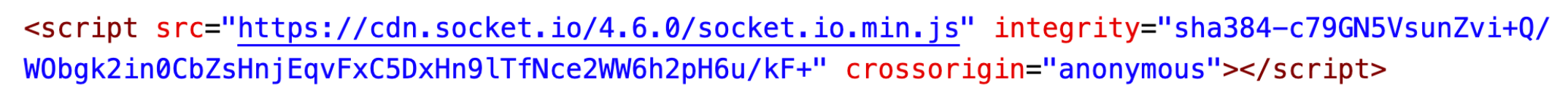
* Netwerkprotocol
* Full-duplex communicatie
* TCP-verbinding
* ws:// of wss://
* speciale request:
  + ‘connection: upgrade’ – aanvraag
* Server response:
  + 101 switching protocols
* Nu kan er gecommuniceerd worden zonder http-request of respones, maar via messages

## Socket.io

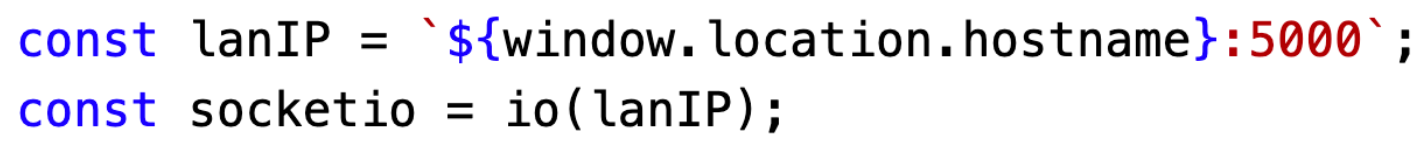
* Javascript library en python extension : Flask-Socketio

### Frontend

* Laad de code van Socket.io in de header in, voor je je eigen app.js inlaadt



* Om nu websocket connectie naar backend/server te openen:



* Eerst halen we huidige hostname op en plakken er :5000 aan: adres v/d backend
* Window.location.hostname: www.website.extensie
* Dan connectie naar backend openen door io(adres\_backend)

### Backend

* Venv
* Pip install -r requirements.txt
* Nieuwe paketten:
  + Flask-socketio
  + Gevent
* Nu in app.py code toevoegen
  + from flask\_socketio import SocketIO, send, emit
  + socketio = SocketIO(app, cors\_allowed\_origins=”\*”)
  + socketio = SocketIO(app, cors\_allowed\_origins=”127.0.0.1:5500”)

A picture containing text, font, screenshot, white

Description automatically generated

* als je nu python runt: nieuwe server opstarten i.p.v. werkzeug

## Messages sturen

### Werking

* Send: Enkel string
  + Frontend => Backend: socketio.send(“boodschap”);
  + Backend => Frontend: send(“boodschap”, broadcast=True/False)

A picture containing text, screenshot, font, algebra

Description automatically generated

* Emit: payload, object samen met custom eventnaam
  + Frontend => Backend: socketio.emit(“naam”, “boodschap”);
  + Backend => Frontend: emit(“naam”, “boodschap”, broadcast=True/False)

A screenshot of a computer code

Description automatically generated with low confidence

* Client naar server: 1-op-1 connectie
* Server naar client: 1-op-1 of 1-op-veel (broadcast)
* Good2Know: code om boodschap te sturen wanneer connectie tot stand komt:

A picture containing text, receipt, font, white

Description automatically generated

# API Authenticatie

## Authenticatietypes

* Anonymous
  + Iedereen krijgt toegang
* API Key
  + Bij elke request een API key meegeven via Authorization Header
    - Bv.: {“api-key”: “5742d889-4de3-4cf7-803c-ea3bff667eb9”}
* Basic authentication
  + Bij elk request een Base64 encoded string meegeven van username en password via authorization header
* Token authentication
  + De client, al dan niet na verificatie van paswoord/username, krijgt een token die vervalt na bepaalde tijd
  + Bij volgende request geeft client token mee via authorization header

A screenshot of a computer

Description automatically generated with low confidence

## Wat zijn JSON Web Tokens (JWT)

* JSON Web Token (Industry standard RFC7519)
  + Open standard om op een compacte en veilige manier gegevens tussen 2 partijen uit te wisselen, zoals een JSON object
  + De info kan geverifieerd en vertrouwd worden door de digitale handtekening
* Toegangsticket dat credentials en claims bevat
* Wordt door elke client verstuurd bij elk request
* 3 delen:
  + Header
    - Bevat gebruikte encryptie algoritme en type JWT
  + Payload
    - Bevat alle belangrijke gegevens over gebruiker, incl. autorisatie rechten zoals claims en permissies
  + Signature
    - Wordt gebruikt om echtheid van het token te garanderen
    - Bestaat uit hash van header en payload aangevuld met secret

A picture containing text, font, screenshot, lilac

Description automatically generated

A picture containing text, font, screenshot, lilac

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

## Setup backend

from flask\_jwt\_extended import JWTManager, jwt\_required,

create\_access\_token, get\_jwt\_identity



## Frontend

* Stap 1:
  + Eerst gebruiker identificeren als geldige gebruiker door POST-request naar loginAPI-endpoint met in payload v/d request een geldig email en paswoord

A screenshot of a computer code

Description automatically generated with low confidence

* A screenshot of a computer code

  Description automatically generated with medium confidenceStap 2:
  + Checken of gegevens kloppen en maken nieuw JWT token:
    - Create\_acces\_token()
* Stap 3:
  + Returnen JWT-token
* Stap 4:
  + Bij volgende request richting API moet verkregen token meegegeven worden zodat API gebruiker kan verifieren
  + Door Authorization header in te vullen met token die het login endpoint had teruggegeven

A picture containing text, screenshot, font, line

Description automatically generated

* A screenshot of a computer code

  Description automatically generated with medium confidenceStap 5:
  + Bevijligen enpoint door decorator toe te voegen
    - @jwt\_required
  + Identiteit opvragen door
    - Get\_jwt\_identity()
* Stap 6:
  + Retourneren response