Plan van aanpak

Security Monitor Webpanel

Naam: Niels Hilgersom

Studentnummer: 500644696

Docent: Arnim Eijkhoudt

Datum: 23/10/2014

Versie: 2.0

Inhoudsopgave

[Inleiding 3](#_Toc402137981)

[1. Het security monitor project 4](#_Toc402137982)

[2. De opdrachtgever 5](#_Toc402137983)

[3. Research 6](#_Toc402137984)

[3.1 Python en het web 6](#_Toc402137985)

[3.1.1 Common gateway Interface (CGI) en fastCGI 6](#_Toc402137986)

[3.1.2 Mod\_python 6](#_Toc402137987)

[3.1.3 WSGI 7](#_Toc402137988)

[3.1.4 Conclusie python en het web 7](#_Toc402137989)

[3.2 Python webframeworks 7](#_Toc402137990)

[3.2.1 Micro frameworks 8](#_Toc402137991)

[3.2.2 Conclusie micro frameworks 9](#_Toc402137992)

[3.3 Grafische elementen en licenties 9](#_Toc402137993)

[4. Milestones 10](#_Toc402137994)

[5 Planning 11](#_Toc402137995)

[Bronnen 12](#_Toc402137996)

# Inleiding

De komende periode zal ik mij bezig houden met het ontwikkelen van een webinterface voor de al reeds bestaande Security Monitor applicatie. In dit document zal ik mij richten op de eisen van de opdrachtgever, de planning voor het project en op het vooronderzoek. Na goedkeuring van dit document zal ik mij gaan richten op de daadwerkelijke ontwikkeling van de python webapplicatie. Mocht er in dit plan van aanpak (PvA) onjuiste informatie staan of mochten de aanbevelingen niet worden overgenomen door de opdrachtgever, dan zal ik dit document op een later moment nog aanpassen.

Ik heb dit PvA in 4 hoofdstukken opgedeeld. Hoofdstuk 1 geeft een korte introductie van het project zoals deze eerder al is opgesteld door mijn voorgangers in het Security Monitor project. In hoofdstuk 2 zal ik de eisen van de opdrachtgever beschrijven zoals deze nu bij mij bekend zijn. Hoofdstuk 3 is een technisch hoofdstuk met achtergrondinformatie over mijn vooronderzoek met aanbevelingen. Dit hoofdstuk beschrijft feitelijk hoe ik het project wil aanpakken en waarom ik bepaalde keuzes heb gemaakt.



# **Het security monitor project**

De applicatie SecMon is een product wat is voortgekomen uit het project Security Monitor, onderdeel van de minor Forensic Intelligence and Security aan de Hogeschool van Amsterdam.

Het project is voor het eerst in 2013 gemaakt door Dave van Ast, Stanley Numan, Jasper Groot, Sujen Kandasamy en Vinesh Koenjbiharie. In tweede helft van het studiejaar 2013/2014 is het project door Christiaan Druif aangepast en verbeterd. In de eerste helft van het studiejaar 2014/2015 zal het project verder worden uitgebreid met een functionerende webinterface door Niels Hilgersom.

Het doel van het project was het opleveren van een applicatie die de VMWare omgeving van de Hogeschool van Amsterdam kan beschermen tegen cyber-aanvallen van zowel binnen uit als buitenaf. De applicatie moet in staat zijn verschillende aanvallen te herkennen en daarop volgend een actie uit te voeren. Dit moet gerealiseerd worden door middel van het parsen van de firewall log(s). Daarna is de applicatie uitgebreid waardoor het nu ook mogelijk is om andere logfiles, zoals Apache logfiles te parsen.

# De opdrachtgever

Tijdens een eerder gesprek zijn er een aantal wensen besproken met de opdrachtgever, deze wensen zijn een verdere specificatie op het al eerder door de examencommissie goedgekeurde project document. Zo staat er in het projectdocument dat ik een webpanel zou ontwikkelen maar niet welke eisen er aan dit web panel zijn verbonden. Belangrijke aspecten van het webpanel zoals de opdrachtgever deze heeft aangegeven zal ik nu bespreken.

Belangrijke eisen voor de fundering van de applicatie

* De python applicatie moet op de server volledig worden geprogrammeerd in python.
* De python applicatie moet werken op python versie 2.6 en hoger.
* De python applicatie moet “lightweight” blijven.
* De python applicatie moet zo min mogelijk “3th party dependencies” gebruiken, bij voorkeur enkel alleen python 2.6 en hoger en geen andere dependencies.

Belangrijke eisen voor de functionaliteit van de applicatie

* De python applicatie moet de mogelijkheid bieden om rules weer te geven. Wie heeft welke rule op welk moment getriggerd?
* De python applicatie moet de mogelijkheid hebben om rules toe te voegen, te wijzigen en te verwijderen.
* De python applicatie moet een authenticatie mogelijkheid hebben om zo het webpanel te beschermen tegen personen die hier geen toegang toe zouden moeten hebben.

Belangrijke eisen voor het gedrag van de applicatie.

* De python applicatie moet makkelijk aan- en uitgezet kunnen worden.
* De python applicatie moet in een proces draaien op de server op de achtergrond.

Toegestane hulpmiddelen en tools.

* Er mag gebruik worden gemaakt van SQLlite bij het ontwikkelen van de authenticatie functionaliteit.
* Voor de client mag gebruik worden gemaakt van verschillende programmeertalen en libraries zoals bijvoorbeeld HTML, CSS, javascript, jQuery en Twitter bootstrap. Uiteraard moet dit binnen acceptabele grenzen vallen van de “lightwight” eis van de opdrachtgever.

Dit zijn vooralsnog de belangrijkste punten om rekening mee te houden. Na goedkeuring van dit PvA document zijn dit de eisen en de grenzen waar binnen ik de python webapplicatie zal ontwikkelen.

# Research

Om een webapplicatie te ontwikkelen in python is research nodig want er zijn een aantal aspecten waar je als ontwikkelaar rekening mee moet houden. Ook had ik, voor het schrijven van dit document, nog geen idee hoe ik dit moest aanpakken. Belangrijkste aspect is de visie van de opdrachtgever. Deze stelt technische eisen aan de ontwikkeling van de webapplicatie waar ik mij op zal moeten richten. Ook op technisch vlak is het belangrijk om goed onderzoek te doen naar dit aspect, zo kan ik eventuele problemen voor zijn. In dit hoofdstuk richt ik mij dan ook vooral op het vooronderzoek voor de python webapplicatie. Wat kan wel, wat kan niet? Waar begin ik en hoe ga ik verder?

## Python en het web

Om een python applicatie te kunnen gebruiken op een UNIX of Windows server, is een webserver vereist welke de programmeertaal python begrijpt. Door de juiste functies en variabelen te gebruiken, kan de server http requests afhandelen en hier een response op geven; dit gebeurd feitelijk wanneer een gebruiker gebruik maakt van door mij ontwikelde python webpanel. Er zijn verschillende technieken beschikbaar die precies dit doel hebben zoals ook terug is te lezen in de officiële python documentatie[[1]](#footnote-1). Python heeft standaard “out-of-the-box” ondersteuning voor een aantal verschillende technieken die zorgen voor een goede request en response afhandeling. Er zijn vier verschillende technieken die veel worden gebruikt namelijk; CGI, fastCGI, mod\_python en WSGI. Ik zal deze technieken in de komende alinea’s beschrijven met de voor- en nadelen.

### Common gateway Interface (CGI) en fastCGI

Een webapplicatie in ontwikkelen d.m.v. een CGI module heeft één groot voordeel: het is de simpelste methode is om een python webserver op te zetten. Het is ook de oudste methode en om die reden is deze techniek standaard door bijna elke UNIX en Windows server met python ondersteund wat weer mooi overeen komt met de gestelde eisen van de opdrachtgever. Het opzetten van een python server die requests kan afhandelen in CGI vergt slechts enkele simpele regels aan code. De CGI module spreekt bij elke request de CGI handler aan wat resulteert in een nieuwe python interpreter bij elke http request, een soort nieuw proces dus. Nadeel van deze methode is dat het lang kan duren voor de server een reactie geeft op de requests omdat het tijd kost om telkens een nieuwe python interpreter te starten waar python in geladen zal worden. Dit is de prijs die men betaald voor de relatief eenvoudige code voor een CGI webapplicatie. Het is niet aanbevolen om in deze methode te conform de officiële python documentatie om de zojuist besproken reden.

Om de de CGI methode te versnellen is er een variant ontwikkeld namelijk fastCGI. Zoals de naam al aangeeft zou deze methode sneller moeten zijn dan de CGI methode en daarmee de agilis hiel van de techniek moeten verhelpen. FastCGI werkt technisch gezien hetzelfde als CGI met één groot verschil; bij fastCGI blijft er altijd één proces op de achtergrond lopen. Door dit proces is het mogelijk om de server te ontlasten en sneller responses te geven, de server start namelijk niet telkens de python interpreter op maar doet dit vanuit het al lopende proces.

### Mod\_python

Mod\_python werkt technisch gezien heel anders dan CGI en fastCGI. Deze techniek integreert zichzelf in het Apache proces om zo webpagina’s in python code te serveren. Het Apache proces loopt standaard op de achtergrond van het besturingssysteem en is gespecialiseerd in requests en responses, wat resulteert in een snelle afhandeling van de webapplicatie. Deze methode werkt overigens anders dan bijvoorbeeld PHP omdat het niet is toegestaan om HTML code te combineren met python waar dat met CGI en fastCGI wel mogelijk is. Wel is het mogelijk om gebruik te maken van een template engine wat de moeilijkheidsgraad van het project verder opschroeft t.o.v. CGI en fastCGI.

Er zijn echter nog een aantal andere nadelen verbonden aan deze methode. Zo maakt de mod\_python standaard gebruik van caching waardoor het Apache proces na elke wijziging opnieuw zal moeten starten. Ook maakt Apache standaard gebruik van zogenoemde “child processess” welke bij gebruik van mod\_python de volledige python interpreter moeten inladen, ook als deze niet worden gebruikt. Dit kan al snel zorgen voor een hoge belasting van de server en kan dus worden gezien als kwetsbaarheid, vergelijkbaar aan CGI en fastCGI. Mod\_python werkt verder enkel alleen met Apache en moet opnieuw worden gecompileerd bij elke upgrade. Dit wil zeggen dat een eventuele update aan Apache of python er voor zorgt dat de webapplicatie niet meer zal functioneren.

### WSGI

De zojuist besproken technieken zijn allemaal “low level gateways” (LLG’s). Vaak werken deze LLG’s samen met andere software zoals Apache om requests af te handelen, vaak gepaard met verschillende nadelen en complicaties. WSGI is een nieuwere techniek welke probeert deze problemen op te lossen door een zelfstandig proces aan te maken zonder extra eisen te stellen aan de server zoals bijvoorbeeld de aanwezigheid van Apache. WSGI staat voor “Web Server Gateway Interface”. Voordeel van deze methode is dat de server optimaal kan functioneren zonder de eerer genoemde negatieve punten. Er kleeft echter ook een groot nadeel aan WSGI en dat betreft de complexiteit van de code. Om een WSGI webserver te programmeren is veel kennis nodig over de structuur en de werking van de WSGI module in Python. De python documentatie schrijft hier het volgende over:

The *Web Server Gateway Interface*, or WSGI for short, is defined in [**PEP 333**](http://www.python.org/dev/peps/pep-0333) and is currently the best way to do Python web programming. While it is great for programmers writing frameworks, a normal web developer does not need to get in direct contact with it. When choosing a framework for web development it is a good idea to choose one which supports WSGI.

The big benefit of WSGI is the unification of the application programming interface. When your program is compatible with WSGI – which at the outer level means that the framework you are using has support for WSGI – your program can be deployed via any web server interface for which there are WSGI wrappers. You do not need to care about whether the application user uses mod\_python or FastCGI or mod\_wsgi – with WSGI your application will work on any gateway interface. The Python standard library contains its own WSGI server, [wsgiref](https://docs.python.org/2/library/wsgiref.html#module-wsgiref), which is a small web server that can be used for testing.

### Conclusie python en het web

Anders dan ik voor mijn research had ingeschat lijkt het gebruik van bestaande webservers zoals Apache, Engine X of lighthttpd niet de beste optie om python http requests af te handelen. Ik zou dan ook sterk afraden om een andere techniek te gebruiken dan WSGI. Dit gezegd hebbende stel ik ook voor om WSGI te gebruiken als basis voor mijn python webapplicatie om zo de zojuist genoemde technische struikelblokken te ontwijken.

Om gebruik te kunnen maken van WSGI server is een framework echter wel noodzakelijk. Het uitpluizen van de WSGI structuur zou een project op zichzelf zijn en daarbij is het begrijpen van de WSGI structuur geen vereiste voor dit project. In het volgende hoofdstuk zal ik meer schrijven over de verschillende Python frameworks.

## Python webframeworks

Er zijn op het internet een groot aantal verschillende frameworks te vinden[[2]](#footnote-2). Er zitten grote verschillen tussen de verschillende frameworks. Op internet maakt men onderscheid tussen full-stack frameworks en micro frameworks. Full-stack frameworks bevatten vaak veel meer functionaliteit en structuur dan de doorgaans minder gebruikte micro frameworks. Full-stack frameworks worden vaker ingezet bij grotere projecten en websites i.v.m. de ingebouwde functionaliteiten zoals bijvoorbeeld authenticatie systemen, database helpers en template engines. Populaire full-stack frameworks zijn o.a. Django, TurboGears en web2py. Veel van deze frameworks zijn niet bepaald lightweight te noemen gezien de aantal regels code in de broncode. Full-stack frameworks zoals Django, TurboGears en web2py zie ik dan ook niet als optie gezien de omvang van deze frameworks.

Micro frameworks aan de andere kant hebben vaak minder functionaliteit en richten zich meer op de basis functionaliteit die ontwikkelaars zoeken. In het geval van een WSGI framework zal dit vaak onder andere routing zijn. Welke URL wijst naar welke pagina? Wat gebeurd er als een gebruiker een URL opent die niet bestaat? Ook functionaliteiten zoals het starten en stoppen van de webserver, het loggen van requests en een template engine staan vaak hoog op het lijstje van ontwikkelaars. Vaak moeten er bij micro frameworks echter wel concessies worden gedaan in de out-of-the-box functionaliteit en structuur. Enkele populaire micro frameworks zijn Flask, Bottle en CherryPy maar er zijn er nog veel en veel meer.[[3]](#footnote-3)

Frameworks zijn geschreven op basis van één of meerdere van de al eerder besproken python webservers. Zo heb je frameworks die moeten worden geïntegreerd met Apache (vaak mod\_python of fastCGI) en frameworks die op zichzelf staan (vaak WSGI).

### Micro frameworks

Om het juiste framework te vinden zijn er een aantal dingen belangrijk.

* Het framework moet lightweight zijn.
* Het framework moet een goede documentatie hebben zodat ik tijdens het ontwikkelen de juiste informatie kan opzoeken.
* Het framework moet zijn gebaseerd op WSGI.
* Het framework moet geen grote inpact kunnen hebben op de server.
* Het framework moet de voor mij belangrijke basisfunctionaliteiten zoals routing, logging en een daemon functionaliteit hebben.
* Het framework moet actief worden onderhouden c.q. niet verouderd zijn in de code.
* Er mogen geen ernstige beveiligingslekken in het framework zitten code die mogelijk een beveiligingsrisico zou kunnen vormen.

Het kost veel tijd om alle frameworks door te spitten. Na wat research kwam ik een tabel tegen die dateert van 22

augustus 2011 met het aantal regels code per framework. Ook heb ik verschillende youtube filmpjes gekeken waar

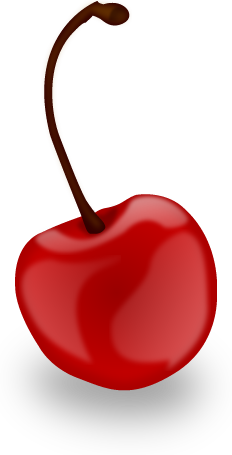
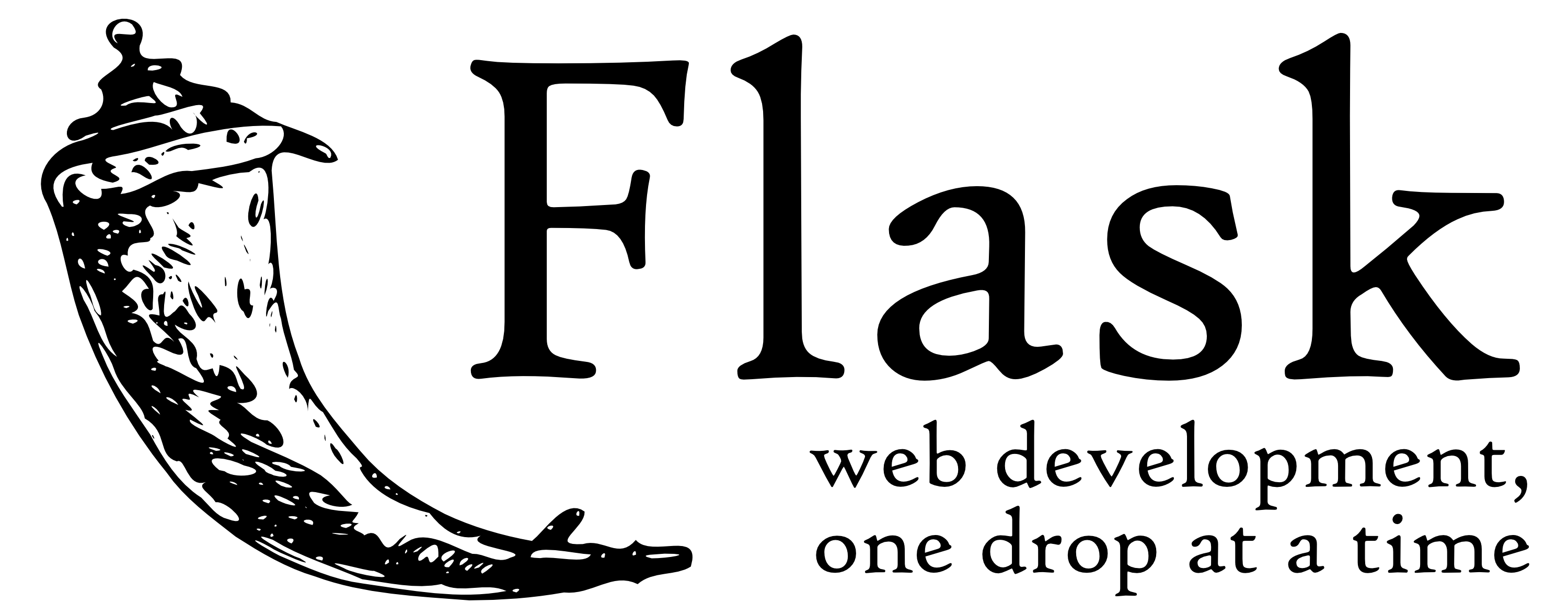
sprekers drie micro frameworks aanduidde als populair namelijk bottle, cherrypy en flask. Deze frameworks maken

allemaal gebruik van de WSGI functionaliteit van python.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Framework | Laatste versie | Laatste update | Omvang van .zip installatie | Actief in ontwikkeling? | Website |
| Bottle | 0.12.7 | 29/04/2014 | 67kb | Ja | www.bottlepy.org |
| Cherrypy | 3.6.0 | 14/09/2014 | 481kb | Ja | www.cherrypy.org |
| Flask | 0.10.1 | 14/06/2014 | 628kb | Ja | flask.pocoo.org |

|  |  |
| --- | --- |
| Framework | Regels code |
| Itty | **593** |
| Cgi+wsgiref | **2441** |
| Bottle | **2697** |
| Aspen.io | **5600** |
| Pesto | **6649** |
| Bobo | **8044** |
| Web.py | **10198** |
| Werkzeug | **16563** |
| Cherrypy | **27768** |
| Flask | **32245** |

*Tabel 2.3.1.1: wat is het kleinste python micro framework? Datum gegevens 22/08/2011, algemene indicatie.*

*Logo’s van drie veel gebruikte micro frameworks.*

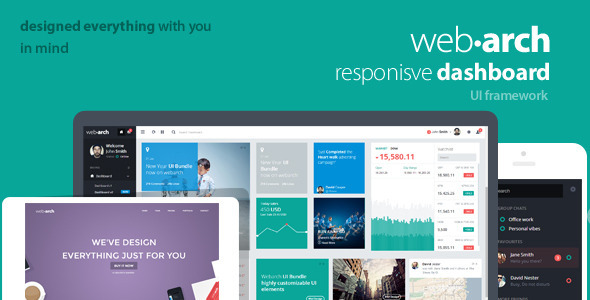
### Conclusie micro frameworks

Voor de frameworks bottle, cherrypy en flask is er goede documentatie beschikbaar. Bijzonder is het bottle framework. Dit is een bijzonder klein framework wat uit enkel één enkel bestand bestaat. Dit bestand kan worden aangeroepen in de command line interface d.m.v. een simpel script. Na wat onderzoek lukte het mij al snel om een WSGI webserver als een daemon te laten werken op een poort naar keuze dankzij het bottle framework. De belangrijkste functionaliteit in dit framework is aanwezig en goed gedocumenteerd. Sommige functionaliteit zal ik wel handmatig moeten programmeren zoals een inlog functionaliteit (authenticatie), deze is niet standaard opgenomen in het bottle framework. Na wat wikken en wegen lijkt bottle mij de beste optie voor dit project en dus zou ik het bottlepy framework graag willen inzetten voor dit project.

## Grafische elementen en licenties

Tijdens het programmeren zal ik nog tegen verschillende dingen aanlopen zoals dat zo vaak gaat. Ik kan veilig zeggen dat ik tijdens de uitvoering van het project ook nog gebruik ga maken van verschillende javascript frameworks zoals bijvoorbeeld het jquery framework. Ook visuele componenten die ik zal inzetten vereisen vaak frameworks en externe code.

Voor het uiterlijk heb ik verschillende thema’s bekeken die kunnen worden gebruikt als back-end. Ik heb hier gekozen voor het WebArch thema[[4]](#footnote-4). Dit thema voorziet in alle mogelijke visuele elementen die ik nodig zou kunnen hebben voor het webpanel en is erg flexibel. Na contact met de makers van dit thema werd mij ook duidelijk dat iedereen de code van dit thema mag herdistribueren op het internet mits men hier geen geld voor vraagt. Zo blijft het dus mogelijk om het project te uploaden op bijvoorbeeld github zonder problemen te krijgen met licenties en rechten. Ik heb een licentie in mijn bezit en kan dus gebruik maken van dit thema.



# Milestones

Gedurende het project zal ik meerdere malen de huidige voortgang opleveren aan de opdrachtgever, deze fases noem ik voor het gemak milestones. Bij de oplevering van deze milestones bestaat voor de opdrachtgever continu de mogelijkheid om bij te sturen mocht dit nodig zijn. Afgaande op de eisen van de opdrachtgever, zoals opgesteld in hoofdstuk 2, stel ik de volgende technische milestones voor welke ik zal opleveren.

Milestone 1: Basis

* Server
  + Mogelijkheid om de webserver te starten en te stoppen op een poort naar keuze via de CLI.
  + De webserver moet op in een proces draaien op de server.
* Client
  + De webserver moet content tonen in de browser op de client.
  + Opmaak van de pagina’s is nog niet belangrijk.

In deze milestone richt ik mij op de onderliggende structuur van het gekozen framework. Veel frameworks werken volgens het model-view-controller principe[[5]](#footnote-5). Ook het door mij aanbevolen Bottle framework maakt gebruik van dit principe[[6]](#footnote-6). Dit principe splitst de daadwerkelijke server code (model) met de client code (view) om dit vervolgens weer samen te voegen (controller) tot de daadwerkelijke applicatie die de gebruiker ziet.  
  
Ik ben bekend met dit ontwerppatroon en heb hier ervaring mee. Echter is de daadwerkelijke implementatie van het MVC-model per framework vaak afwijkend en dus is het verstandig om een basis opzet te creëren met een paar simpele models, views en controllers. Het idee hiervan is dat nieuwe code direct in deze logica kan worden geprogrammeerd en correct worden geïmplementeerd zonder later drastische wijzigingen te moeten maken. Op technisch vlak zal ik deze structuur dus implementeren op de server.

Milestone 2: Authenticatie

* Server
  + Meerdere pagina’s creëren.
  + Pagina’s beveiligen met een log-in systeem.
  + Opmaak van de pagina’s is nog niet belangrijk.

Milestone 3: Log weergave

* Server
  + Functionaliteit toevoegen om (getriggerde) rules weer te geven in een lijstweergave.
  + Opmaak van de pagina’s is nog niet belangrijk.

Milestone 4: Rules toevoegen, wijzigen en verwijderen

* Server
  + Functionaliteit toevoegen om rules te verwijderen, te wijzigen en volledig opnieuw toe te voegen.
  + Opmaak van de pagina’s is nog niet belangrijk.

Milestone 5: Bugs en opmaak

* Server
  + Bugs verhelpen.
* Client
  + Opmaak toevoegen / verbeteren.
* Overige
  + Documentatie schrijven zoals de reflectie, handleiding e.t.c.

5 Planning  
De zojuist genoemde milestones en de audit datum vormen de basis voor mijn planning gedurende de komende periode. Ik heb zelf een schatting gemaakt van de moeilijkheidsgraad van elke milestone met bijbehorende opleverdatum afgaande op goedkeuring van dit PvA. Bij een eventuele afwijzing zal ik mijn planning herzien.

|  |  |  |
| --- | --- | --- |
| Onderwerp | Datum | Tijd |
| Oplevering milestone 1 | 10/11/2014 | Digitaal voor 17:00 |
| Oplevering milestone 1 en voortgang milestone 2 bespreken. | 17/11/2014 | Tussen 16:00 en 17:00 |
| Oplevering milestone 2 | 24/12/2014 | Digitaal voor 17:00 |
| Voortgang milestone 3 bespreken | 01/12/2014 | Tussen 16:00 en 17:00 |
| Oplevering milestone 3 | 08/12/2014 | Digitaal voor 17:00 |
| Voortgang milestone 4 bespreken | 15/12/2014 | Tussen 16:00 en 17:00 |
| Kerstvakantie | 20/12/2014 tot 04/01/2015 | - |
| Oplevering milestone 4 | 12/01/2015 | Digitaal voor 17:00 |
| Voortgang milestone 5 bespreken | 19/01/2015 | Tussen 16:00 en 17:00 |
| Oplevering milestone 5 | 26/01/2015 | Digitaal voor 17:00 |
| Audit | 28/01/2015 | n.n.b. |

Deze planning geeft een algemene indicatie maar kan afwijken omdat ik zeer waarschijnlijk op bepaalde punten vooruit zal werken en mogelijk op sommige punten tegen problemen aan zal lopen. Dit kan op besprekingen worden bijgestuurd.

# Bronnen

**Python (micro) frameworks:**  
<https://www.youtube.com/watch?v=AYjPIMe0BhA>

<http://codecondo.com/14-minimal-web-frameworks-for-python/>

<https://wiki.python.org/moin/WebFrameworks>

**Python webserver achtergrond:**

<https://docs.python.org/2/howto/webservers.html>

1. https://docs.python.org/2/howto/webservers.html [↑](#footnote-ref-1)
2. https://wiki.python.org/moin/WebFrameworks [↑](#footnote-ref-2)
3. http://codecondo.com/14-minimal-web-frameworks-for-python/ [↑](#footnote-ref-3)
4. http://themeforest.net/item/webarch-responsive-admin-dashboard-template/6157416 [↑](#footnote-ref-4)
5. https://nl.wikipedia.org/wiki/Model-view-controller-model [↑](#footnote-ref-5)
6. https://www.digitalocean.com/community/tutorials/how-to-use-the-bottle-micro-framework-to-develop-python-web-apps [↑](#footnote-ref-6)