Realisierungsbericht

|  |  |
| --- | --- |
| **Status** | In Arbeit / In Prüfung / Abgeschlossen |
| **Projektname** | Unsustainable |
| **Projektleiter** | Michael Günter, Elias Schmidhalter |
| **Auftraggeber** | Andres Scheidegger |
| **Autoren** | Michael Günter, Elias Schmidhalter |
| **Verteiler** | Michael Günter, Elias Schmidhalter, Andres Scheidegger |

**Änderungskontrolle, Prüfung, Genehmigung**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Beschreibung, Bemerkung | Name oder Rolle |
| 0.1 | 27.04.2015 | Erstfassung | Michael Günter |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Definitionen und Abkürzungen**

|  |  |
| --- | --- |
| Begriff / Abkürzung | Bedeutung |
|  |  |

**Referenzen**

|  |  |
| --- | --- |
| Referenz | Titel, Quelle |
| [1] |  |
| [2] |  |
| [3] |  |

**Inhaltsverzeichnis**

[1 Zusammenfassung 3](#_Toc414298997)

[2 Technische Detailspezifikation 3](#_Toc414298998)

[2.1 Systemdesign 3](#_Toc414298999)

[2.1.1 Struktur 3](#_Toc414299000)

[2.1.2 Dynamik 3](#_Toc414299001)

[2.2 Schnittstellendefinitionen 4](#_Toc414299002)

[2.3 Datenmodell 4](#_Toc414299003)

[3 Systemdokumentation 4](#_Toc414299004)

[3.1 Anwendungshandbuch 4](#_Toc414299006)

[3.1.1 Übersicht 4](#_Toc414299007)

[3.1.2 Funktionen und Detailbeschreibung 4](#_Toc414299008)

[3.1.3 Fehlerbehandlung 4](#_Toc414299009)

[3.2 Integrations- und Installationshandbuch 5](#_Toc414299010)

[3.3 Betriebshandbuch 5](#_Toc414299011)

[4 Systemtest 5](#_Toc414299012)

[4.1 Unit-Test 5](#_Toc414299014)

[4.2 Systemtest 5](#_Toc414299015)

[4.2.1 Testfälle 5](#_Toc414299016)

[*Testfall 1 „Neuen Benutzer registrieren und anmelden“* 6](#_Toc414299017)

[*Testfall 2 „Bestehenden Benutzer registrieren“* 6](#_Toc414299018)

[5 Testprotokoll 7](#_Toc414299019)

[*5.1* *Systemtest 1* 7](#_Toc414299020)

[*5.1.1* *Testfall 1 „Neuen Benutzer registrieren und anmelden“* 7](#_Toc414299021)

[*5.1.2* *Testfall 2 „Bestehenden Benutzer registrieren“* 7](#_Toc414299022)

[6 Weiterführung der Projektplanung 7](#_Toc414299023)

[6.1 Abgleich von Planung und tatsächlichem Verlauf der Phase Konzept 7](#_Toc414299024)

[6.2 Aktualisierung der Risikosituation 7](#_Toc414299025)

[6.3 Planung der nächsten Phase 7](#_Toc414299026)

[Anhang A: Quellcode 8](#_Toc414299027)

[Anhang B: Testcode Unit-Tests 8](#_Toc414299028)

**Abbildungsverzeichnis**

# Zusammenfassung

Der Realisierungsbericht dient zur Dokumentation der grundsätzlichen technischen Struktur der Applikation. Auch Informationen zur Benutzerführung und zur Applikationsinstallation, sowie Informationen zum gesamten Testprozess werden in diesem Dokument festgehalten.

Geben Sie hier eine kurze Zusammenfassung des Inhalts dieses Dokumentes. Wozu dient das Dokument (Zweck) und welche Informationen enthält es?

# Technische Detailspezifikation

## Systemdesign

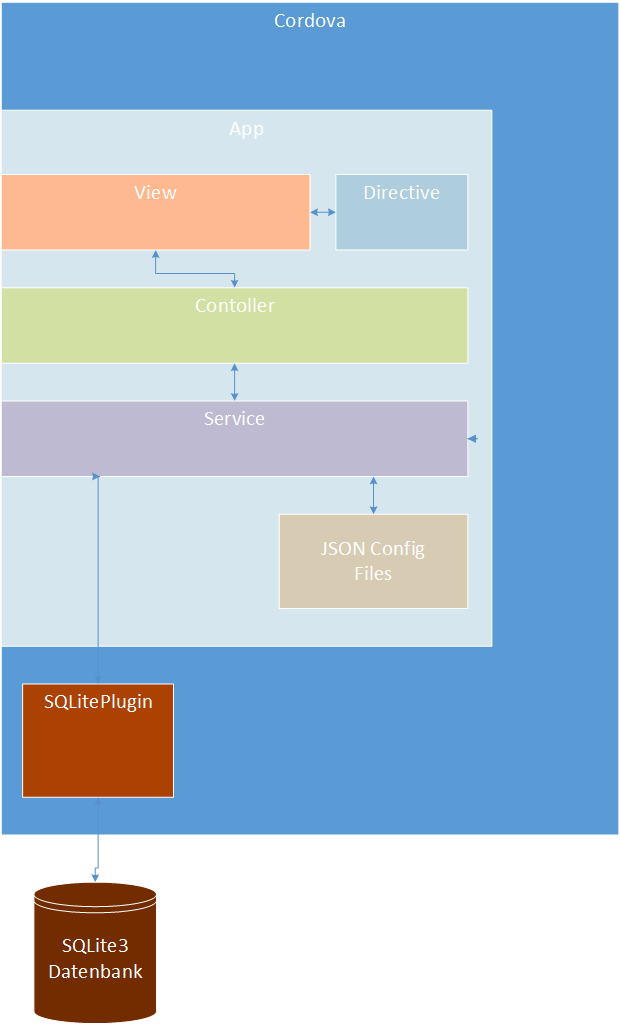
Das Systemdesign beschreibt die statische Struktur der Applikation sowie die grundlegende Dynamik. Es stellt dar, wie die verschiedenen Module miteinander interagieren.

Im Konzeptbericht haben Sie die Systemarchitektur entworfen. Dort haben Sie beschrieben aus welchen Elementen (Schichten, Pakete, Klassen, Module) Ihr System grob besteht. Während der Realisierung des Systems verfeinern Sie diese Architektur schrittweise. Weitere Elemente kommen hinzu, andere müssen aufgeteilt, oder anders angeordnet werden.

Hier geht es nun darum die am Ende der Realisierung gültige Systemarchitektur und den Systemdesign zu beschreiben. Dazu gehören die statische Struktur des Systems und die Dynamik. Der zweite Punkt ist vor allem bei objektorientierter Umsetzung wichtig.

### Struktur

Hervorgehend aus dem Konzeptbericht haben wir eine grundlegende Struktur definiert.

Auf diesem Konzept haben wir aufgebaut, und wir haben unsere Lösung in diese grundlegenden Module unterteilt.

Wie gehabt umschliesst Apache Cordova unsere gesamte Applikation und stellt die Ausführung auf den mobilen Betriebssystemen sicher. Danach ist die Lösung unterteilt in die Module App, View, Directive, Controller und Services. Ausserdem existiert eine SQLite Datenbank für die Speicherung des Spielfortschritts und für die Bereitstellung von Stammdaten.

Die Grafik links, ist jedoch nicht mehr dieselbe wie im Konzeptbericht. Es fällt auf, dass zwischen den Services und dem SQLitePlugin kein ngCordova mehr ist. Wir haben uns während der Entwicklung dazu entschlossen ngCordova wegzulassen, weil wir es eigentlich nur für das SQLitePlugin brauchen und wir dies viel einfacher selber von Hand einbinden können, indem wir einen SQLiteService schreiben, der mit dem Plugin interoperiert und ein besseres API zur Verfügung stellt.

**View**

Eine View ist ein HTML-Dokument, welches verschiedene UI Komponenten anordnet und grafisch aufbereitet darstellt.

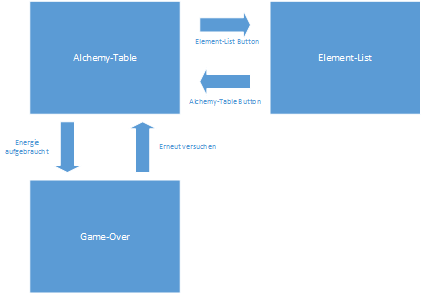
Durch die Benutzung von AngularJS haben wir die Möglichkeit direkt in den Views bestimmte Aktionen durchzuführen.

Wir können zum Beispiel verschiedene Directives in eine View einbinden. Eine Directive ist hat meist ein eigenes Layout und eigene Logik. Sie stellt grundsätzlich eine zusammengefasste UI-Komponente dar.

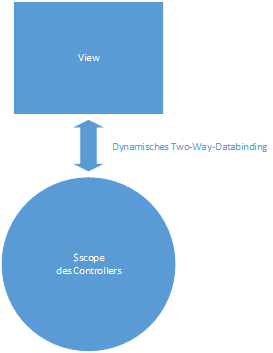
Wir haben genau 3 Views. Einmal der normale Alchemy Table, einmal die Element-Liste und einmal der Game-Over Screen. Zu den Views zählen technisch gesehen auch die Templates für die Directives. Diese werden jedoch später genauer beschrieben.

Im Hintergrund jeder View arbeitet ein korrespondierender Controller, der die JavaScript-Logik implementiert.

Die 3 Views hängen folgendermassen zusammen:



Im Hintergrund aller Views arbeitet ein Controller an der Anwendung der Logik. Der Controller besitzt Variablen auf dem $scope, die direkt auf der View mittels eines dynamischen Two-Way-Databindings verknüpft sind. Heisst, wenn auf dem UI ein Wert verändert wird, kann der neue Wert im Controller unmittelbar verwendet werden. Das gleiche gilt umgekehrt: Wenn im Controller der Wert verändert wird, wurde die Änderung auch direkt im UI übernommen. Dieses Databinding gilt für alle Variablen auf dem $scope des Controllers.



Aber das Two-Way-Databinding wirkt keineswegs nur zwischen View und Controller, sondern auch innerhalb des Controllers und innerhalb der View, wenn zum Beispiel an zwei Stellen auf dieselbe Variable zugegriffen wird. Auch die Argumente für Directives sind nach Wahl ein Databinding und ändern so direkt die Variable im Directive und auch umgekehrt, Änderungen im Directive werden über das Argument zurück in die View reflektiert.

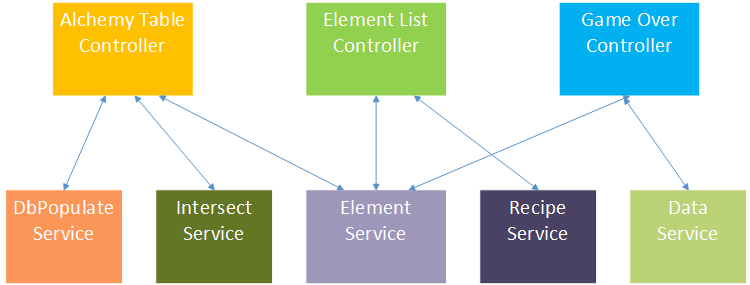
Wir haben uns entschlossen unser ViewModel (Die Variablen auf der View) nicht direkt auf das $scope zu legen, weil es dann leicht versehentlich die globalen Variablen auf dem $rootScope oder die Variablen von hierarchisch höher gelegenen $scopes, überschreiben könnte. Deshalb haben wir ein ViewModel Objekt eingeführt, auf welchem wir unser ViewModel ablegen. Wir schreiben also anstelle von $scope.test -> $scope.vm.test, wobei vm für ViewModel steht.

**Controller**

Der Controller bereitet die Daten aus den Services für die Anzeige auf der View auf. Der Controller schreibt die Daten auf das ViewModel. Das ViewModel ist via Data-Binding an die View gebunden.

Der Controller kann auf verschiedene Services zugreifen. Die Services werden via Dependency Injection eingebunden. Für jede View wurde auch ein Controller erstellt.

Für jede View existiert ein korrespondierender Controller. In folgender Abbildung ist ersichtlich, welche Services von welchen Controllern verwendet werden.



Die spezifischen Aufgaben der Services werden später in diesem Dokument beschrieben.

**Service**

Der Service beinhaltet die Business Logik des Programmes. Hier wird auf die Persistenz Schicht zugegriffen, Daten aufbereitet und eingaben von den Controllern bearbeitet.

Wir haben grundsätzlich den SQLite Service welcher den zugriff auf die Datenbank sicherstellt. Der SQLite Service baut eine Datenbankverbindung zum Datenbankprovider auf. Dieser ist auf den mobilen Endgeräten der lokale SQLite3 Dienst. Wenn die Applikation im Browser gestartet wird, wird, falls vorhanden WebSQL verwendet.

Der SQLite Service bietet eine Query Methode an. Diese kann ein prepared SQL-Query auf der Datenbank ausführen. Zusätzlich gibt es eine Chain Methode, die mehrere Queries nacheinander in der richtigen Reihenfolge ausführt.

Darauf bauen einige Services wie zum Beispiel der DbPopulate Service dieser befüllt die Datenbank mit Stammdaten. Er ist aufgeteilt in die Erstellung des Datenbankschemas und andererseits in das befüllen der Datenbank mit Stammdaten. Der DbPopulate Service wird nur aufgerufen, wenn die App das erste mal gestartet wird.

Der Dataservice erstellt anhand von SQL Abfragen Business Entitäten (JSON Objects).

In diesem Service sind die SQL-Statements abgelegt. Der Dataservice verwendet den SQLite Service um die abgelegten Queries auf der Datenbank auszuführen.

Zu den Methoden des Dataservice gehören folgende:

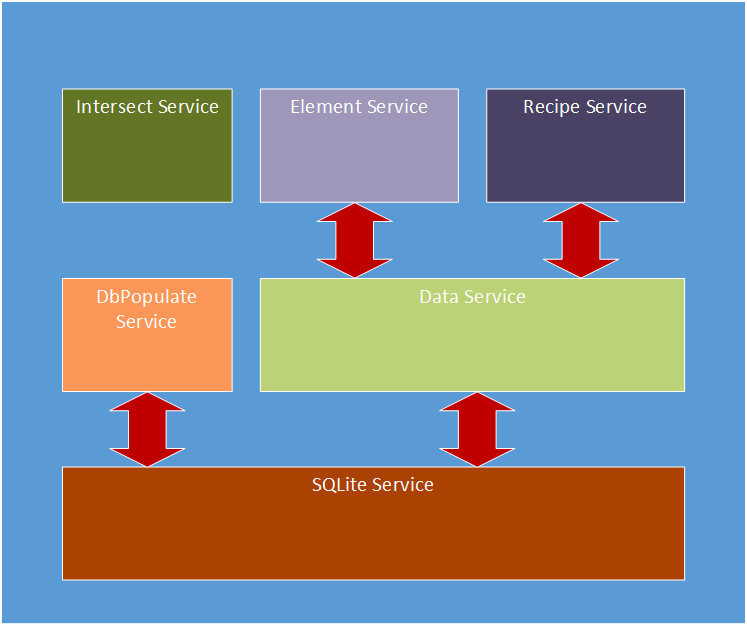
* getCurrentElements, holt alle Elemente, die sich aktuell auf dem Alchemytable befinden
* getAllElements, holt alle Elemente
* getCombinedElement, holt anhand von zwei Elementen das kombinierte Element
* getElementParts, holt anhand des kombinierten Elements die beiden Element Teile
* getBaseElements, holt alle Elemente ohne Rezept
* getBaseElementsExcept, holt alle Elemente ohne Rezept, ausser ein definiertes Element
* isBaseElement, gibt true zurück wenn das Element ein Basis-Element ist
* restoreBaseElements, setzt die aktuellen Elemente auf dem Alchemy Table zurück auf die Basis-Elemente
* updateCurrentElement, setzt die Position des übergebenen Elements
* updateCurrentEnergy, setzt die übrige Energie des Spielers
* getUnlockedRecipes, holt alle freigeschaltenen Rezepte und die zugehörigen Elemente
* unlockRecipe, schaltet ein Rezept frei

Dieser Dataservice wird vom Element und Recipe Service verwendet. Diese bieten eine Schnittstelle zwischen dem Datenservice und den Controllern.

Im Element Service befindet sich die Logik zum Kombinieren, Teilen und Zurücksetzen von Elementen. Ansonsten ist er eine pure Schnittstelle zum Dataservice.

Ausserdem besteht noch der Intersect Service dieser macht keine Zugriffe auf die Datenbank, er überprüft lediglich ob zwei Elemente sich überschneiden.

Dabei existieren die Überprüfung zur Überschneidung von zwei Elementen, sowie eine Überprüfung für zwei Positionen.



**Directive**

Direktiven sind zusammengefasste UI-Komponenten. Für dieses Projekt haben wir insgesamt drei verschiedene Direktiven erstellt.

Die Logik der Directives befindet sich in den einzelnen JavaScript Dateien. Die HTML Datei nennen wir Template. Das Template wird vom Directive referenziert. Einem Directive können verschiedene Parameter übergeben werden. Dies erfolgt direkt auf der View, wo das Directive eingebunden wurde. Diese sind meist dynamisch an das ViewModel des Controllers gebunden, und Änderungen des Wertes in der Directive werden unmittelbar auf das ViewModel des Controllers zurückreflektiert.

Alle Directives werden auf dem Alchemy Table verwendet.

*Element Directive*

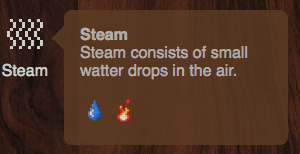
*\\10.2.7.212\Unsustainable\doc\res\fireelementImage.png*Für die Darstellung und die Interaktion mit Elementen haben wir eine Element Direktive erstellt.

Die Logik für das Verschieben von Elementen, sowie der Aufruf für das Kombinieren und Splitten

von Elementen wird in dieser Direktive implementiert.

Die Abbildung zeigt das Aussehen der Direktive auf dem Alchemy Table.

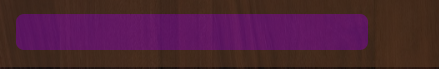
*Flupp Directive*

Für die Darstellung detaillierter Informationen haben wir ein Flupp Directive erstellt. Das Flupp Directive ist eine Art Popout oder Popover, welches erscheint wenn kurz auf das Element getippt wird.

In dem Directive ist Logik für das Öffnen und Schliessen des Flupps, aber auch für die Darstellung der Daten vorhanden.

Die Abbildung zeigt das Aussehen der Direktive nachdem das Element angetippt wurde.

*Energybar Directive*

**

Für die Darstellung der Energie Leiste haben wir eine Direktive erstellt. Das Directive dient nur dazu, die Bar richtig darzustellen und konstant zu aktualisieren. Die Abbildung zeigt die Energie Leiste, die auf dem Alchemy Table dargestellt wird.

**App**

Im App-Modul werden globale Konfigurationen vorgenommen. Darunter fallen die Konfiguration für die Datenbank, die Routen und der App Start.

*Routen*

Die Routen befinden sich im routes.js und definieren für jede View einen Controller und unter welchem State diese erreichbar sind.

*App Start*

Beim Appstart wird das Device Ready von Cordova abgewartet, falls die App auf einem mobilen Endgerät ausgeführt wird. Dies ist notwendig, da ansonsten der Datenbankzugriff über das Plugin nicht gewährleistet ist. Anschliessend wird die App gestartet.

Das Index-HTML ist das erste Dokument, das von Cordova aufgerufen wird. Hier wird grundsätzlich der Bereich definiert, in dem später die Views eingebunden werden. Ausserdem werden die Styles (CSS) eingebunden. Wichtige Einstellungen für den Viewport werden auch hier vorgenommen. Die Scripts werden eingebunden. Dabei ist zu beachten, dass wir lediglich das main.js einbinden, welches vorher gebundelt wurde. Zusätzlich muss cordova.js eingebunden werden, um die Funktionalität von Cordova sicherzustellen. Diese Datei existiert nur auf dem mobilen Endgerät und ist im Source-Code nicht ersichtlich.

Beschreiben Sie ausgehend von der Systemarchitektur den strukturellen Aufbau ihres Systems. Wenn Sie objektorientiert entwickeln, verwenden Sie dazu UML-Klassendiagramme und Paketdiagramme. Andernfalls verwenden Sie Blockdiagramme, welche die einzelnen Module und ihre Funktionen/Prozeduren, sowie die Aufrufbeziehungen zwischen den Modulen zeigen (siehe auch Beispiele im Konzeptbericht).

**Hinweis**: Es muss nicht die hinterste und letzte Hilfsklasse oder Hilfsfunktion in den Diagrammen erscheinen. Vielmehr sollen diejenigen Elemente gezeigt werden, welche nötig sind, um das System zu verstehen. Sie müssen Ihr System also auf einer sinnvollen Ebene abstrahieren.

**Hinweis**: Wenn Sie mit einer Framework oder mit externen Libraries arbeiten, so müssen Sie in der Regel auch Elemente der Framework oder der Library, welche unmittelbar mit Ihrem System zusammenhängen, in Ihre Diagramme aufnehmen. Sonst wird man Ihr System kaum verstehen.

**Wichtig**: Arbeiten Sie mit mehreren Diagrammen. Geben Sie zu jedem Diagramm eine kurze Beschreibung dessen, was Sie mit dem Diagramm darstellen wollen.

**Wichtig**: Dieser Abschnitt enthält nicht einfach nur Diagramme. Sie müssen je nachdem auch einzelne Elemente im Text näher beschreiben (Aufgaben des Elementes, Designentscheidungen, …). Dies gilt vor allem für Elemente, deren Zweck nicht direkt aus den Diagrammen hervorgeht. Denken Sie daran, dass Sie dem Leser, z.B. dem IPA-Experten oder der Person, welche Ihr System erweitern soll, erklären wollen, wie Ihr System gebaut ist und wie es arbeitet.

### Dynamik

Oft reicht eine rein strukturelle Beschreibung nicht aus, um das System zu verstehen.

Illustrieren Sie also hier die wichtigsten Abläufe in Ihrem System z.B. mit Hilfe von Sequenzdiagrammen oder allenfalls Struktogrammen.

Bei der Auswahl der zu illustrierenden Abläufe können Sie sich folgende Fragen stellen:

* Welches ist der komplexeste Anwendungsfall im System? Illustrieren Sie diesen Ablauf mit einem Sequenzdiagramm. Meist wird es sich dabei um den zentralen Anwendungsfall Ihres Systems handeln.
* Gibt es in Ihrem System Schnittstellen zu anderen, allenfalls externen Systemen, wo es darauf ankommt eine bestimmte Meldungsreihenfolge einzuhalten? Oder müssen gar Meldungen auf verschiedenen Schnittstellen koordiniert werden? Setzen Sie auch hier ein Sequenzdiagramm zur Illustration ein.
* Gibt es in Ihrem System komplizierte Algorithmen? Hier eignet sich allenfalls ein Struktogramm zur Illustration.

**Wichtig**: Arbeiten Sie mit mehreren Diagrammen. Geben Sie zu jedem Diagramm eine kurze Beschreibung, dessen, was Sie mit dem Diagramm darstellen wollen.

**Wichtig**: Auch hier gilt, abstrahieren Sie auf der richtigen Ebene. Generierte Sequenzdiagramme, welche die hinterste und letzte Zeile Code illustrieren sind unnütz.

## Schnittstellendefinitionen

Im Konzeptbericht haben Sie bereits Schnittstellen beschrieben. Wir haben uns dabei auf Schnittstellen der folgenden Arten konzentriert:

* Schnittstellen zu externen Systemen
* Schnittstellen zu Web-Services
* Schnittstellen in verteilten Systemen, wie netzwerkfähigen Spielen

Spätestens jetzt müssen Sie solche Schnittstellen genau definieren. Dazu gehören:

* Meldungstypen
* Syntax und Semantik der Meldungen (wie sind die Meldungen aufgebaut, welche Felder enthalten sie, welches sind mögliche Werte, welche Bedeutung haben die Meldung und die einzelnen Felder)
* Gültige Sequenzen von Meldungen

## Datenmodell

Wenn Sie in Ihrem System eine Datenbank haben, fügen Sie hier eine ERD ein. Beschreiben Sie die einzelnen Tabellen wo nötig kurz. Aus dem ERD müssen die Primary- und Foreign-Keys, sowie die Multiplizitäten hervorgehen

Wenn Sie objektorientiert arbeiten und ein Mapping-Framework (z.B. JPA/Hibernate oder ADO.Net Entity-Framework) verwenden, können Sie auch ein Klassendiagramm mit den Entitätsklassen einfügen. Auch hier beschreiben Sie die einzelnen Klassen, wo nötig, kurz.

# Systemdokumentation

Hermes 5 definiert unter anderem das Anwendungshandbuch, die Integrations- und Installationsanleitung, sowie das Betriebshandbuch als Resultate der Realisierungsphase. Hier werden diese Ergebnisse unter dem Oberbegriff Systemdokumentation zusammengefasst.



## Anwendungshandbuch

Das Anwendungshandbuch richtet sich an den Anwender oder Benutzer des Systems. Es „enthält alle Informationen, die der Anwender eines Produkts / Systems braucht, um es ordnungsgemäss zu bedienen und im Fall von Problemen richtig reagieren zu können“ (siehe HERMES 5 Handbuch Seite 127). Dazu gehören Übersicht, Funktionen, Detailbeschreibungen zur Anwendung und Fehlerbehandlung.

Denken Sie daran, dass die Anwender in der Regel keine IT-Fachpersonen sind. Entsprechend muss das Anwendungshandbuch in einer für das Zielpublikum verständlichen Sprache geschrieben sein.

### Übersicht

Hier folgt ein kurzer Überblick des Systems für die Anwender. Beschreiben Sie den Zweck und die Ziele des Systems.

### Funktionen und Detailbeschreibung

Beschreiben Sie hier die einzelnen Funktionen des Systems soweit, dass der Anwender diese erfolgreich anwenden kann. Stellen Sie sich also die Frage: „Was muss dem Anwender gesagt werden, damit er das System erfolgreich benutzen kann?“

Wenn Sie mit Anwendungsfällen gearbeitet haben, können Sie diese als Basis verwenden. Arbeiten Sie mit Screenshots. Denken Sie auch daran, dass es unter Umständen verschiedene Anwender gibt, wie registrierte Benutzer, Administratoren, etc.

### Fehlerbehandlung

Listen Sie hier mögliche Fehler, mit denen der Anwender konfrontierte werden kann auf und beschreiben Sie, was mögliche Gründe sein können und wie beim Auftraten dieser Fehler zu verfahren ist.

## Integrations- und Installationshandbuch

Fügen Sie hier eine Installationsanleitung für Ihr System ein. Je nach Art des Systems, muss eine unabhängige Fachperson oder gar der Anwender selber in der Lage sein, mit dieser Anleitung Ihr System erfolgreich zu installieren. Beschreiben Sie die einzelnen Schritte, allfällige Konfigurationsdaten und Tests zur Überprüfung, ob die Installation erfolgreich war.

## Betriebshandbuch

Wird der Betrieb des Systems nicht durch den Anwender selbst wahrgenommen, so braucht es ein Betriebshandbuch, welches „alle Informationen, die der Betreiber benötigt, um das

System ordnungsgemäss betreiben und im Fall von Problemen richtig reagieren zu können“ enthält (siehe HERMES 5 Handbuch Seite 128). Dazu gehören im Minimum Angaben über den Start / Neustart des Systems, Überwachung des Systems (z.B. Logs), Sicherung des Systems (z.B. Backup) sowie zu ergreifende Massnahmen bei Fehlern

# Systemtest

Im Modul 226 haben Sie die verschiedenen Teststufen und Vorgehensweisen zum Testen kennengelernt. Zur Repetition: Das Ziel beim Testen ist es, Fehler zu finden. Es geht nicht darum zu zeigen, dass das System keine Fehler hat.

Getestet wird auf verschiedenen Ebenen vom Einzeltest bis zum Abnahmetest (Akzeptanztest beim Kunden). Wir beschränken uns auf die beiden Ebenen Einzeltest (Unit-Test) und Systemtest/Abnahmetest.

Tests müssen reproduzierbar sein. Die Ausgangslage, der Ablauf und die erwarteten Resultate eines jeden Testfalls müssen detailliert beschrieben sein. Dies ist der Zweck der Testspezifikation.



## Unit-Test

Unit-Tests liegen in der Verantwortung des Entwicklers. Beschreiben Sie hier, wie und mit welchen Werkzeugen Sie die einzelnen Klassen oder Module Ihres Systems testen. Der Testcode ist im Anhang einzuschliessen.

## Systemtest

Systemtests liegen oft in der Verantwortung einer Testorganisation. In unserem Fall liegt die Verantwortung aber beim Projekt selber. Im Kleinprojekt macht es durchaus Sinn, die Testfälle des Systemtests später dann auch für den Abnahmetest in der Einführungsphase zu verwenden.

Anmerkung: Die folgenden Ausführungen gehen von einem von Hand durchgeführten Systemtest aus. Sollten Sie automatisierte Tests durchführen müssen Sie die Testspezifikation sinngemäss anpassen.

### Testfälle

Nachfolgend beschreiben Sie die einzelnen Testfälle für den Systemtest Ihres Systems. Als Ausgangspunkt nehmen Sie die Ziele aus der Voranalyse und die Anwendungsfälle aus Ihrem Konzept. Jeder Anwendungsfall führt jetzt zu mindestens einem separaten Testfall.

Das Ziel ist die vollständige Abdeckung der Funktionalität Ihres Systems durch die Testfälle. Es wird deshalb nötig sein auch weitere Testfälle zu beschreiben.

Spezifizieren Sie jeden Testfall separat mit einer Tabelle.

Beispiele:

#### *Testfall 1 „Neuen Benutzer registrieren und anmelden“*

|  |  |  |
| --- | --- | --- |
| ***Beschreibung*** | *Ein neuer Benutzer registriert sich und meldet sich beim System an.* | |
| ***Abgedeckte Anwendungsfälle*** | *Benutzerregistrierung* | |
| ***Ausgangssituation*** | *Es besteht noch kein solcher Benutzer.* | |
| ***Vorbereitungsschritte*** | 1. *Die Datenbank des Systems wird über PhpMyAdmin mit dem Script initTF1.sql in den Initialzustand versetzt.* 2. *Starten des Firefox auf der Testclient-Maschine.* | |
| ***Testschritte*** | | ***Erwartetes Resultat*** |
| 1. *Auf dem Testclient im Browser die URL* [*http://www.budget.ch*](http://www.budget.ch) *eingeben* | | *Die Startseite der Budgetplaner-Anwendung wird angezeigt.* |
| 1. *Den Link „Registrieren“ wählen.* | | *Das Registrierungsformular wird angezeigt.* |
| 1. *Eingeben der folgenden Informationen:*    1. *Benutzername:*  *Test1*    2. *Passwort:*  *testABC1*    3. *Passwortwiederholung:*  *testABC2* | |  |
| 1. *Den Button „Registrieren“ wählen* | | *Das Registrierungsformular wir erneut angezeigt. Die Felder für die Passworteingabe sind markiert.* |
| 1. *…* | | *…* |
| 1. *…* | | *In der Datenbank ist in der Tabelle Benutzer ein neuer Record mit dem Namen Test1 vorhanden. Überprüfung mit PhpMyAdmin.* |

#### *Testfall 2 „Bestehenden Benutzer registrieren“*

|  |  |  |
| --- | --- | --- |
| ***Beschreibung*** | *Ein Benutzer versucht sich mit einem bereits bestehenden Benutzernamen zu registrieren.* | |
| ***Abgedeckte Anwendungsfälle*** | *Benutzerregistrierung* | |
| ***Ausgangssituation*** | *Es besteht bereits ein Benutzer mit dem gleichen Namen.* | |
| ***Vorbereitungsschritte*** | 1. *Die Datenbank des Systems wird über PhpMyAdmin mit dem Script initTF2.sql in den Initialzustand versetzt.* 2. *Starten des Firefox auf der Testclient-Maschine.* | |
| ***Testschritte*** | | ***Erwartetes Resultat*** |
| 1. *Auf dem Testclient im Browser die URL* [*http://www.budget.ch*](http://www.budget.ch) *eingeben* | | *Die Startseite der Budgetplaner-Anwendung wird angezeigt.* |
| 1. *Den Link „Registrieren“ wählen.* | | *Das Registrierungsformular wird angezeigt.* |
| 1. *Eingeben der folgenden Informationen:*    1. *Benutzername:*  *Test1*    2. *Passwort:*  *testABC1*    3. *Passwortwiederholung:*  *testABC1* | |  |
| 1. *Den Button „Registrieren“ wählen* | | *Das Registrierungsformular wir erneut angezeigt. Das Feld für die Eingabe des Benutzernamens ist markiert.* |
| 1. *…* | |  |

# Testprotokoll

Das Testprotokoll gibt darüber Auskunft, wer, wann, was getestet hat und welches die Befunde des betreffenden Testdurchlaufes waren. Führen Sie die einzelnen Testdurchläufe in separaten Unterkapiteln auf.

Beispiel:

## *Systemtest 1*

*Getestete Version: 0.5, Iteration 1*

*Tester: Bill Geitsnoguet*

*Datum, Zeit: 13.5.2011, 14.00 – 15.30*

### *Testfall 1 „Neuen Benutzer registrieren und anmelden“*

|  |  |  |
| --- | --- | --- |
| ***Testschritt*** | ***Erfüllt*** | ***Bemerkung*** |
|  | 🗹 |  |
|  | 🗹 |  |
|  | 🗹 |  |
|  | 🗹 |  |
|  | 🗹 |  |
|  | 🗷 | *In der Tabelle Benutzer ist keine Record mit dem Namen Test1 eingetragen worden.* |

### *Testfall 2 „Bestehenden Benutzer registrieren“*

*Ganzer Testfall erfüllt.* (Tabelle mit Testschritten erübrigt sich.)

# Weiterführung der Projektplanung

## Abgleich von Planung und tatsächlichem Verlauf der Phase Konzept

Soll/Ist-Vergleich bezüglich der Zeitplanung für die Phase Realisierung.

Welche Risiken sind während der Phase Realisierung Realität geworden? Wie wurde diese Situation gemeistert? Welche Konsequenzen ergeben sich daraus für das gesamte Projekt?

## Aktualisierung der Risikosituation

Haben sich an der Risikosituation Änderungen (Verschärfung oder Entschärfung, neue Risiken) ergeben? Dann beschreiben Sie hier die Risikosituation neu. Führen Sie wiederum konkrete Massnahmen auf, wie Sie den Risiken in der nächsten Phase begegnen wollen.

## Planung der nächsten Phase

Aktualisieren und verfeinern Sie hier Ihren Projektplan, vor allem in Bezug auf die nächste Phase.

# Anhang A: Quellcode

Fügen Sie zuerst eine **Übersicht mit allen Dateien** des Codes ein. Zum Code gehören auch HTML, CSS, JS, JSP, SQL, etc.

Anschliessend folgt der Code.

**Unbedingt beachten**:

* Formatierung und Einrückung
* Neue Seite bei neuer Datei

# 

# Anhang B: Testcode Unit-Tests

Fügen Sie zuerst eine **Übersicht mit allen Dateien** des Testcodes ein.

Anschliessend folgt der Testcode.

Unbedingt beachten:

* Formatierung und Einrückung
* Neue Seite bei neuer Datei