**CVF**

* **run.py**

Is de “main” van het programma en wordt vanuit hier gestart. Of het programma in debug-mode draait wordt hierin ook bepaald.

* **\_\_init\_\_.py**

Importeert de benodigdheden voor flask (flask, sqlalchemy, bcrypt en login). Verder wordt variabelen gedefinieerd en toegewezen die de debug-mode bepalen, bestandextensie, grootte en locatie van importbestanden bepalen. Ook de sessiesleutel voor de webapplicatie en de achterliggende database (SQLalchemy) wordt gedefinieerd.

* **site.sqlite3**
* Is het bestand/database waar de rijen en kollommen van gebruikers (gebruikersnaam en wachtwoord) bevat.
* **models.py**

Definieert de gebruikers-tabellen in de database (gebruikersnaam en wachtwoord). Ook maakt het mogelijk om de inhoud van deze tabellen te kunnen selecteren.

* **cfghulpengine.py (LEGACY)**

Hoort bij het CH (Configuratie Hulp) ontwerp van het project. Deze wordt NIET gebruikt door het prototype.

* **routes.py**

Is de kern van de flask webapplicatie en definieert de verschillende pagina’s waarnaar genavigeerd kan worden.

* + home()

Wanneer ingelogd wordt home.html ingeladen anders login.html.

* + account()

Wanneer ingelogd wordt account.html ingeladen. Hier is het mogelijk om aan de hand van het juiste wachtwoord in te vullen het huidige wachtwoord te wijzigen.

* + register()

register.html wordt ingeladen. Hier is het mogelijk om doormiddel een username en wachtwoord op te geven een account aan te maken.

* + login()

login.html wordt ingeladen. Wanneer de juiste gebruikersnaam/wachtwoord ingevuld is, wordt er ingelogd.

* + logout()

De ingebouwde functie van flask\_login, logout\_user(), wordt uitgevoerd waardoor de huidige gebruiker wordt uitgelogd.

* + configuratieimport()

Wanneer ingelogd wordt configuratieimport.html ingeladen. Een ingevoerd bestand wordt gecheckt op grootte via verifyFilesize(), lege bestandsnaam en valide extensie via verifyFilename(). Wanneer valide bevonden wordt het bestand opgehaald en doorgegeven aan de functies, readFile() (filehandler.py), cfgFileParsing() (parsingengine.py), deleteImportCache() (filehandler.py), genConfigToTableHTML() (parsingengine.py) en genCardMenus() (parsingengine.py). De resultaten van deze functies worden uitgevoerd in variabelen en doorgegeven aan configuratieimport.html

* + configuratiehulp() (LEGACY)

Hoort bij het CH (Configuratie Hulp) ontwerp van het project. Deze wordt NIET gebruikt door het prototype.

* **filehandler.py**

Leest, verifieert en verwijdert het geïmporteerde (configuratiebestand) bestand.

* + deleteImportCache()

Verwijderd het bestand uit de upload map die overeenkomt met de opgegeven bestandsnaam.

* + verifyFilesize()

Verifieert de bestandsgrootte

* + verifyFilename()

Verifieert de bestandsnaam door te kijken naar een valide extensie

* + readFile()

Leest het volledige bestand doormiddel een opgegeven bestandsnaam en geeft de inhoud terug in een variabele

* **parsingengine.py**
  + cfgFileParser()

analyseert de inhoud van een bestand regel-voor-regel, kijkt hier voor headers zoals “config”, “edit” en “set” en voegt de bijpassende waarde toe aan een dict die uitgevoerd wordt als json bestand.

* + deleteEmpty()

Verwijderd dicts die lege records bevatten

* + unquote()

Verwijderd quotes voor namen/waardes

* **userforms.py**
  + LoginForm()

Definieert de login velden (formulier) voor login.html

* + RegistrationForm()

Definieert de registratie velden (formulier) voor registreren.html

* + AccountChangeForm()

Definieert de account velden (formulier) voor account.html

* **viewengine.py**
  + jsonToHTMLTable()

Accepteert een json object en conventeert deze naar een html table.

* + jsonToAccordeon()

Accepteert een configuratieheader en een kleur en genereert een html table binnen een accordeon menu.

* + jsonToCardMenus()

Accepteert een json object en genereert de configuratiekaarten met een logo, kleur en ruimte voor inhoud. Wanneer er passende inhoud in configContentToCard() gevonden wordt, wordt deze bijgevoegd.

* + configContentToCard()

Accepteert een configuratieheader met bijbehorende data. Binnen deze functie zijn een aantal opmaken voor bepaalde configuratieheaders gedefinieerd. Wanneer een van deze overeenkomen met de ingevoerde header wordt deze opmaak gecreëerd en uitgevoerd.

* + findhighestGradeColor()

Bepaald de kleur met de belangrijkste gradatie, dit in de volgorde van; “danger” (rood), “warning” (geel), “success” (groen), en “info” (blauw).

* + jsonToHeadOveriew()

Accepteert een json object een genereert de header voor het overzicht doormiddel gebruikt te maken van de configuratieheaders “config” en “global”. Verder wordt er een afbeelding weergegeven van de bevonden firewall hardware.

* + getInterfaceAlias()

Haalt de “alias” op die hoort bij een gevonden interface

* + getFirewallServiceCustomPorts()

Haalt de gebruikte poorten voor bepaalde firewall services op, hiervoor wordt gebruik gemaakt van de “firewall service custom” configuratieheader.

* + getFirewallServiceMember()

Haalt de bijbehorende member (deelnemer) van een firewall service op, hiervoor wordt gebruik gemaakt van de “firewall service group” configuratieheader.

* + getPopoverButton()

Geeft een popoverknop terug. Titel, content en kleur wordt bijgevoegd wanneer deze gevonden wordt in getPopoverContents(). Een html string wordt teruggegeven als uitvoer.

* + getPopoverContents()

Leest “reference.json” en checkt of het opgegeven argument (configuratieheader) overeenkomt met een sleutel. Wanneer gevonden wordt de “title”, “content” en “color” opgeslagen in een variabele en teruggeven als uitvoer.

* + boostrapColorToCSSColor()

Converteert boostrap kleuren zoals “success” om naar kleuren zoals “green” zodat deze te begrijpen zijn door CSS.

* **reference.json**

Bevat pre-gedefinieerde toelichtingen en impact voor gevonden argumenten (configuratie/instellingen). Deze inhoud wordt gebruikt om de inhoud van getPopoverbutton() te vullen.