
Technisch Ontwerp

The story so far

- Projectvoorstel in orde
  - Je hebt een duidelijk concept voor een website
  - Je weet welke features er moeten bestaan voor jouw concept
  - Je hebt beeld bij hoe de website eruit gaat zien

- Onderweg met CS50 Finance

High level design

- Bij ProgIK lag de focus op implementatie

- Nu is de uitdaging een groot software project, waar je vanaf scratch begint

- Waar begin je?

Van concept naar een project

- Wat moet er gemaakt worden?
- Hoe kunnen we dat maken?
- Hoe kunnen we samenwerken?

Wat moet er gemaakt worden?

- Je hebt een lijst van features die moeten bestaan (MVP)

- Een feature heeft verschillende technische implicaties

- Bijvoorbeeld om een gebruikersaccount te hebben moet er:

Wat moet er gemaakt worden?

- Je hebt een lijst van features die moeten bestaan (MVP)

- Een feature heeft verschillende technische implicaties

- Bijvoorbeeld om een gebruikersaccount te hebben moet er:
  - Persistent data worden opgeslagen
  - Tijdelijk worden bijgehouden of de gebruiker is ingelogd
  - Een mogelijkheid zijn tot registreren
  - Een mogelijkheid zijn tot inloggen
  - Uitloggen / wachtwoord vergeten?

Wat moet er gemaakt worden?

- Persistente data
  - Een database tabel
- Bijhouden of een gebruiker is ingelogd
  - Cookies
- Een mogelijkheid tot registreren
  - Een registratie pagina
- Een mogelijkheid tot inloggen
  - Een inlog pagina?

Hoe kunnen we dat maken?

- From scratch

- Frameworks
  - Flask
  - Jinja2
  - Bootstrap

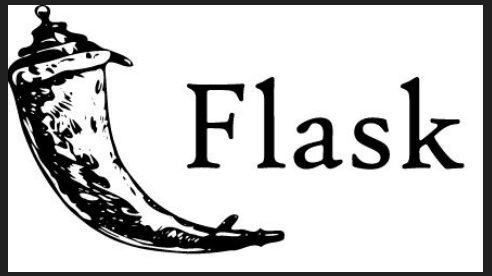
Frameworks

- Veel extra functionaliteit
  - rtfm

- Vraagt/forceert een bepaalde stijl van programmeren
  - Conventies

- Restricties

Frameworks - Flask

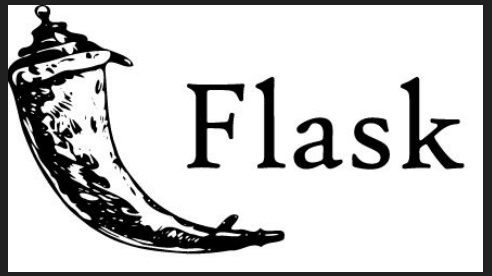


- Een micro framework voor webprogrammeren in Python

- Niet noodzakelijk voor webprogrammeren in Python

- https://github.com/pallets/flask

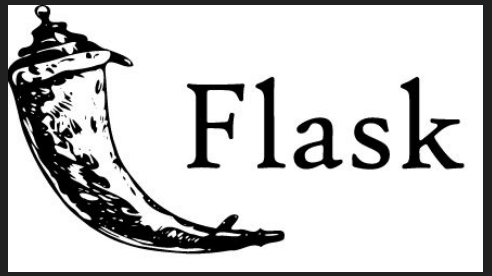
Frameworks - Flask



application.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "<html><header><title>Hello!</title></header><body>Hello world</body></html>"
```

Frameworks - Flask



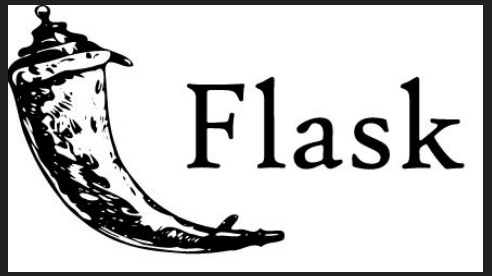
application.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")
```

templates/index.html

```
<html>
    <header>
        <title>Hello!</title>
    </header>
    <body>
        Hello world
    </body>
</html>
```

Frameworks - Flask



application.py

```
from flask import Flask, render_template, request

app = Flask(__name__)
db = ...

@app.route("/", methods = ["GET", "POST"])
def index():
     if request.method == "POST":
          db.query("INSERT INTO users (name) VALUES(:name)", name=request.form["name"])
     return render_template("index.html")
```

Frameworks - SQLAlchemy

models/user.py

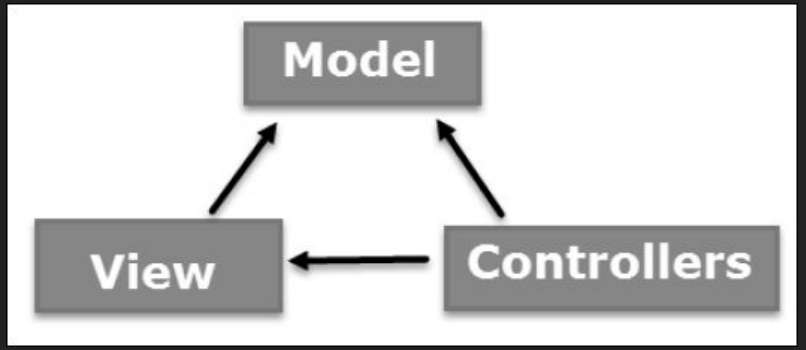
```
from application import db

class User(db.Model):
      __tablename__ = "users"
      id = db.Column(db.Integer, primary_key=True)
      name = db.Column(db.Text)
```

Frameworks - SQLAlchemy

application.py

```
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
from models.user import User

app = Flask(__name__)
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///some_database.db"
app.config["SQLALCHEMY_ECHO"] = True
db = SQLAlchemy(app)

@app.route("/", methods = ["GET", "POST"])
def index():
     if request.method == "POST":
          user = User(name = request.form["name"])
          db.session.add(user)
          db.session.commit()
     return render_template("index.html")
```

Design Patterns - MVC



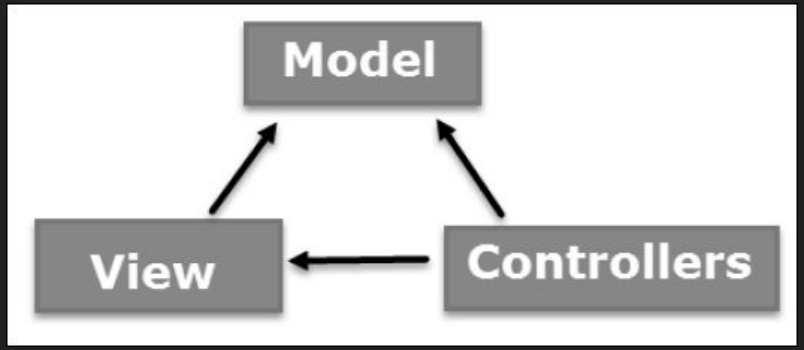
- Seperation of Concerns

- Model - Verantwoordelijk voor het omgaan met data
- Controller - Verantwoordelijk voor het serveren
- View - Verantwoordelijk voor de presentatie

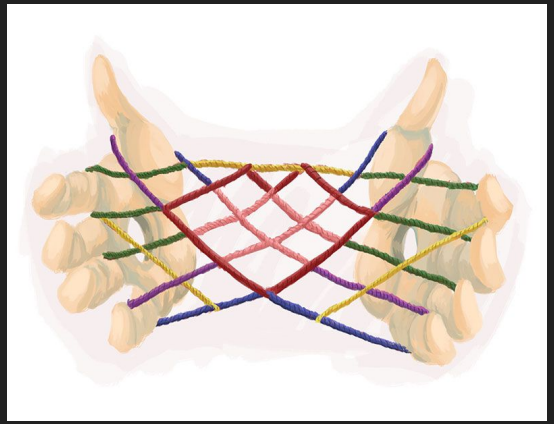
- But why?

Design Patterns - MVC

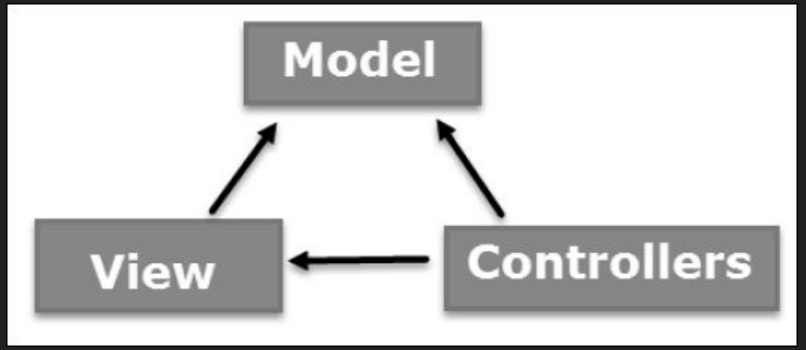
- Seperation of Concerns



- Model - Verantwoordelijk voor het omgaan met data
- Controller - Verantwoordelijk voor het serveren
- View - Verantwoordelijk voor de presentatie



MVC - Model

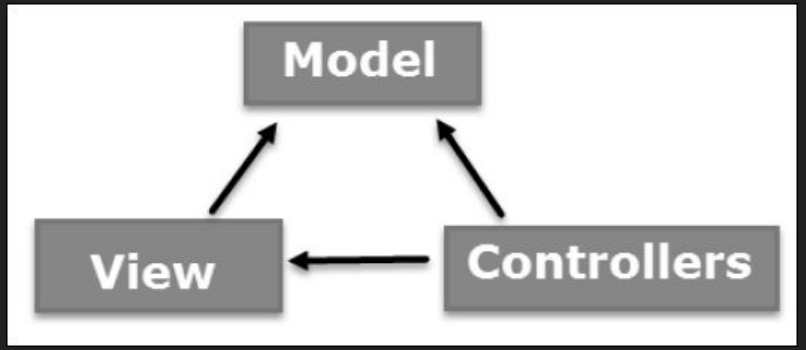


- Interne API voor het omgaan met data
  - Classes / methods / functions

- Onwetend over het bestaan van Controllers / Views
  - Geen kennis over presentatie
  - Geen plek voor het afhandelen van bijvoorbeeld foute URLs

- De enige plek voor domein kennis

MVC - Controller

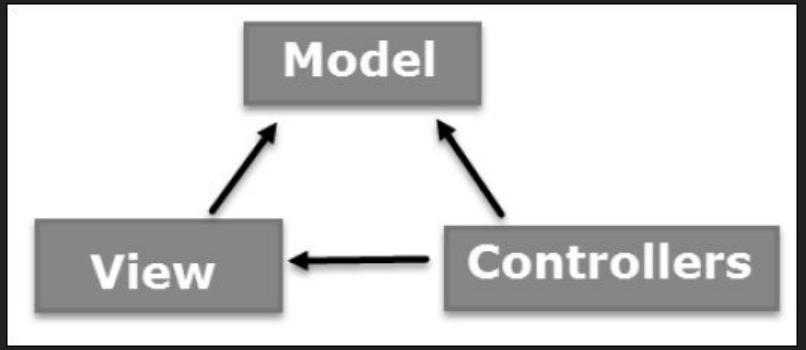


- Externe API van de applicatie
  - Routes

- Weet van het bestaan van Models en Views

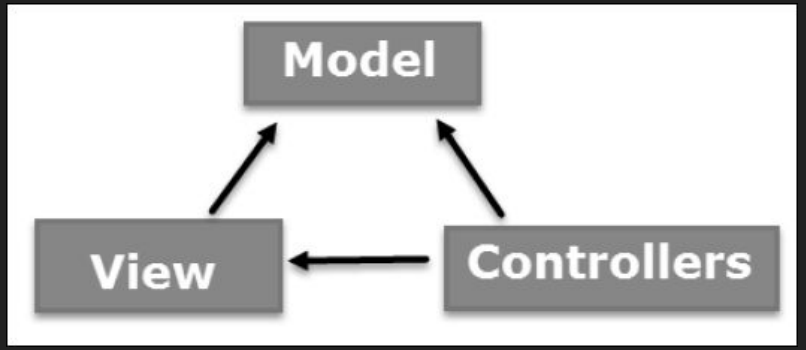
- Verantwoordelijk voor het afhandelen van externe verzoeken
  - Niet verantwoordelijk voor alles rondom presentatie
  - Bezit geen domein kennis
  - Heeft geen toegang tot de database

MVC - View



- Externe API van de applicatie
  - Knoppen, formulieren, links
- Weet van het bestaan van Models
- Verantwoordelijk voor de presentatie
  - En de gebruiker laten interacteren
  - Bezit geen domein kennis
  - Heeft geen toegang tot de database

MVC - Testen



- Werkt jouw applicatie zonder Views en Controllers?

- Met als kanttekening dat je precies weet welke functies je moet aanroepen van de models

MVC met Flask / Jinja2 / SQLAlchemy

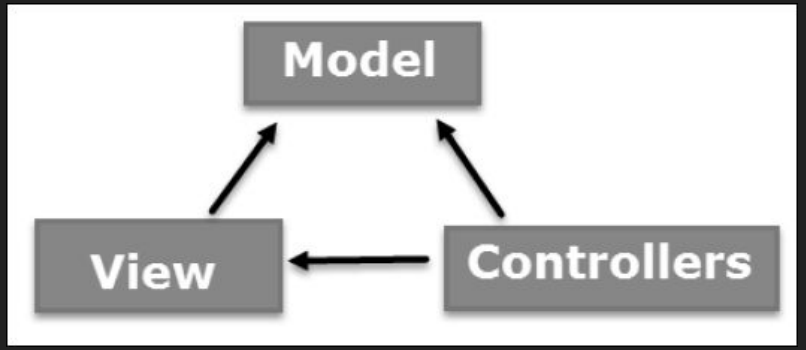
- Standaard is er één controller, application.py
  - Met daarin routes

- Models zijn losse .py bestanden (modules)
  - Met daarin Classes, methoden, en functies

- Views zijn een collectie van bestanden:
  - Templates in een map genaamd templates (conventie)
  - .css bestanden
  - Eventueel Javascript

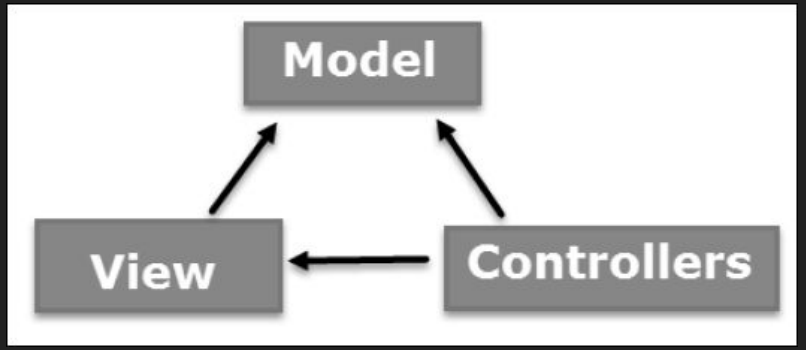
MVC - Feature: inloggen

- Model:
  - Een bestand models/users.py
  - Een class User.py
    - Met de attributen: id, naam
  - Een functie: login(name, password)
    - returned een User of None

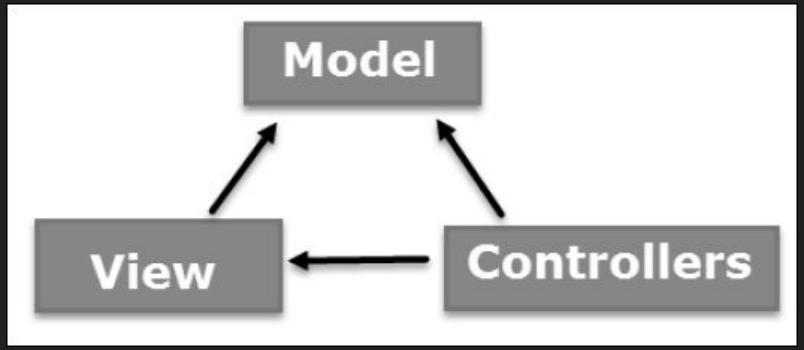


MVC - Feature: inloggen

- Model:
  - Een bestand models/users.py
  - Een class User.py
    - Met de attributen: id, naam
  - Een functie: login(name, password)
    - returned een User of None
- View:
  - Een template: templates/login.html
  - Evt. een stylesheet: login.css



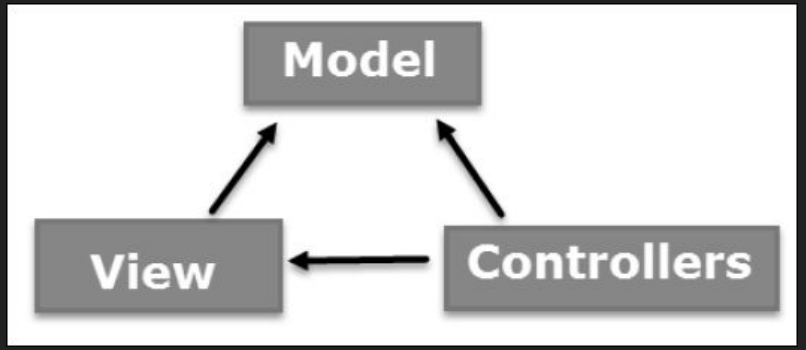
MVC - Feature: inloggen



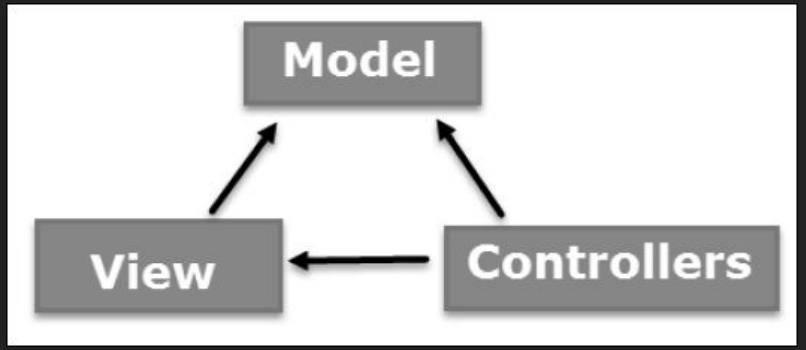
- Model:
  - Een bestand models/users.py
  - Een class User.py
    - Met de attributen: id, naam
  - Een functie: login(name, password)
    - returned een User of None
- View:
  - Een template: templates/login.html
  - Evt. een stylesheet: login.css
- Controller:
  - Een route: /login voor de methodes GET en POST
  - GET:
    - Laat login.html zien
  - POST:
    - Roep models.users.login() aan met de ingevulde naam / wachtwoord
    - Sla user_id in session op als inloggen is gelukt
    - Redirect naar /index.html als inloggen is gelukt, anders /login.html

MVC - Testen

- Model:
  - Een bestand models/users.py
  - Een class User.py
    - Met de attributen: id, naam
  - Een functie: login(name, password)
    - returned een User of None



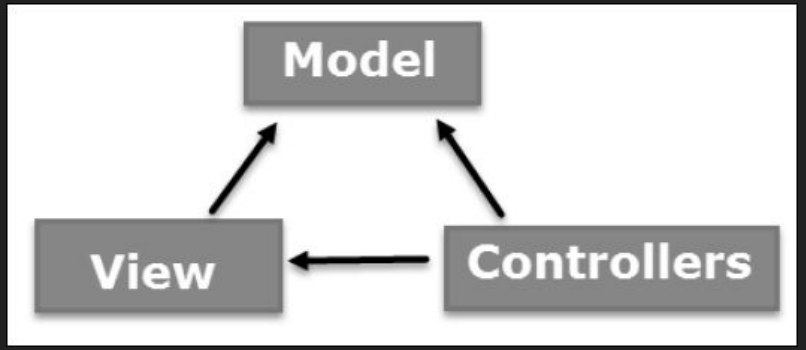
MVC - Feature: registreren



- Model:
  - In het bestand models/users.py
  - Een functie: register(name, password)
    - returned een User of None
- View:
  - Een template: templates/register.html
  - Evt. een stylesheet: register.css
- Controller:
  - Een route: /register voor de methodes GET en POST
  - GET:
    - Laat register.html zien
  - POST:
    - Roep models.users.register() aan met de ingevulde naam / wachtwoord
    - Sla user_id in session op als registreren is gelukt?
    - Redirect naar /index.html als registreren is gelukt, anders /register.html

MVC - Testen

- Model:
  - In het bestand models/users.py
  - Een functie: register(name, password)
    - returned een User of None



SQL - Database

- Welke tabellen heb ik nodig?

- Welke items hebben een relatie met elkaar?
  - Wat voor soort relatie?
    - One to One
    - One to Many
    - Many to Many

- Hebben items bijzondere eigenschappen?
  - Uniek?

Wat moet er gemaakt worden?

- Persistente data
  - Een database tabel
- Bijhouden of een gebruiker is ingelogd
  - Cookies
- Een mogelijkheid tot registreren
  - Een registratie pagina
- Een mogelijkheid tot inloggen
  - Een inlog pagina?

Hoe kunnen we samenwerken?

- Impliciet in MVC zorg je ervoor dat:
  - Taken duidelijk afgescheiden zijn
  - Communicatie tussen componenten overzichtelijk is

- Door af te spreken hoe componenten interacteren kun je parallel werken.
  - Besteed dus extra aandacht aan de interfaces (input / output)

Dynamische websites

- Alle navigatie is nu vrij ouderwets
  - Telkens een nieuwe webpagina laden voor alle communicatie tussen client en server

- Dat kan anders d.m.v. Javascript en AJAX

Javascript

- Een programmeertaal die draait in de browser
- Hate it or love it, er is geen alternatief
- Ontzettend veel plugins en frameworks

AJAX

- AJAX = Asynchronous Javascript And XML

Technisch Ontwerp - Wat ga je doen?

- https://webik.mprog.nl/project/technisch-ontwerp

- Werk vanuit je projectvoorstel uit welke M's, V's en C's er moeten bestaan
  - Waar bestaan ze (welke bestanden)
  - Welke functies / classes etc. hebben ze?
  - Wat moeten ze doen voor elke feature?

- Update je projectvoorstel

Verder deze week

- Dinsdag deadline C$50 Finance
- Woensdag college git (erg belangrijk!)
- Woensdag deadline Technisch Ontwerp
- Vanaf woensdag, programmeren aan de website! :)
- Donderdag of vrijdag voortgangsbespreking
- Dinsdag, woensdag, vrijdag laptopcollege

- Ook elke werkdag een standup
  - In de nabijheid van Science Park
  - **In de ochtend**