



Fachhochschule Köln
Cologne University of Applied Sciences

PROJEKTDOKUMENTATION

„PKDEX“

Dokumentation im Rahmen des Moduls

„Webbasierte Anwendungen 2“

FH Köln - Campus Gummersbach

Betreuer: Prof. Dr. Kristian Fischer, David Bellingroth, Ngoc-Anh Dang

Sommersemester 2015

Ausgearbeitet von:

Ron Kersten

Leonid Vilents

Sascha Jan Kuhlmann

Gummersbach, den 06.09.2015

Inhaltsverzeichnis

	Inhaltsverzeichnis	II
1	Dokumentation des Dienstanbieters	3
1.1	<u>Exposé und Projektidee</u>	3
1.2	<u>Definierte Ressourcen</u>	3
1.3	<u>Anwendungslogik</u>	6
1.4	<u>Datenhaltung</u>	6
1.5	<u>Nicht implementierte Funktionalitäten</u>	7
1.6	<u>Use Cases</u>	8
1.6.1	<i>Pokémon-Ressource anfordern</i>	8
1.6.2	<i>pkTeam erstellen</i>	8
1.6.3	<i>pkTeam löschen</i>	9
2	Dokumentation des Dienstnutzers	10
2.1	<u>Anwendungslogik</u>	10
2.2	<u>Datenhaltung</u>	
2.3	<u>Präsentationslogik</u>	
2.4	<u>Nicht implementierte Funktionalitäten</u>	
2.5	<u>Use Case</u>	
2.5.1	<i>User herausfordern</i>	
3	Dokumentation des Arbeitsprozesses	
3.1	<u>Vorgehensweise und Irrwege</u>	
3.2	<u>Kritische Reflexion</u>	
3.3	<u>Arbeitsmatrix</u>	

1 Dokumentation des Dienstanbieters

1.1 Exposé und Projektidee

Im Folgenden soll das Exposé, dem die Projektidee und einige auszeichnende Elemente des Projekts zu Grunde liegen, vorgestellt werden.

Projekt: „pkDex“ - Eine interaktive Pokédex-Datenbank auf Basis des Pokémon-Universums von Nintendo

Nintendos Videospiel-Phänomen „Pokémon“ betört Kinderherzen seit 1999, als für den Gameboy Color die Editionen „Rot“ und „Blau“ in Europa auf den Markt gekommen sind. Mittlerweile wird das Franchise von Titeln im zweistelligen Bereich für Handheld- und stationäre Konsolen mehrerer Generationen vertreten, ebenso wie die jeweiligen Spielergenerationen. Das legendäre Prinzip, als Kind in eine große Welt auszuführen, und exotische Tiere – die Pokémon – zu fangen, zu sammeln, zu trainieren und gegen die Pokémon anderer in den Kampf zu schicken – ist nicht mehr aus der Welt der Videospielmedien wegzudenken. Gleichzeitig hat es kein anderes Derivat so weit gebracht.

Unsere Gruppe möchte diesem Phänomen Tribut zollen, und eine weitreichende Datenbank existierender Pokémon – einen Pokédex – erstellen. – Vollständige Kategorisierung existierender Pokémon in einer Datenbank – Sortierungsmöglichkeiten nach Typ, Name, Dex-Nummer, etc. – Filtereinstellungen für konkrete Suchoptionen – möglich: Erweiterte Informationen für die Spiel-Meta

Für die einzelnen Datensätze der Pokémon verwenden wir ein XML-Schema. Die Daten lassen sich über eine Webpage abrufen welche designtechnisch dem Originalgerät aus den Spielen entspricht.

Allerdings ist schon an dieser Stelle zu erwähnen dass einige Elemente der Idee bereits im Verlauf der Implementierung verworfen wurden. Dies findet in den entsprechenden Positionen der Dokumentation genauer Erwähnung.

1.2 Definierte Ressourcen

Im Folgenden sollen die definierten Ressourcen des implementierten Dienstnutzers beschrieben, die Semantik der entsprechend anwendbaren http-Verben aufgezeigt und Überlegungen beziehungsweise verworfene Implementierungen und Ideen aufgezeigt werden.

Ressource	GET	PUT	POST	DELETE
/pkDex (Listenressource die die einzelnen Pokémon enthält)	X			
/pkUser (Listenressource die die einzelnen User enthält)	X	X	X	X
/pkTeam (Listenressource die die einzelnen Teams enthält)	X	X	X	

Tabelle 1: Ressourcen und die darauf anwendbaren http-Verben

Die endgültige Definition der Ressourcen und der entsprechenden http-Verben ergab sich iterativ im Laufe der Arbeit an dem Projekt. Entsprechend TABELLE 1 ergeben sich also folgende Ressourcen und entsprechende http-Verben die zur Verfügung stehen und mit denen gearbeitet werden kann:

Ressource	Methode	Semantik	Content-type (req)	Content-type (res)
/pkDex	GET	Anfordern der kompletten Listenressource '/pkDex' die sämtliche Pokémon enthält	application/json	application/json
/pkDex/:prm	GET	Anfordern einer Unterressource von '/pkDex' anhand einer Identifikation ':prm' sprich die Ausgabe eines einzelnen Pokémon	application/json	application/json
/pkTeam	POST	Ein eigenes Team mit vorhergehender Überprüfung auf bereits Vorhandensein der gewählten Identifikation des Teams 'sign' erstellen	application/json	application/json
/pkTeam/:sign	PUT	Eine Unterressource von '/pkTeam' verändern, sprich ein einzelnes Team hinsichtlich seiner Attribute bearbeiten	application/json	application/json
/pkTeam	GET	Anfordern der kompletten Listenressource '/pkTeam' die sämtliche Teams enthält	application/json	application/json

/pkTeam/:sign	GET	Anfordern einer Unterressource von '/pkTeam' anhand einer Identifikation ':sign' also die Ausgabe eines einzelnen Teams und seiner Member	application/json	application/json
/pkUser	POST	Eine neue Unterressource an die Listenressource '/pkUser' anhängen, sprich einen neuen User anlegen	application/json	application/json
/pkUser	DELETE	Löschen einer Unterressource von '/pkUser' anhand einer Identifikation ':sign'	application/json	text/plain
/pkUser/:prm	PUT	Eine Unterressource von '/pkUser' verändern, sprich einen einzelnen User hinsichtlich seiner Attribute bearbeiten	application/json	application/json
/pkUser	GET	Anfordern der kompletten Listenressource '/pkUser' die sämtliche User enthält	application/json	application/json
/pkUser/:prm	GET	Anfordern einer Unterressource von '/pkDex' anhand einer Identifikation ':prm' also die Ausgabe eines einzelnen Pokémon	application/json	application/json

Tabelle 2: Semantik der http-Verben

1.3 Anwendungslogik

Im Folgenden soll die Anwendungslogik des implementierten Webservice beschrieben und Überlegungen zu dieser aufgezeigt werden.

- **Datenhaltung und Bereitstellung der benötigten Ressourcen**
Der Webservice kümmert sich um die persistente Datenhaltung der erstellten Ressourcen und stellt diese auch entsprechend der Anfragen des Dienstnutzers zur Verfügung. Die Datenhaltung wird in einer folgenden Position genauer erklärt.
- **Regulierung des Kampfsystems und der Statistik**
Das Kampfsystem, welches im wesentlichen die Anwendungslogik des Dienstnutzers darstellt, besteht aus einem Zähler der als Event-Handler Klicks des Users zählt und diesen Score mit dem eines weiteren Users vergleicht. Diese Funktion und eine Erstellung einer Rangliste anhand von Highscores übernimmt der Webservice.

Überlegungen die die Anwendungslogik betreffend angestellt wurden waren zum Beispiel die genaue Implementierung der entsprechenden Funktionalitäten des Webservice. Diese haben sich im Verlauf des Projekts zum Teil iterativ verändert.

1.4 Datenhaltung

Im Folgenden soll die Datenhaltung des implementierten Webservice beschrieben und Überlegungen zu dieser aufgezeigt werden.

Um eine persistente Datenhaltung zu ermöglichen wurden drei separate Dateien auf dem eigenen Rechnersystem herangezogen. Diese Dateien können im Verlauf der Anwendung mit den http-Verben POST, PUT und DELETE bearbeitet und mittels dem http-Verb GET angefordert werden. Die hierzu implementierten Funktionen sind writeFile() und readContent().

- **'/pkDexGen1.json'**
Diese Datei beschreibt die Listeressource von Pokémon-Objekten die für die Bildung eines Teams herangezogen werden können. Folgende Syntax beschreibt ein solches Objekt:
`{"index":"001","name":"Bisasam","typ1":"Pflanze","typ2":"Gift","bes":"Zwiebelkröte"}`
- **'/pkTeam.json'**
Diese Datei beschreibt die Listenressource der von den einzelnen Usern zusammengestellten Teams. Folgende Syntax beschreibt ein solches Team-Objekt:
`{"index":"1","team":[{"mem1":"001"}, {"mem2":"004"}, {"mem3":"007"}, {"mem4":"010"}, {"mem5":"013"}, {"mem6":"016"}]}`
- **'/pkUser.json'**
Diese Datei beschreibt die Listenressource von den User-Objekten. Folgende Syntax beschreibt ein solches Objekt: `{"user":[{"Nik":"123"}, {"name":"Hans"}, {"atr":"none"}, {"hscore":""}, {"lscore":""}]}`

Eine Überlegung die angestellt wurde war die Verwendung eines einfachen Datenbanksystems wie zum Beispiel „Redis“. Diese Alternative fand keine Verwendung weil für diesen einfachen Webservice eine Datenbankstruktur überflüssig ist. Die Größe und Komplexität der verwendeten Daten lässt auch die Benutzung von einfachen Dateien zur Speicherung zu.

Ziel war es das Verwalten der Daten möglichst einfach und mit wenig Aufwand zu gestalten und hierfür wurde eben die Möglichkeit der separaten json-Dateien gewählt.

1.5 Nicht implementierte Funktionalitäten

Im Folgenden sollen eventuell nicht implementierte Funktionalitäten des implementierten Webservice beschrieben und hierzu aufgezeigt werden.

Funktionalitäten die auf Grund von zum Beispiel Zeitmangel nicht umgesetzt werden konnten gibt es in dem Sinne nicht, da sich der Service auf die wesentlichen Funktionen beschränkt. Die Funktionen wurden im Laufe des Arbeitsprozesses lediglich immer wieder umgeschrieben und hinsichtlich der Logik verbessert.

Da sich die Funktionalitäten (Ressourcen und Funktionen) iterativ entwickelt wurden gab es natürlich auch einige Überlegungen die hinsichtlich der Logik und Sinnhaftigkeit verworfen wurden. Einige Beispiele sind im Folgenden aufgeführt:

Ressource	Methode	Semantik	Content-type (req)	Content-type (res)
/pkDex/pkm/:prm	GET	Anfordern einer Unterressource von '/pkDex'	application/json	application/json
/pkDex?query	GET	Anfordern einer Unterressource von '/pkDex'	application/json	application/json
/pkDex/pkm	POST	Erstellen einer neuen Unterressource	application/json	application/json

Tabelle 3: *Verworfenne Ressourcen und Funktionalitäten*

Die Funktionalitäten wurden allerdings verworfen da Unklarheiten bei der Benennung der Ressourcen aufgetreten sind und diese nicht dem REST-Prinzip entsprachen. Dies war auch gleichzeitig die größte Herausforderung bei der Definition der Ressourcen.

Auch wurden einige Ansätze verworfen weil sie im Zuge der Erstellung des Projekts als nichtig erschienen sind. So ist zum Beispiel das manuelle Erstellen von pkDex-Unterressourcen nicht notwendig, da diese Informationen vom System bereitgestellt werden.

Allerdings hätten natürlich einzelne Funktionen noch ausgeweitet werden können. Dies wurde aber auf Grund der Zielsetzung das Projekt so minimal wie möglich zu halten nicht berücksichtigt.

1.6 Use Cases

Im Folgenden sollen für die Anwendungslogik des implementierten Webservice beschriebene Use Cases aufgeführt werden. Nicht alle hier eventuell auftauchenden Funktionalitäten wurden auch umgesetzt.

Die Use-Cases wurden im Verlauf des Projekts erstellt und sollen einen Eindruck davon vermitteln wie mit dem System gearbeitet werden kann beziehungsweise wie das System einzelne Anfragen bearbeitet.

1.6.1 *Pokémon-Ressource anfordern*

use case Pokémon-Ressource anfordern

actors Client

precondition Der Client muss mit seinem Userkonto auf dem Server angemeldet sein

main flow m1

1. Der Client fordert eine Unterressource 'pkDex/:prm' mit einem beliebigen ':prm' mittels GET von der Listenressource '/pkDex' an
2. Der Server liefert die angeforderte Unterressource und den Statuscode '200 OK'

postcondition Der User hat erfolgreich ein Pokémon angefordert

exceptional flow a1

- 1a. Der Client fordert eine Unterressource 'pkDex/:prm' mit einem beliebigen ':prm' mittels GET von der Listenressource '/pkDex' an
- 2a. Unterressource wird vom Server nicht gefunden und dieser liefert den Statuscode '404 NOT FOUND'

postcondition Die Suche war nicht erfolgreich.

end Pokémon-Ressource anfordern

1.6.2 *pkTeam erstellen*

use case pkTeam erstellen

actors Client

precondition Der Client muss mit seinem Userkonto auf dem Server angemeldet sein

main flow m1

1. Der Client sendet einen Request mit einem POST an den Server auf die Listenressource '/pkTeam' mit entsprechendem Inhalt
2. Der Server prüft auf Vorhandensein der Signatur des Teams; die Überprüfung ist negativ
3. Das Team wird erfolgreich gesetzt beziehungsweise an '/pkTeam angehängen' und der Server sendet den Plaintext 'PkTeam erfolgreich gesetzt.' als Bestätigung

postcondition pkTeam erfolgreich erstellt

exceptional flow a1

1a. Der Client sendet einen Request mit einem POST an den Server auf die Listenressource '/pkTeam' mit entsprechendem Inhalt

2a. Der Server prüft auf Vorhandensein der Signatur des Teams; die Überprüfung ist positiv

3a. Das Team wird nicht gesetzt und der Server sendet den Plaintext 'Der Trainer: [newsign] besitzt schon ein pkTeam' sowie den Statuscode '500 Internal Server Error ' als Bestätigung

postcondition pkTeam nicht erstellt

end pkTeam erstellen

1.6.3 *pkTeam löschen*

Dieser Use-Case wurde verworfen und durch ein Bearbeiten der Unterressource mittels einem PUT ersetzt. Erwähnung findet er dennoch um den Arbeitsprozess nachvollziehen zu können.

use case pkTeam löschen

actors Client

precondition Der Client muss mit seinem Userkonto auf dem Server angemeldet sein und es muss mindestens ein pkTeam im Userkonto vorhanden sein

main flow m1

1. Der Client fordert eine Seite vom Server an welche alle Teams seines Userkontos auflistet

2. Es wird die Datei pkDex mit GET('/pkDex') vom Server gesendet. Es wird der bei einem erfolgreichen Laden, der Statuscode '200 OK' zurück gegeben

3. Der Client sucht das zu löschende Team aus und sendet ein DELETE('/pkTeam') an den Server. Ist der DELETE erfolgreich, sendet der Server den Statuscode '200 OK'.

postcondition pkTeam wurde erfolgreich gelöscht

exceptional flow a1

1a. Schlägt GET('/pkDex') fehl wird der Statuscode '404 ressource not found' an den Client gesendet

exceptional flow a2

2a.. Schlägt GET('/pkTeam') fehl wird der Statuscode '404 ressource not found' an den Client gesendet

exceptional flow a3

3a. Schlägt DELETE('/pkTeam') fehl wird der Statuscode '500 internal Server Error' an den Client gesendet

postcondition Löschen war nicht erfolgreich

end pkTeam löschen

2 Dokumentation des Dienstnutzers

2.1 Anwendungslogik

Im Folgenden soll die Anwendungslogik des implementierten Dienstnutzers beschrieben und Überlegungen zu dieser aufgezeigt werden.

konkrete Verknüpfung der einzelnen Elemente einer Anwendung als Aufrufreihenfolge, Definition der Paramatern und Bearbeitung der anwendungsrelevanten Daten; sprich: hat Verarbeitungsfunktion auf dem Server; „Kern des Geschäfts“ sprich die Elemente mit denen wirklich umgegangen wird (was wird gemacht, was darf gemacht werden, etc.)

Die Logik des Client besteht darin sich mit anderen Nutzern des Service unter Verwendung der selbsterstellten pkTeams zu messen, sprich einen „Kampf“ auszutragen. Der Kampf selber ist ein Klick-Event bei dem die Anzahl der Klicks entscheidet. Die erstellten pkTeams tragen insofern zu dem Ausgang bei als dass sie einen Multiplikator bilden, der mit den gezählten Klicks verrechnet wird. Rekorde werden abgespeichert.

2.2 Datenhaltung

Im Folgenden soll die Datenhaltung des implementierten Dienstnutzers beschrieben und Überlegungen zu dieser aufgezeigt werden.

2.3 Präsentationslogik

Im Folgenden soll die Präsentationslogik des implementierten Dienstnutzers beschrieben und Überlegungen zu dieser aufgezeigt werden.

Art der Realisation der Benutzerschnittstelle auf dem Client; Programmteil der bestimmt welche Interaktionsmöglichkeiten dem Benutzer zur Verfügung stehen; beschreibt das Aussehen beziehungsweise die „Präsentation“

2.4 Nicht implementierte Funktionalitäten

Im Folgenden sollen eventuell nicht implementierte Funktionalitäten des implementierten Dienstnutzers beschrieben und hierzu aufgezeigt werden.

2.5 Use Case

Im Folgenden soll ein für die Anwendungslogik des implementierten Clients beschriebener Use Cases aufgeführt werden.

2.5.1 *User herausfordern*

use case User herausfordern	
	actors Client
es	precondition Der Client muss mit seinem Userkonto auf dem Server angemeldet sein und muss sich mindestens ein PkTeam in beiden Userkonten befinden und es muss mindestens ein Pkmn in beiden PkTeams vorhanden sein.
	main flow m1
	1.Der Client ruft die Profilseite eines anderen Users auf.
	2.Es wird das Profil des Users ausgegeben.
	3.Der Client wählt die Option „Herausfordern“ aus.
	4.Der Server prüft die Conditions und fragt die Auswahl eines eigenen Pkmn aus einem

Team ab.

Zeit 5. Der Client beginnt den „Kampf“: Der User muss einen Button innerhalb einer gegebenen
möglichst oft klicken. Der herausgeforderte User „verteidigt“ mit seinem letzten Rekord.

6. Nach Ablauf der Zeit vergleicht der Server die Ergebnisse, der User mit den meisten
Klicks gewinnt. Geschlagene eigene Rekorde werden überschrieben.

7. Der User erhält die Möglichkeit, einen weiteren Kampf zu führen oder zu beenden.

postcondition Kampfstatistik beider User aktualisiert, eventuell wurden Rekorde
überschrieben.

alternate flow a1

1. Der User verfügt über kein PkTeam: Die Möglichkeit, einen Kampf zu beginnen, ist nicht
gegeben.

2. Der Gegner verfügt über kein PkTeam: Die Möglichkeit, diesen Gegner herauszufordern,
ist nicht gegeben.

postcondition „Kampf“ kann nicht stattfinden

end User herausfordern

3 Dokumentation des Arbeitsprozesses

3.1 Vorgehensweise und Irrwege

Im Folgenden soll der Arbeitsprozess im Verlauf des Projekts beschrieben und eventuell aufgetretene Schwierigkeiten aufgezeigt werden.

Das Vorgehen orientierte sich weitgehend an den Meilensteinen zum Praktikum des Workshops. Passend zu den einzelnen Terminen wurden auch die Funktionalitäten ergänzt und erweitert. Da diese Termine aber nur die Pflicht darstellten reichte dieses Pensum natürlich nicht aus um ein zufriedenstellendes Ergebnis zu erzielen. Viel wurde auch außerhalb dieser Termine erarbeitet.

Der Prozess zur Erstellung und Bearbeitung des Projekts kann grob in folgende Phasen unterteilt werden (zeitlich in dieser Reihenfolge):

- **Ideenfindung**

Um ein Projekt realisieren zu können gilt es erst einmal eine passende Idee zu generieren. Vorgabe war ein System nach den Prinzipien von „Representational State Transfer“ REST zu entwickeln. Die schließlich umgesetzte Idee mit dem Pokédex und einem damit einhergehenden Kampfsystem stand schließlich recht zügig fest.

- **Dienstanbieter und Dienstnutzer**

Nach der Idee folgt nun die Implementierung. Es galt nun zunächst die erforderlichen Ressourcen für das System zu definieren, schließlich zu implementieren und nutzbar zu machen. Dies wurde zum Beispiel mittels Use-Cases und der Erstellung von Tabellen zur Übersichtlichkeit unternommen. Anschließend wurde der Dienstnutzer realisiert.

- **REST-Konformität**

Über allem stand ständig die Frage nach den REST-Prinzipien als Vorgabe. Diese galt es als Orientierung zu nutzen und zu berücksichtigen.

Irrwege wurden auch beschritten, allerdings ist eher die Beschreibung „fehlende Weitsicht“ zutreffend. Im Optimalfall werden ein Ablauf und Aufgaben definiert die in einer bestimmten Reihenfolge abgearbeitet werden. Dies war auch mittels der Meilensteine des Workshops geschehen.

Eine Schwierigkeit bestand in der Tatsache dass es sich bei REST um ein für uns zunächst unbekanntes Prinzip handelte, dementsprechend wurden auch Fehler gemacht. Allerdings nichts was nicht zu lösen war.

Weitere Schwierigkeiten traten an manchen Stellen bei der Implementierung von einzelnen Funktionalitäten auf. In einem iterativen Prozess wurde als das System ständig überarbeitet.

Ein weiterer wichtiger Punkt bestand auch in der Kommunikation und Arbeitsteilung. Um ein solches Projekt erfolgreich abschließen zu können sind beides wichtige Gesichtspunkte. In unserem Fall hat dies zufriedenstellenden funktioniert.

Der größte „Irrweg“ bestand also in der Tatsache sich mit teils neuen und komplexen Thematiken wie REST oder ejbs zu beschäftigen und diese konkret anzuwenden.

3.2 Kritische Reflexion

Im Folgenden soll das Projekt in seiner Gesamtheit und dem eventuell Nicht-Erreichten und Erreichten kritisch reflektiert und mit einem Fazit beschlossen werden.

Um das Fazit vorne weg zunehmen ist zu sagen dass das Projekt in seiner Gesamtheit und was die von uns gesteckten Ziele angeht erfolgreich abgeschlossen wurde.

Die für die Erstellung des Projekts angesetzte Zeit war ausreichend und auch der Arbeitsaufwand war angemessen. Rückblickend hätte ein besser organisiertes Zeitmanagement unsererseits wohl ein effizienteres Arbeiten ermöglicht.

Auch die Tatsache dass oft an vielen „Baustellen“ und Bereichen des Projekts gleichzeitig gearbeitet wurde hätte wohl eleganter gelöst werden können, indem zum Beispiel eine stringente Arbeitsabfolge erstellt wird.

Es gibt also wohl noch Kleinigkeiten unter dem Gesichtspunkt „Projektmanagement“ zu verbessern. Dies ist aber wahrscheinlich der Tatsache geschuldet das dieses Projekt eines der ersten größeren Softwareprojekte war und einfach noch die Praxis fehlt.

„Erreicht“ wurde also ein Einblick in die Softwareerstellung selber, der damit einhergehenden Teamarbeit und der notwendigen Organisation sowie ein, an unseren Zielen gemessen, zufriedenstellendes Endergebnis.

Unter dem Gesichtspunkt „Nicht-Erreicht“ könnten nicht implementierte Funktionalitäten aufgeführt werden. Diese gibt es im eigentlichen Sinne aber nicht da unsere Ziele und Implementierung iterativ überarbeitet wurden.

3.3 Arbeitsmatrix

Die Matrix soll als Beschreibung dienen um aufzeigen zu können welches Gruppenmitglied zu welchen Anteil in Prozent (%) an welcher Aktivität beteiligt war.

Aktivität	Ron Kersten	Leonid Vilents	Sascha Jan Kuhlmann
Projektidee			
Implementierung Webservice			
Implementierung Client			
Projektdokumentation			