

UNIVERSITEIT TWENTE

ONTWERPPROJECT

---

# Canvas.hs

---

*Auteurs:*

Joost VAN DOORN

Lennart BUIT

Pim JAGER

Thijs SCHEEPERS

Martijn ROO

*Begeleider:*

Robert DE GROOTE

6 december 2013

# Inhoudsopgave

<b>1</b>	<b>Inleiding</b>	<b>2</b>
<b>2</b>	<b>Ontwerp</b>	<b>3</b>
2.1	Requirements . . . . .	3
2.2	Globaal ontwerp . . . . .	4
2.3	Detail ontwerp . . . . .	4
2.3.1	Architectuur . . . . .	4
2.3.2	Externe libraries . . . . .	6
2.3.3	Grafische bibliotheek . . . . .	6
2.4	Testplan . . . . .	6
2.5	Testresultaten . . . . .	6
<b>3</b>	<b>Conclusie</b>	<b>7</b>
3.1	Sectie . . . . .	7
3.1.1	Subsectie . . . . .	7
<b>4</b>	<b>Evaluatie</b>	<b>8</b>
4.1	Sectie . . . . .	8
4.1.1	Subsectie . . . . .	8

# Hoofdstuk 1

## Inleiding

Bij het informatica-keuzevak Functioneel Programmeren gebruiken studenten Haskell om fundamentele concepten van functioneel programmeren te bestuderen. Hierbij wordt door de studenten veel gebruik gemaakt van grafische weergaven om de werking van hun code inzichtelijk te maken. Het gebruik van Haskell voor het maken van grafische weergaven blijkt vaak redelijk gecompliceerd en limiteert studenten doordat zij zich bezig moeten houden met minder intuïtieve en minder essentiële aspecten van Haskell.

Om de focus binnen Functioneel Programmeren op de essentie te houden, is een simpele manier nodig om de Haskell code van de student om te zetten naar een grafische representatie. De interface tussen de code van de student en de grafische interface moet daarom eenvoudig en bruikbaar zijn zonder dat de ambitieuze student hierbij onnodig beperkt wordt.

In dit ontwerpproject is *Canvas.hs* ontwikkeld; een omgeving die Haskell-gebruikers in staat stelt op eenvoudige wijze grafische elementen op een HTML5 canvas te presenteren. Canvas.hs is ontwikkeld met het oog op gebruiksgemak en eenvoud zonder de mogelijkheid tot uitbreiding en het toevoegen van geavanceerde functionaliteit onnodig te beperken.

### Verslagstructuur

In dit verslag wordt in Hoofdstuk 2 het ontwerp van Canvas.hs beschreven, Hoofdstuk 3 beschrijft de conclusies van dit project en in Hoofdstuk 4 wordt dit project met zijn uitkomsten geëvalueerd.

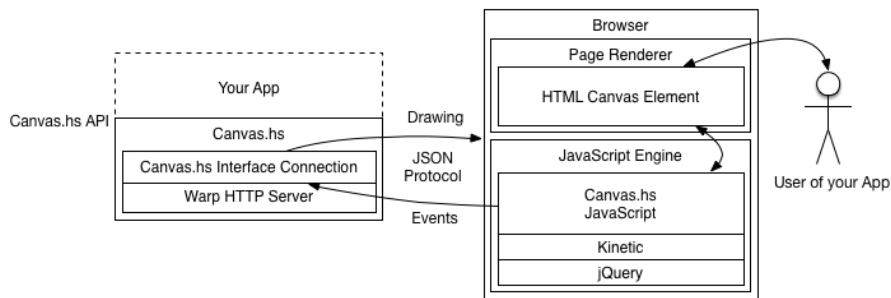
Hoofdstuk 2 Ontwerp is onderverdeeld in de secties: 2.1 Requirements, 2.2 Globaal ontwerp, 2.3 Detail ontwerp, 2.4 Testplan en 2.5 Testresultaten.

## Hoofdstuk 2

# Ontwerp

### 2.1 Requirements

- R1 Het systeem dient events vanuit javascript door te geven aan het Haskell-programma van de student.
- R2 In de javascript-omgeving zou een simpele debug-console moeten zijn.
- R3 Toetsaanslagen vanuit het canvas zouden doorgegeven moeten worden aan de haskell-code van de student.
- R4 Het systeem kan eventueel tekstinput vragen met een pop-up.
- R5 Grafische primitieven zoals cirkels, vierkanten, lijnen, Bézier curves, n-hoeken en tekst moeten op een simpele manier getekend kunnen worden vanuit een Haskell-programma.
- R6 Het moet mogelijk zijn de vul- en lijnkleur van grafische componenten in te stellen.
- R7 Een gebruiker moet grafische componenten aan kunnen passen zonder zijn programma te hoeven hercompileren.
- R8 Het systeem kan eventueel plaatjes inladen op het canvas.
- R9 Het systeem kan eventueel gradients als vulkleur gebruiken.
- R10 Er kan eventueel gezoomd en geschoven worden op de canvas.
- R11 Het systeem kan eventueel stapsgewijze aanpassingen geanimeerd weergeven.
- R12 Het systeem dient duidelijke errors te genereren.
- R13 Het systeem dient gemakkelijk en snel te gebruiken te zijn.
- R14 De FPPrac-module zou automatisch een browser moeten kunnen starten.
- R15 Het systeem zou de canvas opnieuw moeten laden als de Haskellcode opnieuw gecompileerd wordt.



Figuur 2.1: Overzicht van de architectuur van Canvas.hs

R16 Als de verbinding tussen javascript en Haskell verbroken wordt zou hiervan van melding gemaakt moeten kunnen worden aan de gebruiker.

R17 70% code coverage (exclusief frameworks) zou behaald moeten worden.

## 2.2 Globaal ontwerp

## 2.3 Detail ontwerp

### 2.3.1 Architectuur

Canvas.hs heeft een redelijk gecompliceerde architectuur. Dit komt vooral door de verschillende technologieën die nodig zijn om het HTML5 Canvas te verbinden met de te ontwikkelen Haskell API voor het bouwen van interfaces. In Figuur 2.1 is een schematische weergave van de architectuur weergegeven.

#### Haskell

Canvas.hs is een library die de programmeur kan importeren in zijn programma om er daarna met de API die Canvas.hs aanbiedt gemakkelijk een uitgebreide user interface mee te bouwen. Canvas.hs zal zich focussen op elementaire input en geen ondersteuning hebben voor high level interface elementen zoals buttons en textarea's. Deze elementen zouden met behulp van Canvas.hs wel eenvoudig te implementeren moeten zijn.

**API** De programmeur die gebruik maakt van Canvas.hs zal op een eenvoudige wijze gebruik kunnen maken van onze library. Daarom is het belangrijk dat er een API aangeboden wordt die eenvoudig te begrijpen is en niet teveel features bevat maar daarnaast wel de flexibiliteit biedt om een zeer complexe interface mee te bouwen.

**Verbinding met de interface** De verbinding met het canvas wordt bewerkstelligd met een eenvoudige HTTP server. Deze server biedt de gebruiker de mogelijkheid om via een webbrowser het Haskell programma te benaderen. De

HTTP server biedt pagina's aan waarin JavaScript en HTML samenwerken om op het canvas te tekenen.

Als via de API van Canvas.hs begonnen wordt met tekenen zal de HTTP server automatisch gestart worden. Het besturingssysteem wordt aangeroepen voor het openen van de standaard browser –verwijzende naar het adres van de lokaal draaiende HTTP server.

## Protocol

De gegevens die verstuurd worden volgens het protocol zullen op een primaire gegevensstructuur moeten aanhouden. Hiervoor kunnen twee voor de hand liggende conventies gekozen worden: XML en JSON. Ons protocol zal gegevens coderen in JSON (JavaScript Object Notation). De voordelen van JSON zijn onder andere dat de data met weinig moeite direct gebruikt kan worden in JavaScript, lichtgewicht is en makkelijk te lezen is.

**Websockets** Gegevensoverdracht tussen de HTTP server en de browser moet snel gebeuren zonder een al te groote vertraging. Er zijn in de communicatie tussen een HTTP server en client verschillende mogelijkheden waarbij websockets de meestvoordehand liggende is. Websockets is een relatief nieuwe feature van HTTP en wordt op dit moment alleen ondersteund in de nieuwe browsers. Het gebruik van websockets zou als gevolg hebben dat Canvas.hs geen ondersteuning heeft voor oudere browsers.

**Restful vs RPC** Naast de manier waarop gegevens worden gecodeerd is het ook belangrijk via welke wegen deze aan de server worden aangeboden—alsmede hoe deze door de server worden verstrekt.

RESTful HTTP webservices hebben als voordeel dat deze de basis en de functionaliteit van het HTTP protocol optimaal benutten voor het verder definiëren van een eigen protocol. RPC geeft de vrijheid om een volledig, zelf in te vullen, protocol te bouwen op het HTTP protocol. Gezien de werkbaarheid van WebSockets nog onderzocht moeten worden is het de vraag welke aanpak er uiteindelijk gebruikt zal gaan worden.

Voor AJAX requests middels HTTP fetching zou een RESTful webservice een goede optie zijn. Echter is bij WebSockets het HTTP protocol niet meer beschikbaar en zal er gebruik gemaakt moeten worden van een vorm van RPC.

**Interface data** De gegevens die door de HTTP server naar de browser gestuurd worden zullen voornamelijk bestaan uit interface data. Hoe, is de interface opgebouwd, elke elementen staan waar, welke attributen hebben deze elementen en zijn deze elementen in staat input van de gebruiker te accepteren. Hierbij zal er in het protocol rekening gehouden worden met de weergave van de elementen. De elementen die Kinetic.js ondersteund bieden hier een goede basis voor.

**Events** Events zijn de gegevens die de browser terug stuurt naar de HTTP server. Deze gegevens zullen vooral informatie bevatten over interface acties die de gebruiker uitvoerd. Het gaat hierbij om bijvoorbeeld muisbewegingen, muisklikken en toetsaanslagen. Het is belangrijk dat deze events snel door de

server worden ontvangen en verwerkt naar nieuwe output zodat de gebruiker geen zichtbare vertraging ziet in de werking van het programma.

## **Javascript**

Voor het tekenen van de interface op het HTML Canvas alsmede de communicatie vanuit de browser met de server, zal er gebruik worden gemaakt van JavaScript. jQuery biedt een prima uitbreiding op JavaScript waarbij de mogelijkheid wordt geboden verschillende zogenaamde jQuery plugins te gebruiken. In het geval van Canvas.hs zijn twee plugins belangrijk: Kinetic.js en jQuery-websockets.

Kinetic.js zal gebruikt worden voor het tekenen van de verschillende elementen op het canvas. Daarbij heeft Kinetic.js in combinatie met jQuery ook uitstekende ondersteuning voor het doorgeven van input events, die door bijvoorbeeld muisbewegingen aangeroepen worden.

jQuery-websockets is een plugin die het gemakkelijk maakt gegevensoverdracht tussen websockets af te vangen en daar snel actie op te ondernemen. Deze lichtgewicht library maakt het mogelijk snel, zonder veel overhead opdrachten door te sturen naar Kinetic.js zodat deze op het canvas getekend worden.

**Debug Console** Voor de programmeur die gebruik gaat maken van de Canvas.hs library is het belangrijk dat zijn interface er zo uit ziet zoals hij dit wil. Ongetwijfeld zal een programmeur tegen problemen aanlopen bij het bouwen van de interface die hij niet had voorzien bij het schrijven van zijn code. Daarom vinden wij het belangrijk dat de interface van Canvas.hs een zogenaamde debug console bevat waar het aanroepen van teken functies en de invloed van deze API aanroepen goed visueel en tekstueel inzichtelijk worden.

### **2.3.2 Externe libraries**

### **2.3.3 Grafische bibliotheek**

## **2.4 Testplan**

## **2.5 Testresultaten**

## Hoofdstuk 3

# Conclusie

### 3.1 Sectie

#### 3.1.1 Subsectie

Tekst in subsectie.



## Hoofdstuk 4

# Evaluatie

### 4.1 Sectie

#### 4.1.1 Subsectie

Tekst in subsectie.

# Bibliografie