

SPECCHIO Web Interface

Projektarbeit IP-5

Remo Rossi, Christian Schibli  
Windisch, 20.01.2017

Auftraggeber: Andreas Hueni, Universität Zürich

Betreuer: Martin Gwerder

Verlauf

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Name** | **Kommentar** |
| 0.1 | 06.01.17 | Christian | Initial Draft |
| 0.2 | 08.01.17 | Christian | Grobe Dokumentenstruktur |
| 0.3 | 11.01.17 | Remo | Anpassungen an der Struktur |
| 0.4 | 14.01.17 | Christian | Einleitung und diverse Notizen |
| 0.5 | 15.01.17 | Remo | Dokumentation, Architektur, Umsetzung & Entscheide |
| 1.0 | 20.01.17 | Christian / Remo | Endfassung |

Inhaltsverzeichnis

1 Einleitung 5

1.1 Ausgangslage 5

1.2 Ziel der Arbeit 5

2 Hauptteil 6

2.1 Vorgaben 6

2.2 Use Cases 6

2.3 Requirements 7

2.4 Nicht implementierte Requirements 9

2.5 Change Requests 10

2.6 Vorgehensmodell 10

2.6.1 Grobplanung 11

2.6.2 Detailplanung 12

2.6.3 Release-Planung / Meilensteine 12

2.6.4 Arbeitsmethodik 13

2.6.5 Usability Testing 13

2.7 Dokumentationen 14

2.8 Umsetzungsvarianten und Entscheidungen 14

2.8.1 Web Framework 15

2.8.2 Web Server und Version 15

2.8.3 CSS 15

2.8.4 Graph 16

2.8.5 Map 16

2.9 Architektur 17

2.10 Verwendete Technologien 18

2.11 Deployment 18

2.12 Aufgetretene Probleme 19

2.12.1 Performance 19

2.12.2 Einlesen von db\_config.txt 19

2.12.3 PDF Export 20

2.12.4 Git Repository 20

2.13 Testing 21

2.14 Bugfixing 21

3 Schluss 22

3.1 Rückblick 22

3.2 Lessons Learned 23

3.3 Ausblick 23

3.4 Schlusswort 24

4 Anhang 25

4.1 Artefakt-Referenzen 25

4.2 Glossar 25

4.3 Ehrlichkeitserklärung 26

# Einleitung

Dieses Dokument dient dazu eine Übersicht über die Projektarbeit “Specchio Web Interface” zu erhalten, welche von den Studierenden Christian Schibli und Remo Rossi im Rahmen des Moduls IP5 durchgeführt wurde. Es enthält die Informationen der wichtigsten Dokumente, welche während dem Projekt erstellt und auf Git abgelegt wurden, sowie Verweise auf jene Dokumente für genauere Informationen.

## Ausgangslage

Das geografische Institut der Universität Zürich unterhält eine Datenbank für Spektral-Daten und deren Metadaten, die über eine Website[[1]](#footnote-1) publiziert werden. In dieser Datenbank befinden sich Spektraldaten von zahlreichen Pflanzen und Materialien, zusammen mit Metadaten wie Ort, Aufnahme-Art (z.B. Handgerät, Flugzeug oder Satellit) oder Sensortyp. Auf die Daten dieser Applikation kann zur Zeit nur mit einem spezialisierten Programm (auch als Appliance verfügbar) zugegriffen werden.[[2]](#footnote-2)

## Ziel der Arbeit

Grundsätzlich bestand das Ziel der Arbeit darin, einen Zugang auf die SPECCHIO-Datenbank zu schaffen, welcher ohne zusätzliche Installation spezialisierter Software auskommt, sondern lediglich via Webbrowser zugänglich ist. Das SPECCHIO Web Interface sollte ausserdem über eine benutzerfreundliche Suche verfügen.[[3]](#footnote-3)

Für den Datenbankzugang konnten wir die bestehende SPECCHIO API verwenden und konnten uns so mehr auf die Aufarbeitung und Darstellung der Daten konzentrieren.

# Hauptteil

## Vorgaben

Detaillierte Vorgaben zum Endergebnis der Arbeit bestanden zu Projektbeginn nicht. Aus diesem Grund wurden die Use Cases, Anforderungen und die Vision in einem Workshop[[4]](#footnote-4) mit dem Auftraggeber erarbeitet.

## Use Cases

Die Applikation kann grundsätzlich in vier Use Cases eingeteilt werden:

* Nach Spektraldaten suchen
* Detailansicht anzeigen von Spektraldaten
* Spektraldaten exportieren
* Detailansicht exportieren

ML_specchio_webinterface_20161010.png

Eine genauere Beschreibung der einzelnen Use Cases ist im Dokument „Specification“[[5]](#footnote-5) zu finden.

## Requirements

Die Anforderungen wurden in einem Workshop mit dem Auftraggeber ausgearbeitet und in einer Liste mit allen Anforderungen erfasst. Während der Projektdauer wurde agil gearbeitet. Dadurch haben sich fortlaufend Änderungen an den Anforderungen ergeben. Die Anforderungen wurden in 3 Kategorien eingeteilt:

* Basisanforderungen (Prio 1)
* Basisanforderungen (Prio 2)
* Nice to Have (Optional)

Alle Basisanforderungen wurden bis Release 1.0 umgesetzt, ebenfalls gewisse optionale Anforderungen. Es ergab sich auch noch zusätzliche Anforderungen, die nach der Genehmigung des Pflichtenhefts vom 10.10.2016 gewünscht wurde.

Nachfolgend ist eine Tabelle mit der Übersicht der Requirements abgebildet. Zusätzliche Anforderungen sind gelb eingefärbt. Optionale Anforderungen, welche umgesetzt wurden, sind grün eingefärbt. Alle Details zu den Anforderungen sind im Artefakt “Requirements”[[6]](#footnote-6) zu finden.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirement** | **Prio (1–3)** | **Status** |
| 1 | The user must be able to search by keyword. | 1 | done |
| 2 | The user must be able to search by clicking the given pictograms:  – Athmosphere  – Hydrosphere  – Geosphere  – Cryosphere  – Biosphere | 3 | - |
| 3,1 | The user must be able to search by the SPECCHIO attributes (e.g. over dropdown menu). | 2 | done |
| 3,2 | Initially the total amount of spectra in database has to be displayed on the search view. | 2 | done |
| 3,3 | Searching for spectral data will update the total amount of found spectra displayed on the search view. | 2 | done |
| 3,4 | The user must be able to display the found spectral data for his search criteria. | 1 | done |
| 4 | The user should be able to search by map. | 3 | - |
| 5 | Input for map search are coordinates. | 3 | - |
| 6 | Input for map search is location name. | 3 | - |
| 7 | The system will notify about no results if the area in km2 exceeds a specific value. | 3 | - |
| 8 | The system is able to list the found spectra in a table. | 1 | done |
| 9 | Results are initially sorted by date. | 1 | done |
| 10 | The table of results should contain following information:  – date – campaign – owner – name – filename – institute | 1 | done |
| 10,1 | The table of result show the first 10 results and provides paging for the rest. | 1 | done |
| 11 | The system should be able to show a map snippet in the list of results  if the mouse pointer is located over an enabled map icon. | 3 | - |
| 12 | The user must be able to view 1 spectral data set (simplified). | 3 | - |
| 13 | The user must be able to view 1 spectral data set (fullview –> diagram & metaparameter). | 1 | done |
| 14 | The user should be able to view multiple spectral data sets (fullview –> diagram & metaparameter). | 2 | done |
| 15 | The user must be able to view detail information about diagram curve  from the position where the mouse cursor is currently located. | 2 | done |
| 16 | The user should be able to zoom in into diagram. | 2 | done |
| 17 | Only existing categories should be shown. Categories without content must be fade out. | 1 | done |
| 18 | Spectral data from different types should be separated in different tabs. | 2 | done |
| 19 | File format of spectral data export must be csv (comma-separated values). | 1 | done |
| 20 | Multiple data sets from same type must be packed into one csv file. | 1 | done |
| 21 | All generated csv files (data sets from different type) must be packed to ZIP. | 2 | done |
| 22 | The user must be able to export detail view as PDF. | 2 | done |
| 23 | The detail view must contain a map displaying the locations of the spectral data per space. | 3 | done |
| 24 | The result list displays the found spectra with latitude and longitude in a map. | 3 | done |
| 24,1 | Clicking on a marker on the result map displays the detail view for the chosen spectrum. | 3 | done |
| 25 | Only the first 5 attributes per category should displayed initially on the detail view. | 2 | done |
| 25,1 | There has to be a button to show/hide the attributes after the first 5. | 2 | done |
| 26 | The WAR-File must be able to read a db\_config.txt file to access the database. | 2 | done |

Orange: Zusätzliche umgesetzte Anforderungen

Grün: Umgesetzte Anforderungen mit Priorität 3 (optional)

## Nicht implementierte Requirements

Wir konnten alle Basisanforderungen, sowie drei optionale Anforderungen implementieren. Acht dieser Anforderungen kamen während dem Projekt noch hinzu und verdrängten die bestehenden optionalen Anforderungen nach hinten.

Folgende optionale Anforderungen konnten wir nicht mehr implementieren:

* Die Anforderung Nr. 4, ist die Suche via Map, welche wir aus Zeitmangel nicht mehr implementieren konnten.
* Die Anforderungen Nr. 5 - 7, sowie 11 und 12, beziehen sich alle auf die Suche via Map (Anforderung Nr. 4) und konnten daher ebenfalls nicht implementiert werden.
* Die Implementation der Anforderung Nr. 2, war angedacht, da sie nur wenig Aufwand bedeutet hätte, jedoch unterstützt die API des Kunden zur Zeit noch gar keine solche Suche, weshalb wir diese ebenfalls weglassen mussten.

## Change Requests

Während dem Projekt kam es teilweise zu Anpassungen von Anforderungen oder komplett neuen Anforderungen durch den Kunden. Dank unserer agilen Arbeitsweise konnten wir auf diese Kundenbedürfnisse eingehen und konnten sie während der Entwicklung berücksichtigen. Hier ein paar Beispiele:

* Volltextsuche
* Filterung der Suchresultate durch die Attribute der einzelnen Kategorien.
* Anzeige einer Map mit Marker innerhalb der Detailansicht.
* Anzeige der ersten 5 Metainformationen pro Kategorie. Button um die kompletten Metadaten anzuzeigen bzw. auszublenden.

## Vorgehensmodell

Wir arbeiteten von Beginn an sehr eng mit dem Auftraggeber zusammen. Für die Definition des Auftrags führten wir einen Workshop durch, bei dem auch unser Betreuer teilnahm. In diesem Workshop wurden die Use Cases, Requirements, Ziel und die Vision erhoben. Aus diesen Angaben wurde dann ein Pflichtenheft[[7]](#footnote-7) erstellt, welches während der Projektdauer als roter Faden diente.

Danach ging die Arbeitsweise sehr agil vonstatten. Wir führten nach jedem Release eine Review mit dem Auftraggeber durch. Dadurch konnten wir Probleme früh erkennen und beim Auftraggeber ansprechen. Ausserdem wurden dadurch auch die Anforderungen fortlaufend angepasst und verfeinert. Teilweise war es auch möglich, dass der Kunde an der SPECCHIO API Änderungen vornehmen musste, damit die Anforderungen für das SPECCHIO Web Interface erfüllt werden konnten.

Die Kommunikation mit dem Auftraggeber wurde zu Projektbeginn festgelegt. Wir waren deshalb zwischen den Reviews per E-Mail mit dem Auftraggeber in Kontakt. Dabei ging es oftmals um Vorabklärungen für die anstehende Review oder um besprochene Details der vergangenen Review. Alle Review-Sitzungen wurden protokolliert und auf dem Git-Repository abgelegt.[[8]](#footnote-8) Die intensive Kommunikation trug wesentlich zum Gelingen der Umsetzung bei und verhinderte Leerläufe. Nachfolgend werden Details zur Planung und Arbeitsmethodik erläutert.

### Grobplanung

Mit einer ersten Grobplanung wurden die wichtigsten Termine grob abgesteckt. Da wir zu Beginn die Verfügbarkeit des Auftraggebers abgeklärt hatten, wussten wir, dass er ab Mitte Dezember 2016 im Ausland ist. Dementsprechend planten wir die Umsetzung der Basisanforderungen innerhalb des Zeitraums bis zum 16. Dezember 2017. Für die übrigen Anforderungen stand uns der Auftraggeber ab dem 17. Dezember weiterhin per E-Mail zur Verfügung.

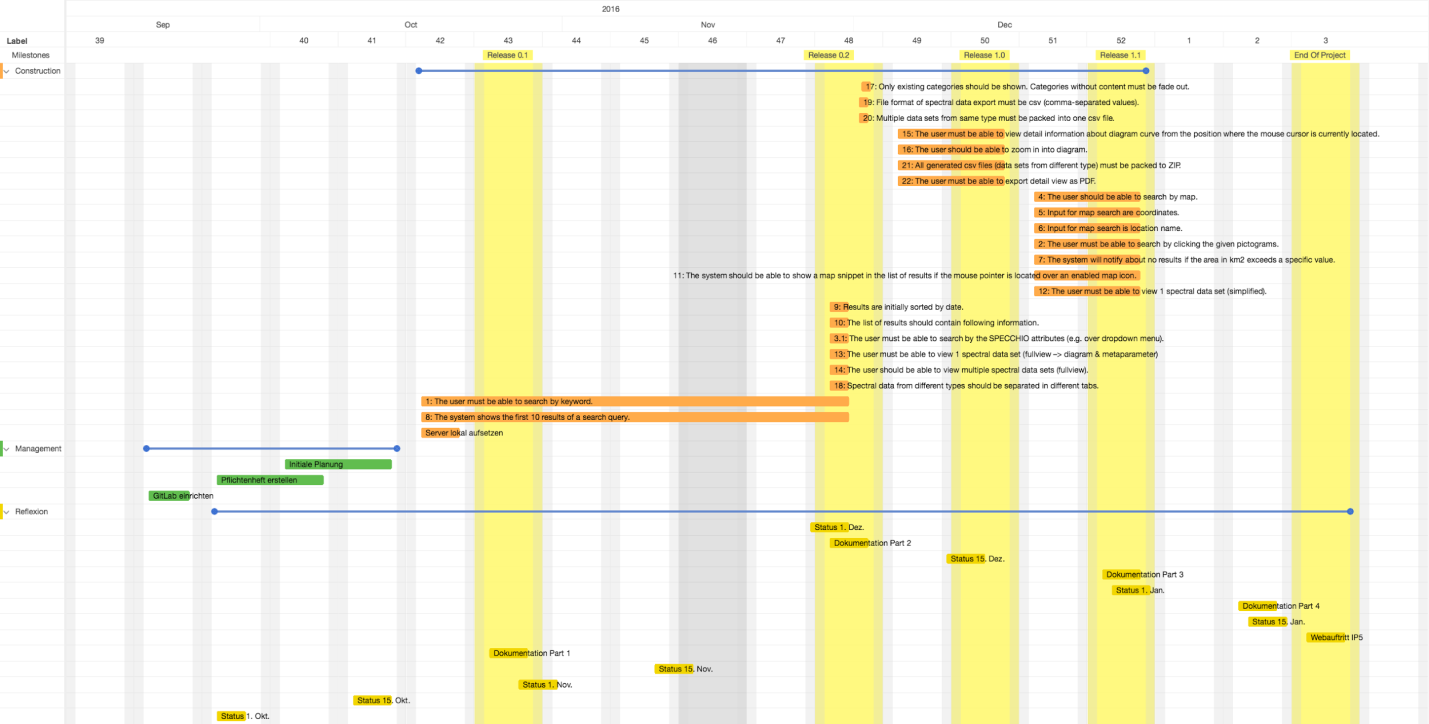
|  |  |  |
| --- | --- | --- |
| **Date** | **Milestone** | **Description** |
| 28.10.2016 | Release 0.1 | Rough prototype/proof of concept |
| 02.12.2016 | Release 0.2 | Most important requirements with priority 1 and 2 implemented and tested. |
| 16.12.2016 | Release 1.0 | Approval of requirements with priority 1 and 2. |
| 30.12.2016 | Release 1.1 | Requirements with priority 3 implemented. |
| 20.01.2016 | End of Project | Go live. |

Die den Auftraggeber betreffenden Termine wurden zu Beginn des Projekts kommuniziert.[[9]](#footnote-9)

### Detailplanung[[10]](#footnote-10)

Die Detailplanung wurde in drei Bereiche aufgeteilt:

1. Construction: Umsetzung der Arbeitspakete für die einzelnen Anforderungen
2. Management: Sitzungen, Requirement Engineering
3. Reflexion: Dokumentation, Abgabe der Zwischenberichte (Logfile)[[11]](#footnote-11)



### Release-Planung[[12]](#footnote-12) / Meilensteine

Es wurden 3 Releases während der Umsetzung der Anforderungen mit Priorität 1 und 2 geplant, sowie 2 weitere Releases für optionale Anforderungen und den finalen Release. Es hat sich gezeigt, dass für den finalen Release noch eine weitere Versionen nötig war.

* Release 0.1 (Priorität 1 und 2)
* Release 0.2 (Priorität 1 und 2)
* Release 1.0 (Priorität 1 und 2)
* Release 1.1 (optionale Anforderungen)
* Release 1.2 (final)
* Release 1.3 (final)

Die Release-Daten waren gleichzeitig unsere Meilensteine.

### Arbeitsmethodik

Um an der Umsetzung des Projekts zu arbeiten, trafen wir uns in der Regel wöchentlich am Campus der FHNW Brugg-Windisch.[[13]](#footnote-13) Für die Zusammenarbeit verwendeten wir das online Kollaborations-Tool Trello[[14]](#footnote-14). Hier konnten wir gleichzeitig auch die Detailplanung führen mit dem Planungstool[[15]](#footnote-15) elegantt.com. Alle Arbeitspakete waren in Trello in einem Kanban-Board erfasst und wurden in der folgenden Abfolge abgearbeitet:

1. Backlog
2. Next
3. In Progress
4. Staged
5. Approved
6. Done

Zur Zeiterfassung erstellten wir eine Google Tabelle auf Google Drive, welche wir dann alle zwei Wochen auf Git hochgeladen haben. Auch nutzten wir Google Drive für die einfache gemeinsame Bearbeitung anderer Dokumente.

### Usability Testing

Das Usability Testing wurden jeweils direkt in den Review-Sitzungen mit dem Kunden durchgeführt und das Feedback wurde im Protokoll festgehalten.

## Dokumentationen

Im Verlauf der Arbeit haben wir diverse Dokumente erstellt, welche auf Git abgelegt wurden. Hier sind die wichtigsten davon aufgelistet und kurz beschrieben.

|  |  |
| --- | --- |
| **Dokument** | **Inhalt** |
| Spezifikation | * Ziel und Vision * Use Case Diagramm * Use Case Beschreibung * Requirements |
| SAD | * Architektur der Applikation * Implementations-Informationen * User Guides |
| Testprotokoll | * Testcases für die Abnahmetests |
| Detail Planung und  Release Planung | * Zeitplanung über das Semester * Übersicht über Requirements und bis wann diese zu erfüllen sind |

Der genaue Dateiname und Ablageort dieser und weiterer Dokumente in Git ist im Kapitel 4.1 hinterlegt.

## Umsetzungsvarianten und Entscheidungen

Für die Umsetzung einiger Anforderungen, gab verschiedene Vorgehensweisen, zwischen denen wir uns entscheiden mussten. In diesem Kapitel gehen wir auf die wichtigsten Entscheidungen ein.

### Web Framework

Zu Beginn des Projektes stellte sich die Frage, ob wir die Web Applikation mit Hilfe eines Frameworks, wie Beispielsweise Spring MVC, aufbauen sollten. Da wir aber noch nicht viel Erfahrung mit neueren Web Frameworks gesammelt haben und auch unser Kunde damit nicht bewandert ist, haben wir uns schlussendlich dafür entschieden, die Applikation als einfache Java Webapplikation mit Servlets, Java Beans und JSP zu implementieren, was für deren Umfang vollkommen ausreichend ist. Wir haben uns dabei an die MVC Architektur gehalten. Dadurch hatten wir keinen zusätzlichen Aufwand durch das Aneignen von Know-How bezüglich eines Web Frameworks und das Schreiben eines User Guides dazu und konnte diese Zeit in die Umsetzung der Anforderungen investieren. Auch für den Kunden bedeutet dies weniger Aufwand, da er sich bei Wartung und Erweiterung der Applikation nicht noch in einen Guide zum Framework einlesen muss.

### Web Server und Version

Aktuell verwendet der Kunden einen GlassFish Server der Version 3.1, welche nicht Kompatible ist mit der Java Version 1.8. Daher haben wir uns zu Beginn die Frage gestellt, ob wir für unsere Applikation eine neuere GlassFish Version verwenden sollten, oder gar ganz ein anderer Web Server wie zum Beispiel Tomcat. Schlussendlich haben wir uns dann aber dafür entschieden, ebenfalls die GlassFish Version 3.1 zu verwenden, da der Kunde im Umgang damit bereits vertraut war und er so einfach eine neue GlassFish Instanz auf seinem Server einrichten musste anstelle einer neuen Web Server Installation. Somit muss sich der Kunde auch weiterhin nur um die Wartung von GlassFish 3.1 kümmern.

### CSS

Für das Styling unserer Views stellte sich die Frage, ob wir das gesamte CSS selbst schreiben, oder ob wir ein CSS Framework dazu verwenden sollten. Damit wir den Designaufwand relativ gering halten und trotzdem ein ansprechendes GUI implementieren konnten, entschieden wir uns schlussendlich dafür das Bootstrap Framework zu verwenden, welches bekannt ist für seine leichte Verwendbarkeit und gute Dokumentation. Ausserdem bringt es bereits eine minimale Responsiveness der Views mit.

### Graph

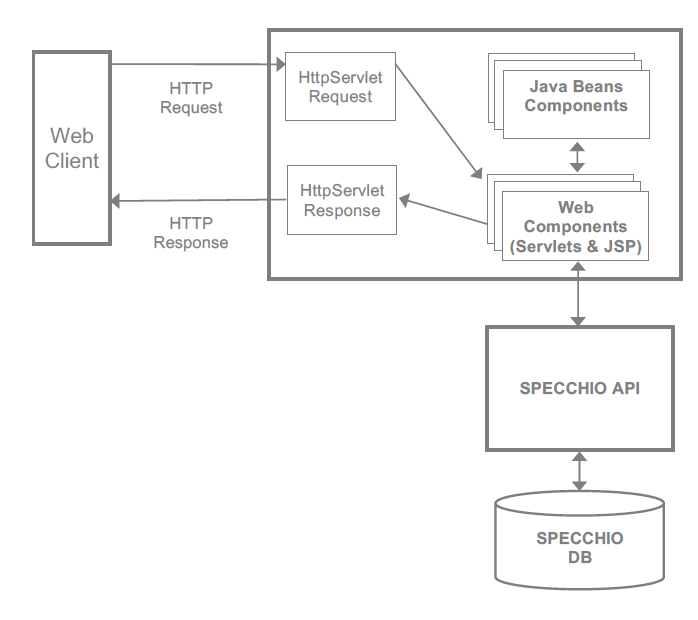
Für das Linien Diagramm der Spektraldaten hatten wir uns, nach Recherche im Internet, für die C3.js Chart Library entschieden, da sie einfach zu verwenden ist und die Funktionalität anbietet, welche wir brauchten. Sobald wir aber praxisnahe Testdaten verwendet hatten, merkten wir, dass die Datenmenge für C3 zu gross wurde, was zu Lags und unschöner Darstellung führte. Daher haben wir uns nach einer Alternative umgesehen und haben uns dann für Highcharts entschieden, welche auch kostenlos ist und mit grösseren Datenmengen problemlos umgehen kann.

### Map

Um herauszufinden, welche API wir für die Umsetzung der Landkarte verwenden sollen, haben wir beispielsweise Google Maps, Open Street Map und Leaflet angeschaut. Schlussendlich haben wir uns für Google Maps entschieden, da es am bekanntesten ist und uns das Vorgehen beim Anfragen eines API-Keys bereits bekannt war.

## Architektur

In der untenstehenden Grafik wird die grobe Architektur unserer Applikation aufgezeigt. Eine detaillierte Architektur mit Beschreibung der Elemente ist im SAD enthalten.



## Verwendete Technologien

|  |  |  |
| --- | --- | --- |
| Technologie | Verwendungszweck | Version |
| SPECCHIO API | Zugriff auf Datenbank und erstmalige Aufarbeitung der Daten | 3.2.1.6-Alpha |
| Java | Webapplikation-Backend. Aufarbeitung der Daten aus der SPECCHIO API fürs Frontend | 7 |
| JSP | Einbettung von Java Code in HTML-Frontend | 2.2 |
| Java Servlets | Request/Response Handling | 3.0 |
| Gson | Umwandlung von Java- in JSON-Objekte und zurück | 2.8.0 |
| JavaScript | Dynamische Erzeugung von GUI Elementen und Bereitstellung von GUI Funktionalität | Browser-spezifisch |
| JQuery | Dynamische Erzeugung von GUI Elementen und Bereitstellung von GUI Funktionalität | 2.1.4 |
| Bootstrap | CSS Framework zur Darstellung des User Interfaces | 3.3.7 |
| Highcharts.com | Darstellung der Liniendiagramme | latest |
| Google Maps API | Darstellung der Landkarten | latest |
| HTML | Strukturierung des Frontends | HTML5 |
| Oracle Glassfish | Webserver | 3.1 |

## Deployment

Für den Betrieb der Produktionsumgebung ist der Auftraggeber verantwortlich. Ebenso für deren Sicherung/Backup. Zum deployen des SPECCHIO Web Interfaces von der Entwicklungsumgebung in die Produktionsumgebung, haben wir im SAD einen Guide erstellt. Dieser Umfasst unter anderem den WAR-File Export aus Eclipse und das Deployment des WAR-Files auf den GlassFish Server.

## Aufgetretene Probleme

Bei der Umsetzung des Projekts sind wir auf verschiedene Probleme gestossen. Nachfolgend wird auf einzelne Fälle eingegangen.

### Performance

Die Haupteinbussen bei der Performance entstanden durch die API Aufrufe. Dies machte sich vor allem bemerkbar, als wir über die API Metadaten einzelner Attribute abrufen wollten. Das Problem dabei war, dass die API nur eine Methode anbot, bei der man für Spektren den Wert eines einzelnen Attributes abfragen konnte. Da wir aber die Werte mehrerer Attribute brauchten mussten wir diese Methode entsprechend oft aufrufen. Um dieses Problem zu beheben schlugen wir dem Kunden vor, die API um eine Methode zu erweitern, welche mit einer Liste von Attributen aufgerufen werden kann. Der Kunde stimmte zu und setzte dies um.

### Einlesen von db\_config.txt

Eine Anforderung des Kunden war es, dass unser WAR-File ein db\_config.txt File, welches zur Verbindung zur Datenbank verwendet wird, lesen könne. Dies hat zum Vorteil, dass die Zugangsdaten zur Datenbank (db\_config.txt) nicht im WAR-File verpackt werden, sondern gesondert auf dem Server abgelegt werden können. Dadurch können die Zugangsdaten unabhängig vom WAR-File geändert werden.

Das Problem bestand einerseits darin, dass wir keinen Zugriff auf die Produktionsumgebung hatten und deshalb nur mit den Informationen des Auftraggebers auskommen mussten. Testen konnten wir also nicht selber. Die andere Schwierigkeit bestand darin, auf dem Filesystem ausserhalb der SPECCHIO Web Interface-Struktur eine Datei einzulesen, die zugleich die Mittel der SPECCHIO API nutzt.

Die Lösung bestand darin, mehrere Methoden zu schreiben, die fähig sind zum richtigen Zeitpunkt den aktuellen Dateipfad zu lesen und diesen zu einem späteren Zeitpunkt richtig zusammen zu setzen.

### PDF Export

Ein Rückschlag war der Client-seitige PDF-Export aus dem Webbrowser. Es war nicht möglich einen äquivalenten Export der Detail View zu machen. Eine mögliche Lösung mit jsPDF[[16]](#footnote-16) funktionierte nicht, weil das Liniendiagramm nicht mit dieser Library dargestellt werden konnte. Es hätte einen zu hohen Aufwand erfordert, selber eine von Grund auf neue Lösung zu programmieren.

Die Lösung bestand darin über den Druckdialog des Browsers zu leiten. In Absprache mit dem Auftraggeber konnten wir auf diese Weise eine angemessene Lösung finden.

### Git Repository

Zu Beginn des Projektes hatten wir noch kleinere Probleme um Schreibzugriff auf das für uns erstellte Git Repository zu erhalten, dies konnte dann aber durch Herr Gwerder behoben werden. Im Laufe des Projektes kam es zu Problemen beim merging, zwischen unseren Änderungen. Dies lag wahrscheinlich daran, dass das .gitignore File nicht richtig konfiguriert wurde wodurch Files von einem Workspace gepushed wurden, welche dann im anderen Workspace zu Problemen führte. Schlussendlich entschieden wir uns dafür Sourcecode Änderungen nur noch von einem Workspace aus zu pushen.

## Testing

Während dem Entwickeln testeten wir laufend unsere Anpassungen in diversen Browsern. Der Kunde konnte die aktuelle Version einerseits in den Kundenmeetings direkt testen. Ausserdem sendeten wir ihm regelmässig ein WAR-File des aktuellen Releases, damit er es bei sich deployen und testen konnte. Das Feedback des Kunden erhielten wir daher direkt in den Meetings oder per Mail.

Für den Release 1.0, welcher alle Grundanforderungen enthielt, erstellten wir ein Testprotokoll mit Testcases, welche von uns und dem Benutzer durchgespielt wurden. Schlussendlich musste dieses vom Benutzer für die Abnahme unterschrieben werden.

## Bugfixing

Da die Applikation während des Projektes laufend durch uns und den Kunden getestet und Feedback eingeholt wurde, zog sich auch die Fehlerbehebung übers ganze Projekt hinweg. Teilweise konnten wir auch direkt in den Kundenmeetings mit dem Kunden Bugs identifizieren und beheben.

# Schluss

## Rückblick

Zu Beginn war uns noch nicht ganz klar, was vom Kunden genau gewünscht ist, daher waren wir unsicher, ob wir die Bedürfnisse des Kunden erfüllen werden. Das Thema Spektraldaten war für uns beide Neuland und wir wussten nicht wie viel Wissen wir in diesem Gebiet brauchten, um die Applikation zu implementieren. Die API des Kunden ist sehr gross und nicht optimal beschrieben, was die Programmierarbeit teilweise verzögert hat. Durch die gute Kommunikation und die regelmässigen Meetings mit dem Kunden, konnten wir aber viele Fragen klären und haben nicht viel Zeit verloren. Der Kunde war sehr sympathisch und kooperativ. Die enge Zusammenarbeit mit dem Kunden gefiel uns sehr gut.

Wir wurden nicht stark durch Vorschriften des Betreuers eingeschränkt und hatten deshalb viele Freiheiten und die Möglichkeit selbständig zu arbeiten, was wir sehr geschätzt haben. Bei allfälligen Fragen konnten wir uns stets an unseren Betreuer wenden.

Eine sinnvolle Entscheidung war, das Deployment der Applikation in der produktiven Umgebung schon früh anzugehen. Denn es entstanden Schwierigkeiten, die dadurch schon früh in der Projektphase erkannt und angegangen werden konnten.

Da unser Kunde ab Mitte Dezember bis Projektende ausser Landes sein würde, planten wir bis dahin den Release 1.0, welcher alle Grundanforderungen umfasste, fertig zu stellen, um diese dem Kunde nochmals persönlich präsentieren zu können. Danach konnten wir ihn nur noch per Mail erreichen, wobei es wegen der Zeitverschiebung oftmals zu Verzögerungen kam. Schlussendlich konnten wir die Übergabe und Abnahme dennoch problemlos per Mail und DropBox durchführen. Das Feedback des Kunden war sehr positiv.

## Lessons Learned

Die Probleme, welche wir beim Pull und Push mit Git hatten, lernte uns, dass man von Beginn an das Projekt sauber aufsetzen muss. Die Probleme sind nicht bloss nur nervenaufreibend, sondern auch sehr zeitraubend.

Das Thema Spektraldaten und die bestehende SPECCHIO API waren nicht einfach zu verstehen. Viele Fragen zur Thematik klärten wir zu Beginn direkt beim Kunden. Erst im Verlaufe des Projekts nutzten wir auch intensiver die bestehenden Dokumentation zu den einzelnen Komponenten von SPECCHIO. Bei einem zukünftigen Projekt würden wir gleich zu Beginn auf diese Unterlagen zurückgreifen, um sich in kurzer Zeit gleich von Beginn weg ein Bild über die Thematik machen zu können. Die Gefahr besteht dann zwar, dass man sich evtl. ein falsches Bild resp. falsche Annahmen macht, aber im Gespräch mit dem Kunden weiss man dann trotzdem schon eher von was er spricht, wenn in der Diskussion bestimmte Begriffe fallen.

Während der Umsetzung des SPECCHIO Web Interfaces kam es mehrere Male vor, dass die SPECCHIO Applikation auf der Virtual Machine aktualisiert werden musste. Dies erforderte ein explizites Know-how und beanspruchte deshalb zusätzliche Zeit, die wir nicht einkalkuliert hatten. In Zukunft müssen wir klare Zeitfenster für den Erwerb von Know-how und den Support der projektbezogenen Systemen einplanen.

## Ausblick

Der Kunde nahm in den Review-Sitzungen die Rolle des Users ein. Die Bedienung des SPECCHIO Web Interface basiert somit aus der Optik und der Erfahrung des Kunden. Bei der Weiterentwicklung des Projekts wäre der Einbezug von weiteren Benutzern sinnvoll, mit welchen die Applikation getestet werden kann, um die Bedienbarkeit gegebenenfalls zu verbessern.

Die optionalen Anforderungen könnten in einer Weiterführung des Projekts umgesetzt werden. Es handelt sich dabei primär um die Suche via Map. Auch wäre eine Idee, das Bearbeiten der Spektraldaten über die Webapplikation zu implementieren, anstatt nur das Anzeigen. Dazu müsste die Applikation wohl auch noch um ein Login erweitert werden.

Die Responsiveness des SPECCHIO Web Interface war keine Anforderung und wurde hier nur minimal behandelt. Deshalb wäre eine Möglichkeit, die Responsiveness noch zu optimieren, damit auch bei kleinen Display-Grössen die Darstellung ausreicht.

## Schlusswort

Das Projekt hat uns gut gefallen und wir sind sehr zufrieden mit dem Ergebnis. Zu Beginn waren wir unsicher, ob wir überhaupt alle Grundanforderungen erfüllen können, doch schlussendlich konnten wir sogar noch einige optionale Anforderungen umsetzen. Wir konnten viel neues lernen, auch wenn es manchmal sehr zeitaufwändig war. Wir sind vor allem auch sehr froh, dass es dem Kunden so gut gefällt. Somit, war dieses Projekt für uns eine äusserst positive Erfahrung.

# Anhang

## Artefakt-Referenzen

|  |  |
| --- | --- |
| **Artefakt** | **Pfad** |
| Detail Planning.png | … / 05\_Planning |
| FHNW Projektbeschrieb IMVS17 SPECCHIO Webinterface.pdf | … / 01\_Admin |
| Logfile.xlsx | … / 03\_Arbeitsjournal |
| Release Planing.xlsx | … / 05\_Planning |
| ReviewDates\_SPECCHIO.docx | … / 05\_Planning |
| Specification\_SPEECHIO | … / 04\_Specification |
| Protokoll\_SPECCHIO\_Nr02\_20160922.docx | … / 02\_Meetings |
| Test-Protokoll.docx | … / 10\_Transition |

## Glossar

|  |  |
| --- | --- |
| Begriff | Erklärung |
| Git | Zentrale Ablageverwaltung unserer Projektdaten und des Source Code. |
| MVC | Model View Controller |
| SPECCHIO | Ist ein Informationssystem zur Aufnahme von Referenzspektren und spektralen Kampagnendaten, die durch Spektroradiometer erfasst werden. Die Einbindung von umfangreichen Metadaten-Sets sorgt für die dauerhafte Haltung von Spektraldaten und ermöglicht die gemeinsame Nutzung von Spektraldaten zwischen Forschungsgruppen. Auf die Daten von SPECCHIO  kann zur Zeit nur mit einem spezialisierten Programm zugegriffen werden. |
| SPECCHIO Web Interface | Ermöglicht den direkten Zugriff auf die SPECCHIO-Datenbank ohne Installation zusätzlicher Software. Es sind nur Abfragen möglich, es können keine Daten editiert werden. |
| WAR | Ist ein Dateiformat und steht für Web Application Archive. Dieses Format beschreibt wie eine vollständige Webanwendung nach der Java-Servlet-Spezifikation in eine Datei im JAR- bzw. ZIP-Format verpackt wird (www.wikipedia.de) |

## Ehrlichkeitserklärung

Hiermit erklären wir, die vorliegende Projektarbeit selbständig, ohne Hilfe Dritter und nur unter

Benutzung der angegebenen Quellen verfasst zu haben.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Ort/Datum |  | Remo Rossi |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Ort/Datum |  | Christian Schibli |

1. http://www.specchio.ch [↑](#footnote-ref-1)
2. Git: FHNW Projektbeschrieb IMVS17 SPECCHIO Webinterface.pdf [↑](#footnote-ref-2)
3. Git: Specification\_SPECCHIO [↑](#footnote-ref-3)
4. Git: Protokoll\_SPECCHIO\_Nr02\_20160922.docx [↑](#footnote-ref-4)
5. Specification\_SPECCHIO [↑](#footnote-ref-5)
6. Git: Requirements.xls [↑](#footnote-ref-6)
7. Git: Specification\_SPECCHIO [↑](#footnote-ref-7)
8. Git: 02\_Meetings [↑](#footnote-ref-8)
9. Git: ReviewDates\_SPECCHIO.docx [↑](#footnote-ref-9)
10. Git: Detail Planning.png [↑](#footnote-ref-10)
11. Git: Logfile.xlsx [↑](#footnote-ref-11)
12. Release Planing.xlsx [↑](#footnote-ref-12)
13. Logfile.xlsx [↑](#footnote-ref-13)
14. https://trello.com [↑](#footnote-ref-14)
15. https://elegantt.com/ [↑](#footnote-ref-15)
16. https://parall.ax/products/jspdf [↑](#footnote-ref-16)