miLody

**Technische documentatie**

Lysander.vogelzang@meertens.knaw.nl

Meertens Instituut

2012/10/16

Inhoudsopgave

1. Inleiding 4

1. Achtergrond 5

1.1. Inleiding 5

1.2. Eisen voor gebruikers 5

1.3. Eisen wat betreft techniek 5

1.4. Randvoorwaarden 5

2. Informatie Infrastructuur 6

2.1. Inleiding 6

2.2. Bronnen 6

2.3. Toegang tot bronnen 6

3. Technische blauwdruk 10

3.1. Inleiding 10

3.2. miLody: de backend 10

3.2.1. Overzicht 10

3.2.2. De basis: de generieke klassen 11

3.2.3. Afgeleiden van de basisklassen 12

3.2.4. Singleton (clsRestron) 12

3.2.5. Gebruikersgedefinieerde collecties 13

3.2.6. Muzikale gebruikersinvoer verwerken 13

3.2.7. Exceptions 15

3.3. Alignment Server 16

3.3.1. Inleiding 16

3.3.2. Communicatie framework en alignment server 16

3.3.3. Daemon – Spawn constructie: fork() 16

3.3.4. Response 16

3.3.5. Kernfuncties 17

3.3.5.1. Main 17

3.3.5.2. createIndex() 17

3.3.5.3. Listen() 17

3.3.5.4. ParseMessage() 17

3.3.5.5. doMatchMidi() en doMatchNLB() 17

3.4. Gebruikersinterface 18

3.4.1. Uitvoermogelijkheden 18

3.4.2. Uitvoernotatie 18

3.4.3. Gereserveerde GET parameters 18

4. Collectie Speelmuziek 20

4.1. Inleiding 20

4.2. Dataprovider 20

4.2.1. Instellingen 20

4.2.2. getTuneById() 20

4.2.3. getTunes() 20

4.3. Gebruikersinterface 21

4.3.1. Overzicht 21

4.3.2. Meertaligheid 21

4.3.3. Interface voor muzikale gebruikersinvoer: virtuele piano 21

4.3.4. Interface voor muzikale gebruikersinvoer: neuriën 22

4.4. Muzikale gebruikersinvoer 22

4.4.1. Inleiding 22

4.4.2. Uploaden MIDI-bestand 22

4.4.3. Virtuele piano 22

4.4.4. Neuriën 23

# Inleiding

Op 1 september 2011 is begonnen aan het ontwikkelen van een zoekmachine om de *Speelmuziek* collectie van het Meertens Instituut te ontsluiten. Het doel was om met als basis van het proefschrift van Peter van Kranenbrug, en de daarbij horende programmacode, een interface te bieden voor het vergelijken van grote aantallen melodieën. In lijn met de ambitie van het Meertens Instituut om ontwikkelde applicaties gemakkelijk te delen met andere instituten, was het een vereiste dat de ontwikkelde applicatie modulair van opzet, zodat andere instituten op een gemakkelijke wijze kunnen profiteren van de inspanningen van het Meertens Instituut. Na tien maanden werk wordt dit opgeleverd: miLody.

Er is een drietal zaken ontwikkeld: een raamwerk voor het ontsluiten van collecties van liederen in het algemeen, een implementatie hiervan voor de *Speelmuziek* collectie en een alignment server voor het vergelijken van liederen.

Het raamwerk is de downloaden via de instructies op <https://sourceforge.net/p/milody/code/> , de bijhorende alignment server is te vinden op <https://sourceforge.net/p/milody-as/code/> .

In dit document wordt de techniek achter de ontwikkelde applicaties besproken. In het eerste hoofdstuk wordt de achtergrond van het project kort besproken en worden de vereisten genoemd die leidend waren voor keuzes wat betreft techniek tijdens de ontwikkeling. Vervolgens wordt de infrastructuur voor de ontsluiting van de beschikbare data toegelicht. In het vierde hoofdstuk wordt de techniek van het raamwerk en de aligment server uitvoerig behandeld. Hierbij wordt zowel aandacht besteed aan de samenhang van de onderdelen als de belangrijkste functies binnen elke module. Het laatste hoofdstuk behandelt de voor het raamwerk geschreven module voor *Speelmuziek,* alsmede de modules voor muzikale gebruikersinvoer die hiervoor zijn geschreven.

Naast deze technische handleiding is er ook een beheerdershandleiding beschikbaar, waarin een toelichting wordt gegeven op het geschikt maken voor gebruikers van miLody.

Ter verduidelijking van deze handleiding wordt gebruik gemaakt van speciale opmaak:

* Engelse, technische termen die niet zijn vertaald zijn weergeven in het lettertype Courier: Engelse computerterm
* Termen uit de code van miLody die worden gebruikt, zijn weergegeven in lettertype Courier en bovendien grijs gearceerd: miLody code
1. **Achtergrond**

##  Inleiding

Het Nederlands Muziek Instituut te Den Haag verwierf in 2006 een unieke verzameling Nederlandse speelmansboeken uit de 18e en 19e eeuw. Deze collectie, bijeengebracht door de wiskundige en musicoloog dr. Christiaan C. Vlam (1916-1999), bestaat uit 21 Nederlandse volksmuziek-handschriften uit de periode 1700-1900. Het zijn handschriften waarin violisten, fluitisten en andere speellieden het repertoire noteerden waarmee zij hun brood verdienden. Deze manuscripten vormen de belangrijkste bron van onze kennis van de populaire muziek van die tijd. Musicoloog Prof. Dr. Louis Grijp stelde in 2006 vast dat de cultuurhistorische waarde van de collectie Vlam onmiskenbaar is, al is het alleen maar om het aantal unieke melodieën dat de collectie bevat.

In het project 'Speelmuziek' werd de collectie Vlam gescand en online toegankelijk gemaakt op de website van het NMI. Omdat de collectie verband houdt met contemporain liedmateriaal, is zij tevens ontsloten in de Nederlandse Liederenbank van het Meertens Instituut,[[1]](#footnote--1) een database die liederen en melodieën op talrijke manieren doorzoekbaar maakt. Met hulp van vrijwilligers zullen vele melodieën muzikaal doorzoekbare notatie worden toegevoegd. Speelmuziek zoals in de collectie Vlam is een namelijk een waardevolle bron voor de grote groep zich op volksmuziek richtende musici in Nederland en Vlaanderen, een groep die uit honderden, zo niet duizenden, mensen bestaat. Deze actieve muzikanten hebben de expertise om de handschriften te interpreteren: zij kunnen als geen ander begrijpen wat de 18e- en 19e-eeuwse speellieden hebben bedoeld. In samenwerking met vrijwilligerscoördinator Mark van Nieuwstadt wordt de website van de Nederlandse Liederenbank uitgebreid met een speciaal op deze muzikanten gerichte module, waarin zij met melodieën kunnen transcriberen, inzien en van commentaar kunnen voorzien.

Speciaal voor Speelmuziek werd een invoer- en conversiemodule gebouwd waardoor het mogelijk is met verschillende soorten muzieknotatie te werken. Voor het doorzoekbaar maken van de muziek is binnen het project WITCHRAFTplus (What is topical in Cultural Heritage: Content-based Retrieval Among Folksong Tunes) opdracht gegeven tot het bouwen van een gebruikersvriendelijke melodieënzoekmachine. Het is deze zoekmachine die in dit document beschreven wordt.

## Gebruikerswensen

* Gebruikers moeten de liederen in de databank kunnen doorzoeken, en gelijkende liederen kunnen vinden.
* Gebruikers moeten in staat zijn om zelf input te leveren om te vergelijken met liederen uit de databank
* Te gebruiken door iedereen met Internet Explorer 6+, Firefox 3+, Google Chrome of Safari.

## Technische vereisten

* Er moet een web-applicatie gebouwd worden.
* Het vergelijken van liederen moet gebruik maken van het vergelijkingsalgoritme uit het proefschrift van Peter van Kranenburg en de in C++ geprogrammeerde libmusical library die daarbij hoort.
* De vergelijken moet snel zijn: het zoeken moet een 'instant'-gevoel geven
* Het vergelijken moet schaalbaar zijn, en ook bij grote aantallen melodieën snel werken.
* Gebouwd onder REST-architectuur
* Het onderhoud van de applicatie moet uitgevoerd kunnen worden door de afdeling Technische Ontwikkeling van het Meertens Instituut
* Aansluiting op de bestaande infrastructuur van de Nederlandse Liederenbank

##  Randvoorwaarden

* De applicatie moet gemakkelijk zijn te gebruiken door anderen, waarbij concreet mogelijkheden worden gezien voor Répertoire International des Sources Musicales (RISM).
* De code moet open source gepubliceerd worden

# Informatie Infrastructuur

##  Inleiding

Om de in het vorige hoofdstuk genoemde vereisten zo efficiënt mogelijk te implementeren in een functioneel geheel, is een blauwdruk opgesteld. In dit hoofdstuk wordt een toelichting gegeven op de indeling die is gemaakt voor wat betreft de techniek, deze wordt in de volgende paragraaf toegelicht. In paragraaf 2.2 wordt een overzicht gegeven van de bronnen die in de REST-architectuur zijn onderscheiden. Hoe, met deze bronnen als leidraad, de verdeling heeft plaatsgevonden wordt beschreven in paragraaf 2.3.

In de blauwdruk van het project is een scheiding gemaakt tussen een generiek raamwerk ter beheer van collecties met melodieën enerzijds en de specifieke implementatie daarvan voor de *Speelmuziek* collectie van het Meertens Instituut. Deze opzet heeft er voor gezorgd dat op eenvoudige wijze meerdere collecties op eenzelfde server kunnen worden gehost, en dat onderhoud voor deze collecties slecht eenmaal hoeft plaats te vinden. De beschrijving in dit document gaat over het gehele raamwerk.

## Bronnen

Binnen het raamwerk zijn twee bronnen onderscheiden:

 1) algemene informatie, zoals een thuispagina of help-pagina,

 2) pagina's betreffende de beheerde collecties.

Binnen de collectie bestaan drie subbronnen, die de kern vormen van het raamwerk:

 a) informatie-pagina's

 b) overzichten en details van melodieën in de database: de tunes

c) overzichten en details van melodieën die gebruikers hebben gedeeld binnen deze collectie: de userfiles

##  Toegang tot bronnen

Zoals genoemd in het vorige hoofdstuk is er voor gekozen om het project vorm te geven middels een REST-architectuur. Voor elke van deze bronnen zijn vier acties in een REST-context, namelijk die van de vier HTTP-request GET, POST, DELETE en PUT. Elk van deze mogelijkheden wordt in de volgende paragraaf besproken. De PUT-actie wordt slecht ter volledigheid genoemd, deze is nergens geïmplementeerd.

Op basis van de in de vorige paragraaf genoemde bronnen kunnen de resources in de onderstaande tabel worden onderscheiden. Voor de het raamwerk in zijn huidige staat zijn alleen de GET requests geïmplementeerd, met uitzondering van /collections/userfiles/, die ook een POST methode kent. Per mogelijke actie is ook weergegeven wat de actie zou moeten inhouden.

|  |  |  |  |
| --- | --- | --- | --- |
| GET | POST | DELETE | PUT |
| **/** |
| De thuispagina van het raamwerk. Hierop kan een welkomst-boodschap worden gegeven over de collecties die worden beheerd. Deze pagina is niet vereist binnen het project, voor een toelichting hierop, zie het hoofdstuk betreffende het gebruik. | NG | NG | NG |
| **/info/** |
| Een overzicht van de beschikbare informatie pagina's binnen het raamwerk. Deze pagina's zijn niet vereist, zie hoofdstuk 5 voor details over het gebruik |  |  |  |
| **/collections/** |
| Een overzicht van de collecties binnen deze instantie van het raamwerk | Niet geïmplementeerd. Het is technisch niet mogelijk om dynamisch via een HTTP-request een collectie toe te voegen, vanwege de veiligheidsrisico's die hieraan verbonden zijn. |  |  |
| **/collections/<col-id>/** |
| De thuispagina van een collectie. In het geval en *Speelmuziek* is er voor gekozen om hier een simpel zoekveld weer te geven |  | Niet geimplementeerd. Het is technisch niet mogelijk om dynamisch via een HTTP-request een collectie te verwijderen, vanwege de veiligheidsrisico's die hieraan verbonden zijn |  |
| **/collections/<col-id>/userfiles/** |
| Het verkrijgen van een overzicht van alle geüploade bestanden door gebruikers. Deze manier van opvragen kent een aantal parameters, waarvan een naslagwerk is te vinden in hoofdstuk 5. | Het uploaden van een gebruikersinput, zoals het uploaden van een midi-bestand |  |  |
| **/collections/<col-id>/userfiles/<file-id>** |
| Krijg de details van een door een gebruik geupload bestand, of begin met een vergelijking. Zie hoofdstuk 5 voor details over deze compare actie. |  | Deze actie zou het verwijderen van gebruikers-input betekenen. |  |
| **/collections/<col-id>/tunes/** |
| Een opsomming van alle melodieën in deze collectie. Deze manier van opvragen kent een aantal parameters, waarvan een naslagwerk is te vinden in hoofdstuk 5. | Niet geïmplementeerd. Deze actie zou het toevoegen van een nieuwe melodie behelzen en zou zo een vervanging of aanvulling kunnen zijn op de huidige *Speelmuziek* editor. |  |  |
| **/collections/<col-id>/tunes/<nlb-id>** |
| De details van een specifieke melodie zoals deze zijn opgeslagen in de database van de opgegeven collectie. Ook dient deze bron als beginpunt voor een vergelijking. Zie hoofdstuk 5 voor informatie over het gebruik van de parameters hiervoor. |  |  |  |
| **/collections/<col-id>/info/** |
| De 'overige' pagina's van een specifieke collectie. Bijvoorbeeld een colofon of een help pagina. |  |  |  |
| **/collections/<col-id>/info/<info-id>** |
| Toon de informatie-pagina. Alleen HTML output mogelijk, aangezien deze pagina's vanuit een SMARTY-template worden gegeneerd. |  |  |  |

# Technische blauwdruk

##  Inleiding

Om te komen tot een webapplicatie die voldoet aan de in het eerste hoofdstuk geformuleerde eisen, is het project opgedeeld in vier onderdelen:

1. miLody: een raamwerk om verzoeken van gebruikers af te handelen. In dit raamwerk kunnen collecties worden geïnstalleerd, zoals de collectie *Speelmuziek* van het Meertens Instituut. Er gekozen om de backend te ontwikkelen in PHP, waarbij als uitgangspunt PHP versie 5.1.6 is genomen.
2. De alignment server. De aligment server is een onafhankelijk van het raamwerk opererend programma dat de berekeningen uitvoert om melodieën te vergelijken. De server is geprogrammeerd in C++ om optimale snelheid te garanderen.
3. De gebruikersinterface. Om de gebruiker met het raamwerk te laten communiceren is uiteraard een gebruikersinterface vereist. Er is gekozen voor het gebruik van de template-engine SMARTY. Voor de frontend is gebruik gemaakt van HTML, CSS en Javascript uitgebreid met de jQuery bibliotheek. Voor de gebruikte CSS3 en Javascript-functies is er gezorgd voor ‘Gracefull degradation’: wanneer de gebruiker niet over deze mogelijkheden beschikt, is de site alsnog goed te bereiken.
4. De collectie *Speelmuziek* , zodat het oorspronkelijk doel van de ontsluiting van de *Speelmuziek* collectie uit de liederenbank kan worden bereikt.

Van elk van deze vier onderdelen wordt in dit hoofdstuk de globale werking worden behandeld. De bijbehorende code is - in het Engels - uitgebreid gedocumenteerd op regelniveau. Voor het PHP raamwerk is er ook een PHPdoc documentatie beschikbaar waarin alle functies en parameters zijn samengevat.

##  miLody: de backend

### Overzicht

miLody is een raamwerk dat bestaat uit een set van PHP klassen. In elk PHP-bestand is een enkele klasse vertegenwoordigd, uiteraard met uitzondering van index.php. De namen van de klassen zijn logisch opgebouwd naar de positie die zij innemen in de paden van de resources.

Er zijn vier soorten PHP scripts te onderscheiden en voor elk van deze soorten is een aparte map aangemaakt:

* / de start van het raamwerk, waar index.php zich bevindt
* /frestune/ Het raamwerk om bronnen binnen de collectie te identificeren.
* /usermethods/ Mogelijkheden om gebruikersinvoer te parsen, zoals uploaden.
* /collections/ Dataproviders van toegevoegde collecties, zoals *Speelmuziek* .

Het startpunt van de webapplicatie is uitaard index.php. Hier wordt de beschikbare data verzameld en het verzoek van de gebruiker geïnterpreteerd. Alle paden die worden gebruikt binnen miLody zijn namelijk virtueel: er is geen sprake van een bestandssysteem dat overeenkomt met de gevraagde URL, zoals gebruikelijk is. In plaats daarvan wordt door middel van een mod-rewrite[[2]](#footnote-0), te vinden in het .htaccess-bestand van het raamwerk, het verzoek als GET-parameter doorverwezen naar index.php. Daar wordt de opgegeven URL vertaald in instructies.

In index.php worden de randvoorwaarden geschetst voor de applicatie: de benodigde bestanden worden handmatig benoemd; specifieke exception-klassen worden benoemd; het logboek wordt aangemaakt; de excepties worden afgehandeld. Daarnaast werkt index.php ook als bootstrap van de resources, wanneer in de clsRestTron-klasse de opgevraagde bron wordt geïdentificeerd.

### De basis: de generieke klassen

In het vorige hoofdstuk is de informatie architectuur bespoken die aan het raamwerk ten grondslag ligt. Uit deze blauwdruk is afgeleid dat er twee soorten informatiebronnen zijn: collecties en resources. Collecties beheren resources: zo bieden mogelijkheden tot toevoegen, verwijderen, bijwerken en geven bovendien een overzicht van alle resources binnen de collectie. Resources zijn de items in de collectie zelf: deze bevatten de data binnen de collectie.

Binnen het raamwerk zijn drie basisklassen gedefinieerd:

* genericCollection
* genericResource
* genericDataProvider

De klassen genericCollection en genericResource bevatten de functies die voor de afhandeling van alle collecties respectievelijk resources nodig zijn. Voor elke collectie en voor elke resource binnen het raamwerk bestaat een afgeleide van deze klassen. Beide klassen hebben de volgende opzet, met als meest interessante functies \_\_construct(), handleRequest() en display().

Het gebruik van de constructor zorgt ‘kettingreactie’ bij het afwerken van het verzoekpad van de gebruiker. De \_construct() functie ontvangt twee parameters: de parent van de aangemaakte klasse – dat wil zeggen: het object dat hoort bij het vorige niveau – en de singleton waarin, zo wordt in paragraaf 3.2.4 uitgelegd, onder andere een gesplitste versie van het pad is opgeslagen.

Bij het initialiseren van het object wordt eerst de data voor de specifieke resource of collectie geladen. Deze data is op deze manier ook voor de ‘kinderen’ van deze klasse beschikbaar.

Als er nog niveaus na het huidige niveau moeten worden herleid, wordt een instantie van de opgevraagde klassen aangemaakt. Dit gebeurt in makeChild(). Deze functie is uiteraard voor elke specifieke instantie van de generieke klassen anders. Omdat van deze klasse de constructor functie weer hetzelfde doet, ontstaat een kettingreactie waardoor geen handmatige functie-aanroepen nodig zijn.

Wanneer er geen niveaus meer zijn om te herleiden in het pad wordt er actie ondernomen door middel van handleRequest(). Deze functie stuurt het verzoek door naar de juiste procedure, afhankelijk van de gevraagd http actie: GET, POST, DELETE, PUT. De inhoud van deze functies is ook voor iedere instantie anders, maar zal er vaak uit bestaan om voor enige vorm van output te zorgen. Output wordt geregeld in de display() functie.

De laatste generieke klasse, genericDataProvider, is bedoeld als bouwsteen voor de collecties die beheerders van het raamwerk kunnen toevoegen. Zoals de naam doet vermoeden wordt in deze klasse wordt primair de communicatie met databronnen verzorgd: de database en de alignment server. Ook zijn er aantal functies opgenomen voor het formatteren van data, deze zijn makkelijk te begrijpen en bovendien goed gedocumenteerd.

Wel enige aandacht verdient de constructor. Hier wordt namelijk een aantal bijzondere GET paramaters geïnterpreteerd: de velden waarop gezocht kan worden, het veld waarop gesorteerd moet worden en de waardes voor de gapOpeningScore en gapExtensionScore die worden gebruikt door de alignment server.

### Afgeleiden van de basisklassen

De klassen zijn volgens een vast patroon genoemd:

* Klassen waarvan de namen beginnen met het prefix 'mgm' zijn uitwerkingen van genericCollection
* Klassen waarvan de namen beginnen met het prefix 'res' zijn uitwerkingen van genericResource.

Verder zijn de namen een directe aanwijzing van de naam van de resource of klasse zoals die in een URL voor kan komen. Collecties zijn altijd in meervoud genaamd, bijvoorbeeld: mgmTunes, de resources zijn altijd in enkelvoud genoemd: resTune.

Voor de meeste klassen zijn de functies redelijk simpel. Er wordt in de loadResource() procedure data uit de database geladen en die wordt via restGET() en display() weergegeven. Een aanvulling hierop zijn de doCompare() functies in zowel resTune en resUserfile, die een verzoek tot het vergelijken van liederen afhandelen.

Er zijn twee punten in de afgeleide klassen die bijzondere aandacht verdienen:

* In de loadResource() functie van resCollection wordt de opgevraagde collectie geinitialiseerd. Hier wordt de door de beheerder aangemaakte dataProvider aangemaakt en toegevoegd aan het singleton-object dat door de gehele code beschikbaar is.
* In de restPOST() functie van mgmUserFiles worden de verschillende manieren van gebruikersinput afgehandeld. Deze gebruikersinput wordt behandeld in paragraaf 3.2.5. Dit is binnen het raamwerk de enige klasse die een geïmplementeerde versie van restPOST heeft; de andere klassen kunnen geen data van gebruikers verwerken.

### Singleton (clsRestron)

De data die binnen het gehele raamwerk beschikbaar moet zijn, is opgeslagen in een instantie van clsRestron: een volgens het singleton-patroon opgezette klasse. Binnen deze klasse worden collecties, methoden van gebruikersinput en noodzakelijke variabelen zoals de databasehandler geregistreerd in respectievelijk registerSpecialAction(), registerCollection(), registerCollection().

In de functie getPathInfo() wordt het verzoek van de gebruiker geïnterpreteerd. Een van de kernaspecten is de lijst die opsomt wat geldige paden zijn. Hoewel er binnen de basisklassen uit de vorige paragraaf een foutafhandeling bestaat om ongeldige paden af te vangen, is deze functionaliteit ingebouwd als extra bescherming tegen ongeldige verzoeken. De structuur van het virtuele verzoek wordt logisch gevolgd. Per laag van bronidentificatie, wordt de goede klasse gemaakt. In de \_\_construct() functie van deze klasse wordt dit proces voortgezet. Bijvoorbeeld, de URL:

 http://foo.bar/collections/speelmuziek/tunes/

zorgt voor het volgende klassenschema:

 resRoot => mgmCollection => resCollection => mgmTunes

In executeResource() van clsRestron wordt de allesomvattende klasse mgmRoot aangeroepen, met de singleton zelf als enige parameter. Vanaf hier wordt de code afgehandeld door de basisklassen uit de volgende paragraaf.

### Gebruikersgedefinieerde collecties

De ontsluiting van door de gebruiker ingebrachte collecties gebeurt door middel van klassen die de generieke klasse genericDataProvider als basis hebben. De basis van deze klasse is reeds besproken in paragraaf 3.2.2. Het aanmaken van collecties is beschreven in de beheerdershandleiding van miLody.

Klassen om collecties te ontsluiten moeten worden geregistreerd in index.php, door middel van de registerCollection() functie van het singleton object. Daar wordt ook gespecificeerd met welke URL deze klasse kan worden aangeroepen. Voor een collectie met de naam ‘speelmuziek’ wordt de URL bijvoorbeeld:

http://www.foo.bar/collections/speelmuziek/

**Opmerking:** Omdat deze naam wordt gebruikt voor de URL, is het aan te raden geen speciale tekens of spaties te gebruiken.

De klassen om collecties te ontsluiten, staan in de map /collections.

### Muzikale gebruikersinvoer verwerken

Methode om muzikale invoer van gebruikers te verwerken worden modulair ingevoegd in het raamwerk. De verwerking bestaat er uit dat de invoer wordt omgezet naar een formaat dat de alignment server kan verwerken: het JSON-bestand of een MIDI-bestand. Deze klassen gaan zelf niet over het vergelijken van de invoer: zij maken enkel het bestand aan waarmee later de vergelijking kan worden gedaan.

**Opmerking:** Wanneer muzikale invoer is verstuurd, zorgt dit voor een bestand dat blijft bestaan, ongeachte het einde van de sessie van de gebruiker. Deels heeft dit een technische reden: een REST-archtitectuur kent geen ‘sessies’, een gedaan verzoek moet altijd hetzelfde resultaat geven. Een ander belangrijk voordeel van deze methode is dat een gebruiker een melodie niet voor elk bezoek hoeft in te voeren, maar ook dat de gebruiker een URL naar zijn invoer kan versturen naar anderen, die dan zelf de melodie kunnen vergelijken.

Er zijn vier vereiste functies aan een klasse om muzikale invoer te verwerken:

* getFileType()
* getFileOrigin()
* getFileContents()
* getFileResponseType()
* \_\_construct()

De functies getFileType(), getFileOrigin(), getFileContents() en getFileResponseType() zijn wrappers en hebben simpele taken. getFileType() geeft het formaat terug waarnaar de invoer is geconverteerd: “nlb” of “midi”. Deze waarde moet exact overeenkomen met de mogelijke waarden voor het aanroepen van het **RANK**-commando voor de alignment server. getFileOrigin() geeft aan wat het bronformaat was; dit is een unieke aanduiding van deze klasse die puur voor administratieve doeleinden wordt ingezet. getFileContents() geeft de inhoud van de conversie terug (dus in NLB of MIDI-formaat).

getFileResponseType() geeft aan hoe de locatie van de aangemaakte resource moet worden teruggegeven aan de gebruiker. Mogelijke waarden zijn ‘html’,n of ‘redirect. In het geval van ‘html’ wordt een antwoord gestuurd met http-code 200 en als berichtinhoud

 Location=<locatie van de resource>

In het geval ‘redirect’ wordt een antwoord gestuurd met http-code 201 en een Location-header naar de gemaakte resource. De waarde ‘html’ kan gebruikt worden voor componenten in de gebruikersinterface. In het specifieke geval van de collectie *Speelmuziek* is er gebruik gemaakt van een javascript-component dat niet om kan gaan met status-codes of http-headers, maar wel met berichtinhoud. Dit was de directe aanleiding voor introduceren van deze functie. Het gebruik van ‘redirect’ heeft de voorkeur.

De daadwerkelijke conversie vindt plaats in de construct() functie. Deze is voor elke invoer anders. Hoofdzaak is om een variabele te vullen met de inhoud van de converteerde gebruikersinvoer, die later via getFileContens() kan worden teruggegeven.

**Opmerking:** Let op dat de conversie niet plaatsvindt in getFileContents() maar in de \_\_construct() functie. Het is niet uitgesloten dat de getFileContents() in de toekomst meerdere malen wordt aangeroepen in de code, waardoor een conversie onnodig meer dan eens wordt uitgevoerd.

**Opmerking:** Het opslaan van de geconverteerde gebruikersinvoer gebeurt in mgmUserFile en is geen taak voor de klasse voor muzikale invoer. Zorg dus dat alle tijdelijke bestanden worden verwijderd, omdat deze later niet meer zijn te traceren.

Het raamwerk wordt geleverd met drie soorten verwerking van gebruikersinvoer: het uploaden van een MIDI-bestand, het inspelen van een melodie op een virtuele piano en het neuriën van een deuntje.

Klassen om gebruikersinvoer te verwerken moeten worden geregistreerd in index.php, door middel van de registerSpecialAction() functie van het singleton object. Daar wordt ook gespecificeerd met welke URL deze klasse kan worden aangeroepen. Wanneer bijvoorbeeld een klasse wordt geregistreerd met als naam ‘upload’, wordt de URL waarnaar het POST verzoek tot conversie moet worden verstuurd:

[http://www.foo.bar/collections/<collectienaam>/userfiles/\_upload/](http://www.foo.bar/collections/%3Ccollectienaam%3E/userfiles/_upload/)

**Opmerking:** Omdat deze naam wordt gebruikt voor de URL, is het aan te raden geen speciale tekens of spaties te gebruiken.

De klassen om muzikale gebruikersinvoer te converteren, staan in de map /useractions.

### Exceptions

Binnen het gehele raamwerk wordt gebruik gemaakt van de try/catch constructie, met bijbehorende Exceptions. Deze uitzonderingen worden steeds op het laagst mogelijke niveau gegenereerd en daarna ongewijzigd naar boven gepropageerd, totdat ze worden gevangen in index.php. Elke foutmelding heeft zijn eigen code in het formaat:

 <Klasse identificatie>:<functieafkorting>:<foutnummer>

Daarna volgt een toelichting op de gegeven fout en in sommige gevallen extra data over deze fout.

De mogelijke Exceptions zijn:

|  |  |  |
| --- | --- | --- |
| **Exceptie-klasse** | **HTTPcode** | **Uitleg** |
| excResourceNotFound | 404 | De bron kon niet gevonden worden. |
| excRequest | 401 | De gebruiker heeft een verzoek ingediend dat ongeldig was. |
| excDatabaseError | 500 | Een fout met betrekking tot het ophalen van data uit de database. |
| excServer | 500 | Een algemene serverfout, zoals een permissie-fout |
| excNotImplemented | 500 | Een niet geïmplementeerde HTTP-methode werd gevraagd, . de default-response uit genericResource en genericCollection. Deze fout wordt alleen gegeven als een gebruiker moedwillig niet-ondersteunde verzoeken stuurt. |
| excConfiguration | 500 | Een fout die gemaakt is tijdens het instellen van - waarschijnlijk - de collectie. Dit kan zijn een niet-kloppende naam, een verwijzing naar een niet-bestaande klasse of een pad dat niet bestaat. Dit zijn |

*Tabel 3.2.7 Specifieke excepties*

Voor de fouten met HTTP-code 404 en 500 zijn aparte errordocuments geleverd, die in de stijl van gebruikersomgeving de fout weergeven.

Van elke fout wordt ook melding van in het logboek. Het logboek is een zeer simpele klasse in de vorm van clsLogger, die enkel de fout toevoegt aan een eerder opgegeven logboek. Ook kan in het geval van de fouten met http-code 500 een mail verstuurd naar de beheerder van de website.

##  Alignment Server

### Inleiding

De alignment server voert de berekeningen uit die nodig zijn om melodieën met elkaar te vergelijken. De server is in C++ geprogrammeerd. De server en bestaat uit een Main() en een klasse clsServer, die kan worden gezien als een wrapper om libmusical, het vergelijkingsalgoritme van Peter van Kranenburg.

Per instantie van de alignment server kan slechts een enkele collectie worden ontsloten, maar het is mogelijk om meerdere alignment servers binnen een server omgeving te laten opereren. Elke server heeft een unieke toegangspoort nodig die gekoppeld is aan een map met JSON bestanden die de collectie vormen.

### Communicatie framework en alignment server

De communicatie tussen het PHP raamwerk en de alignment server verloopt via named pipes, ook wel FIFO’s genoemd. Een pipe is een type bestand waarnaar kan worden geschreven via de standaard functies om naar bestanden te schrijven in zowel PHP als C++. In plaats van dat de data wordt weggeschreven, wordt de data doorgegeven aan processen met een READ handler van de pipe.

Per collectie van de server wordt een pipe aangemaakt, waar de server luistert naar berichten van clients. Omdat deze pipe dient als publieke toegangspoort voor de andere processen, wordt in deze documentatie en in de code van de server ook gesproken over de public pipe.

Een client maakt per gestuurd verzoek zelf een named pipe aan, de private pipe. Hier zal de server de output naar toe sturen. De client is verantwoordelijk voor het verwijderen van deze pipe.

### Daemon – Spawn constructie: fork()

Elke keer dat een client verbindt met de server, wordt een kloon van de server gemaakt: een fork. Hiermee wordt de server, die luistert naar binnenkomende berichten, de daemon en het gekloonde proces dat het verzoek afhandelt de worker.

Het voordeel van deze constructie is in de eerste plaats de snelheid: clients hoeven niet te wachten op eerdere verzoeken van andere clients. Een tweede voordeel is dat in het geval dat het afhandelen van het proces zou leiden tot een fout, door bijvoorbeeld verkeerde invoer, een crash geen gevolgen heeft voor volgende verzoeken: de daemon blijft opereren.

### Response

De commando’s **START** en **STOP** kennen geen response. De response voor het commando **RANK** bestaat uit een JSON-array met als key-element de naam van het bestand uit collectie en als waarde de genormaliseerde vergelijking met het gevraagde bestand.

**Opmerking:** Er wordt logdata geschreven naar de log-stream (stream 2).

### Kernfuncties

### Main

Het starten van de server wordt afgehandeld in Main(). Daar wordt een oppervlakkig controle of het aantal vereiste argumenten is meegegeven, waarna een clsServer-object wordt aangemaakt met de opgegeven parameters.

In het geval een **START** opdracht wordt via de functie createIndex() een index gemaakt alle JSON bestanden die zich in het opgegeven pad bevinden. Deze bestanden worden ingelezen en omgezet in een Sequence-object uit libmusical bibliotheek. Wanneer de bestanden zijn ingelezen wordt er geluisterd naar berichten in de pipe van de server in de functie Listen().

Bij het ontvangen van een **STOP** opdracht wordt de STOP() functie van de server aangeroepen, waarbij een stop commando wordt verstuurd naar alle lopende processen van de opgegeven collectie en elk worker zichzelf stopt en een stopsignaal stuurt naar de daemon.

### createIndex()

Deze functie laadt alle melodieën in het geheugen zodat deze doorzocht kunnen worden. De bestanden dienen in het JSON formaat van Peter van Kranenburg te worden aangeleverd. Voor elk bestand wordt de betandsnaam opgeslagen in de array indexSequencesNames en wordt de verwerkte melodie als Sequence opgeslagen in de array indexSequences. Beide variabelen zijn klassevariabelen.

**Opmerking:** Er is geen destroyIndex() functie.

### Listen()

In deze functie wordt de pipe aangemaakt en wordt het proces van data ontvangen in gang gezet. Het lezen van de buffer gebeurt met een standaard while-loop , die wacht tot een EOF wordt ontvangen. Een van de speciale kenmerken van een pipe is dat deze geen EOF ontvangen zolang er tenminste een WRITE-operatie aan de pipe verbonden is. Daarom wordt tegelijkertijd met de public pipe een tweede pipe aangemaakt: een dummy. Deze blijft bestaan zolang er geen **STOP** commando is gegeven en zorgt daarmee voor een altijd luisterend oor.

### ParseMessage()

Het centrale regelpunt van de klasse clsServer is parseMessage(). In deze functie worden alle binnengekomen berichten van clients afgehandeld.

Zoals eerder aangegeven wordt hier de splitsing tussen daemon en spawn gemaakt. In het geval van een RANK commando wordt een bericht gedecodeerd – base64 codering wordt ondersteund – en wordt ofwel doMatchMidi() of doMatchNLB() aangeroepen.

### doMatchMidi() en doMatchNLB()

De functies doMatchMidi() en doMatchNLB() zorgen voor het matchen van respectievelijk een gestuurd MIDI-bestand of NLB-bestand met de NLB-bestanden die behoren tot de collectie.

De voor libmusical benodigde objecten worden aangemaakt en in een simpele loop-constructie wordt het gemaakte Sequence-object vergeleken met alle Sequence objecten uit de array indexSequences die in de functie createIndex() is aangemaakt.

##  Gebruikersinterface

### Uitvoermogelijkheden

De gebruikersinterface wordt aangestuurd via vier punten: De foutafhandeling in index.php, of de display() functies in de resource-klassen, de display() functie in collectie-klassen of - in een enkel geval – displayHTTPHeaders(), als er een resource gemaakt is. Dit is alleen het geval in de eerder genoemden POST handler in mgmUserFiles.

Het front-end kent twee soorten output: JSON en HTML. De JSON output wordt door middel van een json\_encode in objectnotatie gestuurd. De HTML-code wordt gegenereerd door middel van SMARTY en de daar bijhorende templates. De locatie van de templates kan worden ingesteld in de dataProvider van de collectie. Het template voor de root van het raamwerk is *root.tpl.*

### Uitvoernotatie

De SMARTY templates krijgen dezelfde data toegestuurd als er in de JSON-output te zien is, zij het als variabele. De volgende variabelen worden altijd meegegeven:

|  |  |
| --- | --- |
| **Variable** | **Inhoud** |
| templateURL | De locatie van de templates als URL, die wordt gebruikt voor het laden van CSS-bestanden en plaatjes. |
| collectionURL | De locatie van de collectie, als URL, die kan worden gebruikt bij het leggen van links |
| records | De data, als array. |
| recordsTotal | Het totale aantal records dat er gevonden is, **zonder** in achtneming van paginering.  |
| recordsPerPage | Het aantal pagina’s dat maximaal kan worden weergegeven. Als dit getal kleiner is dan recordsTotal, betekent dit dus dat er meerdere pagina’s zijn. |

*Tabel 3.4: De response die uit een JSON-verzoek komt of meegeven aan het SMARTY-template*

 **Opmerking:** de variabele recordsTotal is afkomstig van SQL\_CALC\_FOUND\_ROWS.

### Gereserveerde GET parameters

Er is een viertal GET paramaters met een speciale betekenis, namelijk:

1. sort
2. page
3. gapOpeningScore
4. gapExtensionScore

De eerste parameter sort geeft aan op welk veld gesorteerd moet worden bij een zoekactie. De geldige waarden voor dit veld worden in de dataProvider van elke collectie apart benoemd. De standaard sortering hangt af van de SQL-query die wordt gebruikt in getTunes() van de dataProvider.

De tweede parameter page geeft de pagina aan waarop moet worden begonnen met weergeven. Dit hangt samen met het aantal opgegeven records per pagina in de dataprovider. De eerste pagina heeft als index 0. De standaardwaarde van page is 0.

De derde en vierde parameters gapOpeningScore en gapExtensionScore worden gebruikt bij het vergelijken van melodieen onderling of muzikale gebruikersinvoer en de melodieen in de collectie. Deze waarden moeten een numeriek getal zijn tussen 0 en 100. De standaardwaarden voor gapOpeningScore en gapExtensionScore zijn vastgelegd in de array $daemonAdvancedDefault in de dataProvider.

Naast deze vier paramaters worden ook de mogelijke zoekvelden uit de dataProvider speciaal behandeld.

Naast deze GET paramaters staat het de beheerder vrij om andere GET parameters te gebruiken bij het maken van templates.

# Collectie Speelmuziek

##  Inleiding

Om het primaire doel van het miLody-project – het ontsluiten van de collectie *Speelmuziek* - te realiseren is voor deze collectie een module gemaakt voor binnen het raamwerk. In dit hoofdstuk zal een toelichting worden gegeven op de module geschreven code die horen bij de gebruikersinterface. De code bij deze module is in de distributie bijgevoegd als clsDataProviderSpeelMuziek.example.php.

##  Dataprovider

### Instellingen

De meeste instellingen zijn simpel van aard en bestaan uit locatiepaden en de databaseconfiguratie. Er zijn enkele instellingen die specifiek op deze collectie gericht zijn:

|  |  |
| --- | --- |
| **Variabele** | **Uitleg / waarden** |
| dataConstants |  |
| infoPages | Er zijn drie speciale pagina’s binnen de collectie *Speelmuziek*: help, about en colofon. Deze zijn te vinden in de templates help.tpl, about.tpl en colofon.tpl, respectievelijk. |
| searchParameters | De parameters waarop gezocht kan worden; deze komen overeen met de namen van de input-elementen in de templates, met name in het bestand mgmTunes. Deze elementen kunnen worden benaderd in getTunes(), waar deze velden worden omgezet in databasevelden. De volgende velden zijn opgenomen: writer, incipit, location, title, tune, sourcetitle, sourcecentury, freesearch voor auteur, beginregel, opnamelocatie, titel, wijsaanduiding, brontitel, eeuw van vervaardiging van de bron en het ‘zoek alles’-veld.  |
| sortParameters | De velden waarop gezocht kan worden; deze waarden komen overeen met de options uit het select-element sort in mgmTunes.tpl. sortTitle en sortCentury worden ondersteund voor respectievelijk de titel van het lied en de eeuw waarin de bron is vervaardigd. |
| daemonAdvancedDefault | De standaardwaarden voor een vergelijking tussen liederen, die overeenkomen met de gapOpeningScore en gapExtensionScore voor . Deze waarden kunnen worden aangepast door de GET-parameters gapOpeningScore en gapExtensionScore . |

### getTuneById()

Deze functie geeft simpelweg alle noodzakelijk data terug voor een enkel id. Welke velden noodzakelijk zijn in de uitvoer, hangt af van de velden die worden opgevraagd in het template resTune.tpl.

### getTunes()

Het doel van de functie getTunes() is om de noodzakelijke informatie op te halen voor het tonen van melodieën uit de databank die aan de opgegeven kenmerken voldoen. Het bijhorende template is mgmTunes.tpl, de velden die daarin worden aangesproken moeten beschikbaar gemaakt worden middels de functie getTunes().

Het verkrijgen van de opgegeven kenmerken vindt plaats door middel van for…each loop over searchParameters. Deze array is opgegeven in de basis van de dataProvider en inmiddels gevuld met de door de gebruiker opgegeven kenmerken, voor zover van toepassing. Voor elk van deze velden moet – mits de waarde is opgegeven – een WHERE-restrictie worden toegevoegd. Een bijzondere variant is de freesearch parameter, omdat deze door meerdere velden in de database zoekt. In het voorbeeld van de dataProvider van *Speelmuziek* wordt gebruik gemaakt van de functie createWordGroupsSql, die eerder in dit document is toegelicht.

Het sorteren gebeurt op gelijksoortige wijze. Merk op dat het sorteren op de Boolean waarde die volgt uit de vergelijking

(VELD = ‘’)

er voor zorgt dat velden waarvoor het kenmerk niet is opgegeven, achteraan in de lijst komen te staan in plaats van vooraan.

##  Gebruikersinterface

### Overzicht

Het raamwerk ondersteund voor elke resource en voor elke collection een template. Voor elke collection is er een template dat begint met list- , voor elke resource een template dat begint met view-. Zo wordt het overzicht van de melodieën uit de databank van mgmTunes doorgegeven aan het template list-tunes.tpl. Een uitzondering hierop is de gebruikersinterface voor het invoeren van muzikale invoer. Deze zouden logischerwijs in list-userfiles.tpl moeten staan, maar vanwege clustering van belangrijke onderdelen is er voor gekozen om deze functionaliteit onder te brengen in list-tunes.tpl.

In de collectie voor *Speelmuziek* zijn niet alle templates uitgewerkt. Zo is het niet mogelijk om een overzicht te krijgen van alle bestanden die worden gemaakt bij muzikale gebruikersinvoer omdat er geen uitgewerkte versie is van list-userfiles.tpl.

Elk template maakt gebruik van index.tpl, index.css, index.js, header.tpl en footer.tpl.

### Meertaligheid

De gebruikersinterface voorziet in twee talen: Nederlands (standaard) en Engels. De waarden voor beide talen zijn opgeslagen in language.tpl, dat aan ieder document wordt toegevoegd. In ieder template is altijd de variabele curLang beschikbaar. Deze variabele heeft de waarde voor de huidige taal omgezet in een GET parameter. Om de huidige taal te behouden moet deze waarde dus worden meegeven in elke link binnen het miLody raamwerk.

Nieuwe talen zijn gemakkelijk toe te voegen door het aanpassen van language.tpl.

### Interface voor muzikale gebruikersinvoer: virtuele piano

De interface voor het invoeren van een melodie op piano is vastgelegd in list-tunes.tpl in het DIV element met id sttunecon-play. Het toetsenbord is een aanpassing van de piano uit musipedia en gebruikt SoundManager. Voor het gebruik van deze invoer zijn aanpassingen vereist in het .htaccess-bestand, zie hiervoor de beheerdershandleiding.

### Interface voor muzikale gebruikersinvoer: neuriën

De invoer van het neuriën is de standaard interface voor het jQuery component jRecorder[[3]](#footnote-1). Deze op javascript gebaseerde module neemt de invoer van de microfoon of via een verborgen Flash object en stuurt dit naar de backend. Voor documentatie over de mogelijkheden van jRecorder, zie de website.

**Opmerking:** Voor de module is geen gebruik gemaakt van het bijgeleverde Flash-object, maar van een beta-release. Let er bij het updaten van jRecorder op dat de nieuwe versie van jRecorder in staat een statusboodschap terug te geven voor het succesvol uploaden van een bestand.

**Opmerking:** De maximale opnametijd is ingesteld op 30 seconden, omdat het de bedoeling is dat gebruikers korte, herkenbare delen van een melodie neuriën. Bij opnametijden langer dan 2 minuten duurt het uploaden van het gemaakte bestand te lang om verwerkt te worden en zal er bij het insturen een foutmelding optreden.

##  Muzikale gebruikersinvoer

### Inleiding

Voor het ontsluiten van de collectie *Speelmuziek* zijn drie vormen van muzikale gebruikersinvoer beschikbaar gemaakt:

* Het zoeken van gelijkende melodieën op basis van een geupload MIDI bestand
* Het zoeken van gelijkende melodieën op basis van een op een virtuele piano ingespeelde melodie
* Het zoeken van gelijkende melodieën op basis van het neuriën van een melodie

### Uploaden MIDI-bestand

Het uploaden van een MIDI-bestand is simpel. PHP handelt het uploaden van het bestand af, de klasse clsUserMethodUpload vangt fouten af.

### Virtuele piano

Het voorbereiden van een geneuriede melodie voor gebruik met de aligment server kent drie stappen:

1. Het openen van een WCE-template
2. Het maken van een WCE-bestand, gebaseerd op het WCE-template en gevuld met de door de gebruiker ingevoerde melodie
3. Het converteren van het WCE-bestand naar een KRN-bestand
4. Het converteren van een KRN-bestand naar een JSON-bestand

De invoer uit de virtuele piano wordt afgehandeld in de klasse clsUserMethodPlay afgehandeld. De gebruikersinterface geeft een parameter mee, namelijk de ingespeelde melodie in WCE-notatie. Deze wordt door middel van de bijgeleverde progamma’s wce2krn en krn2json omgezet in een bestand in het NLB-formaat. Dit bestand wordt gebruikt voor het vergelijken.

Om het gebruikte WCE-bestand te genereren wordt een template gebruikt dat gelezen kan worden door wce2krn. Dit template kan worden aangepast. Er wordt aan dit template een aantal parameters meegegeven, die kunnen worden aangepast in de klasse clsUserMethodPlay.

De paden voor wce2krn, krn2json en het template worden gesteld in de klasse.

### Neuriën

Het voorberieden van een geneuriede melodie voor gebruik met de aligment server kent drie stappen:

1. Het opnemen van de stem van de gebruiker in WAV formaat
2. Het versturen van het WAV-bestand naar de webserver
3. Het omzetten van het WAV bestand naar een MIDI bestand via WAON[[4]](#footnote-2), een ‘wave-to-midi transcriber’

De gebruikersinterface voor de muzikale invoer voor het neuriën zorgt voor de ruwe input van een WAV-bestand naar de std://in stream , die wordt ingelezen in clsUserMethodHum. Daar wordt deze stream rechtstreeks naar een bestand geschreven via stream\_copy\_to\_stream en via waon omgezet naar een MIDI-bestand. Het pad naar waon wordt opgegeven in de dataProvider.

1. http://www.liederenbank.nl/ [↑](#footnote-ref--1)
2. http://httpd.apache.org/docs/current/mod/mod\_rewrite.html [↑](#footnote-ref-0)
3. http://www.sajithmr.me/jrecorder-jquery [↑](#footnote-ref-1)
4. http://waon.sourceforge.net/ [↑](#footnote-ref-2)