



Abschlussprüfung Sommer 2021

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Veltins Dashboard

Web-Anwendung für Veltins Gewinnspiel-Statistiken

Prüfungsbewerber:

Ömer S. ACAR
Laaken 85
42287 Wuppertal

Umschulungsträger:



IT-Akademie Dr. Heuer GmbH
Konrad-Zuse-Straße 2b
44801 Bochum

Ausbildungsbetrieb:



Schleier-IT
Michaelstr. 24A
45138 Essen

Inhaltsverzeichnis

Tabellenverzeichnis	ii
Abbildungsverzeichnis	ii
Abkürzungsverzeichnis	ii
1 Einleitung	1
1.1 Projektbeschreibung	1
1.2 Projektziel	1
1.3 Projektbegründung	2
1.4 Projektschnittstellen	2
1.5 Projektumfeld	2
1.5.1 Ausbildungsbetrieb	2
1.5.2 Kunde	2
2 Planung	3
2.1 Projektphasen	3
2.2 Ressourcenplanung	3
2.3 Entwicklungsprozess	4
2.4 Ist-Analyse	4
2.5 Soll-Konzept	4
2.6 Pflichtenheft	5
2.7 Wirtschaftlichkeitsanalyse	5
2.7.1 „Make or Buy“-Entscheidung	5
2.7.2 Projektkosten	6
2.7.3 Amortisationsdauer	6
2.8 Programmablaufplan	7
2.9 Entwicklungsumgebung und die Bibliotheken	7
3 Implementation	8
3.1 Benutzeroberflächen	9
3.2 Datenbankkommunikation	11
3.3 Geschäftslogik	12
4 Test/Qualitätssicherung	13
4.1 Genauigkeit	13
4.2 Benutzerfreundlichkeit & Verständlichkeit	13
4.3 Ausnahmebehandlung	14
5 Dokumentation	14
6 Fazit	14
6.1 Ist-Soll Vergleich	14
6.2 Gewonnene Erkenntnisse	15
Literaturverzeichnis	a

Anhang	b
A.1 Detaillierte Zeitplanung	b
A.2 Verwendete Ressourcen	c
A.3 Ausschnitt des Kanban-Boards	c
A.4 Auszug aus dem Pflichtenheft	d
A.5 Beispiel Screenshots der Programmablaufplan	e
A.6 Ein Screenshot der Benutzeroberfläche	f

Tabellenverzeichnis

Tabelle 1 - Grobe Zeitplanung	3
Tabelle 2 - Berechnung der Projektkosten	6
Tabelle 3 - Soll/Ist Vergleich	14

Abbildungsverzeichnis

Abbildung 1 - Aktionsauswahl	9
Abbildung 2 - Gesamtzahlen einer Aktion	9
Abbildung 3 - Zeitfilterwerkzeugen	9
Abbildung 4 - Aktivitätstabelle	10
Abbildung 5 – Geschlechter - Alter Verhältnis, Herkunft-Tabelle	10
Abbildung 6 - Aktionsvergleich	11

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVC	Model-View-Control
NPM	Node Package Management
PHP	Hypertext Preprocessor
XML	Extensible Markup Language

1 Einleitung

Die folgende Projektdokumentation schildert und erläutert den Verlauf des IHK-Abschlussprojektes, welches der Autor im Rahmen seiner Ausbildung zum Fachinformatiker in der Fachrichtung Anwendungsentwicklung durchgeführt hat. Absicht dieser Projektdokumentation ist die Erläuterung des Prozesses von der Planung bis hin zum Einsatz der Webanwendung.

1.1 Projektbeschreibung

Das Unternehmen Veltins organisiert verschiedene Gewinnspiele für Werbezwecke, um seine Produktivität zu erhöhen. Die Bewertung und Interpretation dieser Gewinnspiele ist wichtig für die Verkaufs- und Werbekampagnenrichtlinien des Unternehmens. Durch Auswertung der Gewinnspiele kann der Kunden ein Wachstum der Teilnehmer Jahr für Jahr auswerten, Marketing-Aktionen wie Facebook-Postings, Online-Werbekampagnen oder Push-Benachrichtigungen direkt mit der Zahl und der Eigenschaften neuer Teilnehmer in Gewinnspielen korrelieren lassen und so die Aktionen bewerten.

In diesem Projekt geht es darum ein webbasiertes Dashboard zu entwickeln, welche dafür genutzt wird, alle Statistiken und Berechnungen eines Gewinnspiels über verschiedene Funktions- und Kategorieoptionen zu erstellen und mit Grafiken zu präsentieren. Die hierfür erforderlichen Daten werden aus der vorhandenen Gewinnspiel-Datenbank entnommen.

Da es sich um einen direkten Kundenauftrag handelt, wird die genaue Wirtschaftlichkeit des Projekts in der Dokumentation dargestellt. Die Planung des Projekts wird so angelegt, dass die Durchführung im Rahmen des vereinbarten Budgets bleibt und für die Schleier-IT ein positives wirtschaftliches Ergebnis liefert.

1.2 Projektziel

Ziel des Projekts ist die Entwicklung einer Webanwendung mit grafischer Benutzeroberfläche, Datenbankverbindung und Geschäftslogik. Auf diese Weise wird die Erstellung von Statistiken und Berechnungen für diese Wettbewerbe automatisiert, genauer, effizienter und aktueller bewertet und die für diesen Prozess erforderlichen Kosten reduziert.

1.3 Projektbegründung

Die manuelle Durchführung dieser Auswertungen ist sehr Zeit- und Arbeitsaufwendig. Darüber hinaus sind die Bewertungen aufgrund der ständigen Änderung der vorhandenen Gewinnspielsdaten schnell veraltet. Daher benötigte der Kunde dieses Dashboard, in dem diese Statistiken erstellt, wie gewünscht dargestellt und gleichzeitig aktualisiert werden.

1.4 Projektschnittstellen

Damit diese Anwendung funktioniert, ist Zugriff auf eine MySQL-Datenbank erforderlich. Diese Datenbank ist auf dem Server verfügbar, auf dem die Anwendung veröffentlicht wird. Abgesehen davon ist das Projekt unabhängig von allen anderen technischen Schnittstellen.

1.5 Projektumfeld

Die Projektumfeld wird in zwei Teilen als Ausbildungsbetrieb und Kunde bewertet.

1.5.1 Ausbildungsbetrieb

Der Ausbildungsbetrieb ist Schleier-IT, ein IT-Dienstleister, der von Kunden aus verschiedenen Branchen in Auftrag gegeben werden und seinen Kunden mit seiner 15 Mitarbeitern seit 2014 Webdesign, Software- und mobile App-Entwicklung, Webhosting und Domains anbietet. Das Projekt wird für einen Kunden von Schleier-IT eingesetzt werden und bei Schleier IT durchgeführt.

1.5.2 Kunde

Veltins GmbH & Co. KG ist eine im Jahre 1824 gegründete Brauerei und hat mehr als 600 Mitarbeitern. Der Sitz befindet sich in Meschede-Grevenstein im Sauerland. Schleier-IT betreut das Unternehmen Veltins in verschiedenen Bereichen, speziell die Veltins App. Die Gewinnspiele werden derzeit innerhalb der App abgewickelt. Schleier-IT wird den Serverdienst dieser Anwendung bereitstellen.

2 Planung

In diesem Abschnitt geht es darum, wie das weitere Vorgehen bestimmt wurde. Das Projekt wurde dazu zeitlich geplant, alle benötigten Ressourcen ermittelt, der Entwicklungs- und Testprozesse bestimmt.

2.1 Projektphasen

Der Autor stand ein Durchführungszeitraum von 70 Stunden zur Verfügung. Eine grobe Zeitplanung findet sich in Tabelle 1 - Grobe Zeitplanung. Eine detailliertere Übersicht mit den einzelnen Teilaufgaben kann im Anhang A.1 Detaillierte Zeitplanung eingesehen werden. Die Projektphasen basieren auf dem Wasserfallmodell, das ein lineares Vorgehensmodell ist und das in aufeinander folgenden Projektphasen organisiert ist. Der Grund für die Wahl des Wasserfallmodells besteht darin, sicherzustellen, dass der Arbeitsumfang und die Kosten zu Beginn des Projekts gut geschätzt werden können.

Projektphase	Geplante Zeit
A - Analyse & Planung	19h
B - Implementation	32h
C - Test/Qualitätssicherung	4h
D - Dokumentation	13h
E - Nachbearbeitung	2h
Gesamt	70h

Tabelle 1 - Grobe Zeitplanung

2.2 Ressourcenplanung

In der Übersicht, die sich im Anhang A.2 Verwendete Ressourcen befindet, sind jene Ressourcen aufgelistet, welche für dieses Projekt eingesetzt wurden. Hier wurde neben den Hard- und Softwareressourcen auch das Personal mit aufgenommen. Bei der Auswahl der Software wurde darauf geachtet, dass diese kostenfrei (z.B. Open source) zu der Verfügung steht oder Schleier-IT bereits über die passenden Lizenzen verfügt. Die Projektkosten sollen dadurch möglichst gering gehalten werden.

2.3 Entwicklungsprozess

Um eine flexible Umsetzung der Anforderungen zu ermöglichen, hat sich der Autor für einen agilen Entwicklungsprozess entschieden. Da es sich um ein vergleichsweise kleines Projekt mit wenigen Projektteilnehmern handelt, wurde die agile Methode Kanban verwendet. Die Methode Kanban ermöglicht die flexible Änderung der Anforderungen und einen fließenden Entwicklungsprozess, ohne dabei viele Einschränkungen zu definieren. Die Umsetzung erfolgt mit Hilfe eines Kanban-Boards, das in der Projektmanagement-Software "ClickUp" verwaltet wird. Die Projektphasen werden hier als Aufgaben hochgeladen, um den Entwicklungsprozess zu verfolgen. Ein Ausschnitt dieses Boards befindet sich im Anhang A.3 Ausschnitt des Kanban-Boards.

Die Anwendung wird zunächst mit Hilfe der Visual Studio als IDE und der Ampps-Anwendung, die den Apache-Server und die MySQL-Datenbank bereitstellt, auf dem lokalen Computer entwickelt und dann auf den Schleier-IT-Server übertragen. Bei der Entwicklung der Anwendung werden Beispieldatenbanken, Tabellen und Daten verwendet, die gemäß der tatsächlichen Datenstruktur erstellt werden.

2.4 Ist-Analyse

Die Auswertungen werden derzeit manuell für mehrere Tabellenkalkulationsdateien durchgeführt. Alle Teilnehmerdaten werden von einem Mitarbeiter von Schleier-IT mithilfe mehrerer SQL-Abfragen manuell aus verschiedenen Datenbanken als Tabellenkalkulationsdatei exportiert. Nach dem Export werden die Berechnungen dank der Tabellenkalkulationsfunktionen durchgeführt. Die berechneten Werte werden in eine Matrix in dieser Tabellenkalkulationsdatei eingegeben und mit manuell erstellten Grafiken angezeigt und ausgewertet. Dieser Prozess ist sehr zeit- und arbeitsintensiv und kostet den Kunden. Darüber hinaus ist das Tabellenkalkulationsformat für diese Prozesse sehr verwirrend und es müssen täglich neue Tabellen erstellt werden, da sich die Situation ständig ändert.

2.5 Soll-Konzept

Für dieses Projekt wird eine Webanwendung erstellt. Die Daten werden direkt aus der vorhandenen Veltins Gewinnspiel-Datenbank abgerufen, die bereits von Schleier-IT administriert wird. Schleier-IT wird den Serverdienst dieser Anwendung bereitstellen. Der

Zugriff auf die Anwendung erfolgt über ein Anmeldesystem durch Eingabe eines Benutzernamens und eines Kennworts. Statistiken, Auswertungen und Berechnungen variieren dynamisch mit Optionen und Eingabewerkzeugen, die dem Benutzer auf der Benutzeroberfläche zur Verfügung stehen.

Entsprechend den Kundenanforderungen sollte das Dashboard auch in der Lage sein, zwischen den Aktionen zu wechseln, um bessere Statistiken über die Teilnehmer während und nach der Durchführung von Gewinnspielen über Zeitoptionen wie heute, gestern, aktuelle Woche, letzte Woche oder ein beliebiges Zeitintervall zu erhalten. die Teilnehmer nach Geschlecht, Altersgruppe, Region und Zeitpunkt der Teilnahme zu bewerten und die verschiedenen Gewinnspiele mit den Vorjahren vergleichen zu können.

Darüber hinaus kann es zwischen Bundesländern wechseln und eine detaillierte regionale Bewertung der Teilnahme an einer Aktion anzeigen. Die aktuellsten Statistiken können jedes Mal abgerufen werden, wenn die Anwendung geöffnet, die Seite aktualisiert oder eine Funktion verwendet wird.

2.6 Pflichtenheft

Das Pflichtenheft legt die in der Entwicklung verfolgten Ziele des Produktes fest. Es wird verwendet, um den Entscheidungsraum für die Realisierung abzustecken. Diese Abgrenzung erfolgt durch das Festlegen von Wunsch-, Muss- und Absteckungskriterien. Ein Auszug aus dem für dieses Projekt erstellten Pflichtenheft befindet sich im Anhang A.4 Auszug aus dem Pflichtenheft.

2.7 Wirtschaftlichkeitsanalyse

Aufgrund des geschilderten Sachverhalts, der in Abschnitt 1.4 Projektbegründung und in Abschnitt 2.4 Ist-Analyse beschrieben wurde, ist die Umsetzung des Projekts erforderlich. Ob die Realisierung und die damit verbundenen wirtschaftlichen Aufwendungen gerechtfertigt sind, soll in den folgenden Abschnitten betrachtet werden.

2.7.1 „Make or Buy“-Entscheidung

Dieses Dashboard ist ein unternehmensspezifisches Produkt der Veltins. Bei der Recherche und Betrachtung von Software mit vergleichbarer Funktionalität konnte festgestellt werden,

dass keine der vorhandenen Anwendungen am Markt die Anforderungen erfüllt und im Verhältnis zu den im nächsten Abschnitt kalkulierten Projektkosten steht. Daher soll das Projekt in Eigenentwicklung durchgeführt werden.

2.7.2 Projektkosten

Im Folgenden sind die berechneten Kosten aufgeführt, die während der Entwicklung des Projekts entstanden sind. Dabei werden sowohl die Personal-, als auch sonstige Ressourcenkosten berücksichtigt. Als Ressourcenkosten wurde vom Management eine Pauschale von 10,00 € pro Stunde festgelegt, die sich aus mehreren Komponenten zusammensetzt. Diese Komponenten umfassen Stromkosten, Büromietkosten, Anschaffungskosten für Hardware und Software, Wartungs- und Lizenzkosten für den Server sowie Gemeinkosten. Da die exakten Personalkosten aus Datenschutzgründen nicht herausgegeben werden dürfen, wurde die Kalkulation anhand von beispielhaften Kosten durchgeführt. Ein beispielhafte Stundenpauschale eines Mitarbeiters liegt bei 50,00 €, die eines Auszubildenden 10,00 €. Sämtliche anfallende Projektkosten, können der folgenden Tabelle entnommen werden.

Vorgang	Mitarbeiter	Zeit	Personalkosten	Ressourcenkosten	Gesamtkosten
Entwicklung	Auszubildender	70h	700,00 €	700,00 €	1.400,00 €
Code-Review	Ausbilder	2h	100,00 €	20,00 €	120,00 €
Abnahme	Ausbilder	1h	50,00 €	10,00 €	60,00 €
					1.580,00 €

Tabelle 2 - Berechnung der Projektkosten

2.7.3 Amortisationsdauer

Im Folgenden soll geprüft werden, ab welchem Zeitpunkt sich das Projekt amortisiert. Ein Mitarbeiter der Schleier-IT sollte täglich diese Berechnungen machen und das Ergebnis dem Kunden schicken. Ein Mitarbeiter dem Kunden sollte die Statistiken und Bewertungen erstellen. Dafür müssen beide Mitarbeiter etwa 30 Minuten am Tag arbeiten. Dies erfordert insgesamt 1 Stunde Arbeit pro Tag. Oben wurde für einen Mitarbeiter eine beispielhafte Stundenpauschale von 50 € pro Stunde angegeben.

Demzufolge;

$1 \text{ Std.} \times 50 \text{ €} / \text{Std.} = 50 \text{ € Tageskosten}$

$1580 \text{ €} / 50 \text{ €} = 31,6 \sim 32 \text{ Tage}$

Es gibt ungefähr 20 Arbeitstage pro Monat;

$32 \text{ Tage} / 20 \text{ Arbeitstage} = 1,6 \text{ Monat}$

Nach 32 Arbeitstage oder 1,6 Monat wird sich das Projekt amortisieren.

2.8 Programmablaufplan

Für jede Aufgabe, die zur Implementierung der Anwendung erforderlich ist, wird ein Programmablaufplan erstellt. Es beschreibt den Algorithmus und die Abfolge der Operationen im Voraus, um die Aufgaben zu lösen. Das Programmablaufplan für dieses Projekt wird mit der PapDesigner-Anwendung erstellt. Beispiel Screenshots des Programmablaufplans findet sich im Anhang unter A.5 Beispiel Screenshots des Programmablaufplans.

2.9 Entwicklungsumgebung und die Bibliotheken

In diesem Abschnitt hat der Autor die folgenden Entscheidungen über die Entwicklungsumgebung und die Bibliotheken getroffen, die als Ergebnis seiner Forschung verwendet werden sollen.

Als Entwicklungsumgebung wird Visual Studio Code verwendet, da es das Erstellen von Websites mit HTML, CSS, JSON, PHP und JavaScript unterstützt und Syntaxhervorhebung, automatische Vervollständigung, eckige Klammeranpassung usw. für Programmiersprachen bietet¹.

Da es Open Source und kostenlos ist und von fast allen Servern unterstützt wird, wurde beschlossen, die Programmiersprache PHP² für die Kommunikation mit der vorhandenen MySQL-Datenbank zu verwenden.

¹ Timotic, 2018

² Thattil, 2015

NPM wird als Paketmanager für Frontend verwendet, da es über die größte Bibliothek verfügt und dem Entwickler den Komfort bietet³, die erforderlichen Module einfach zu laden. Als andere Paketmanager wird Composer verwendet. Composer ist ein anwendungsorientierter Paketmanager für die Skriptsprache PHP⁴.

Obwohl die Teilnehmerdatenstruktur keine geschlechtsspezifischen Informationen enthält, fordert der Kunde auch eine Bewertung der Teilnehmer nach Geschlecht an. Der Autor suchte nach einer Lösung, die eine hohe Genauigkeit bietet und kostenlos ist. Hierzu wird das Gender-Detector-Paket⁵ verwendet, das mit Hilfe von Composer in das Projekt aufgenommen wird. In diesem Paket wird das Geschlecht des Teilnehmers anhand seines Vornamens und seines Landes geschätzt.

Verschiedene Bibliotheken wie Chart.js, C3.js, Chartist.js, MetricsGraphics.js wurden untersucht und getestet, um zu entscheiden, welches JavaScript-Framework für Statistiken, Diagramme und Grafiken im Dashboard verwendet wird. Chart.js wurde für die Datenvisualisierung entschieden, weil es visuelle und funktionale Merkmale nach Kundenwunsch und eine kurze und einfache Syntax bietet, die ein einfaches Debuggen und Aktualisieren ermöglicht. Die Moment.js-Bibliothek wird für die Zeitrahmenauswahl verwendet, und Bootstrap, eine CSS-Bibliothek, die mit ihren vordefinierten Klassen für die Benutzeroberfläche Komfort bietet, wird neben HTML und CSS auch verwendet.

3 Implementation

In diesem Abschnitt wird das Projekt in Übereinstimmung mit Pflichtenheft, Zeit- und Ressourcenplanung, ausgewählten Entwicklungswerkzeugen und Ablaufplan durchgeführt. Dieses Projekt wird nach dem MVC-Modell entworfen. Diese Vorgehensweise bietet eine geordnete Architektur, wodurch die Möglichkeit ausgeschlossen wird, mit dem falschen Element umzugehen oder einige Elemente zu vergessen. Diese Vorgehensweise bietet auch eine klare Unterscheidung zwischen Benutzeroberfläche, Modellen und Logik. Bei dieser Methode werden zuerst die erforderlichen Klassen und Prozeduren erstellt. Durch Erstellen von Objekten aus diesen Klassen wird dann die Logik der Daten erstellt, die in die Ansicht übertragen werden sollen. dann werden die erhaltenen Daten in der Ansicht angezeigt.

³ Rhymes, 2018

⁴ Wikipedia, 2020

⁵ Michael, 2020

3.1 Benutzeroberflächen

In dieser Phase wird die Benutzeroberfläche (View) erstellt und Methoden, die in der vorherigen Phase erstellt wurden, mittels Tools wie Dropdown-Menü und Schaltflächen aufgerufen und die Ausgabe von Methoden durch die entsprechende Darstellung angezeigt. Entsprechend der ausgewählten Aktion wird eine Datenanforderung aus der entsprechenden Tabelle der Datenbank gestellt und die Berechnungsergebnisse für die ausgewählte Aktion werden angezeigt.



Abbildung 1 - Aktionsauswahl

Entsprechend der ausgewählten Aktion wird zuerst die Gesamtzahl der Teilnehmer für die gesamte Aktion, die Gesamtzahl der Gerubbelte Felder für die gesamte Aktion und die Anzahl der Teilnehmer für den aktuellen Tag angezeigt.

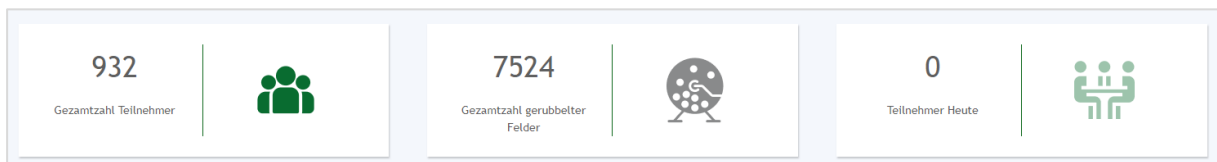


Abbildung 2 - Gesamtzahlen einer Aktion

Die ausgewählte Aktion wird zunächst standardmäßig als vollständige Aktion geladen. Anschließend können mithilfe von Zeitfilterwerkzeugen die Daten der Aktion in einem bestimmten Zeitintervall gefiltert werden.

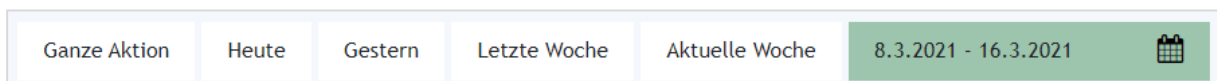


Abbildung 3 - Zeitfilterwerkzeugen

Die nächste Aktivitätstabelle zeigt Folgendes mit dem neuesten Eintrag oben: Täglicher Benutzer, tägliche neue Adressdaten, Anzahl der geriebenen Websites, kumulierte

Gesamtzahl der geriebenen Bereiche und Gesamtzahl der Teilnehmer. Diese Tabelle ändert sich dann dynamisch basierend auf der Auswahl von Heute, Gestern, Letzte Woche, Aktuelle Woche und einem beliebigen Datumsbereich.

Aktivität					
Datum	User pro Tag	Neue Adressdaten pro Tag	Anzahl gerubelter Felder	Gesamt gerubelter Felder	Gesamt Teilnehmer
13-03-2021	1	1	382	6839	1133
12-03-2021	1	1	333	6457	1132
11-03-2021	1	1	266	6124	1131
10-03-2021	1	1	215	5858	1130
09-03-2021	1	1	98	5643	1129

Abbildung 4 - Aktivitätstabelle

Anschließend wird das Geschlechterverhältnis der Teilnehmer für die gesamte Aktion in einem Donut-Diagramm und das Altersverteilungsverhältnis in einem Balkendiagramm angezeigt. In der Herkunft-Tabelle wird zunehmend die Verteilung der Teilnehmer nach Bundesländern für alle Aktionen angezeigt.

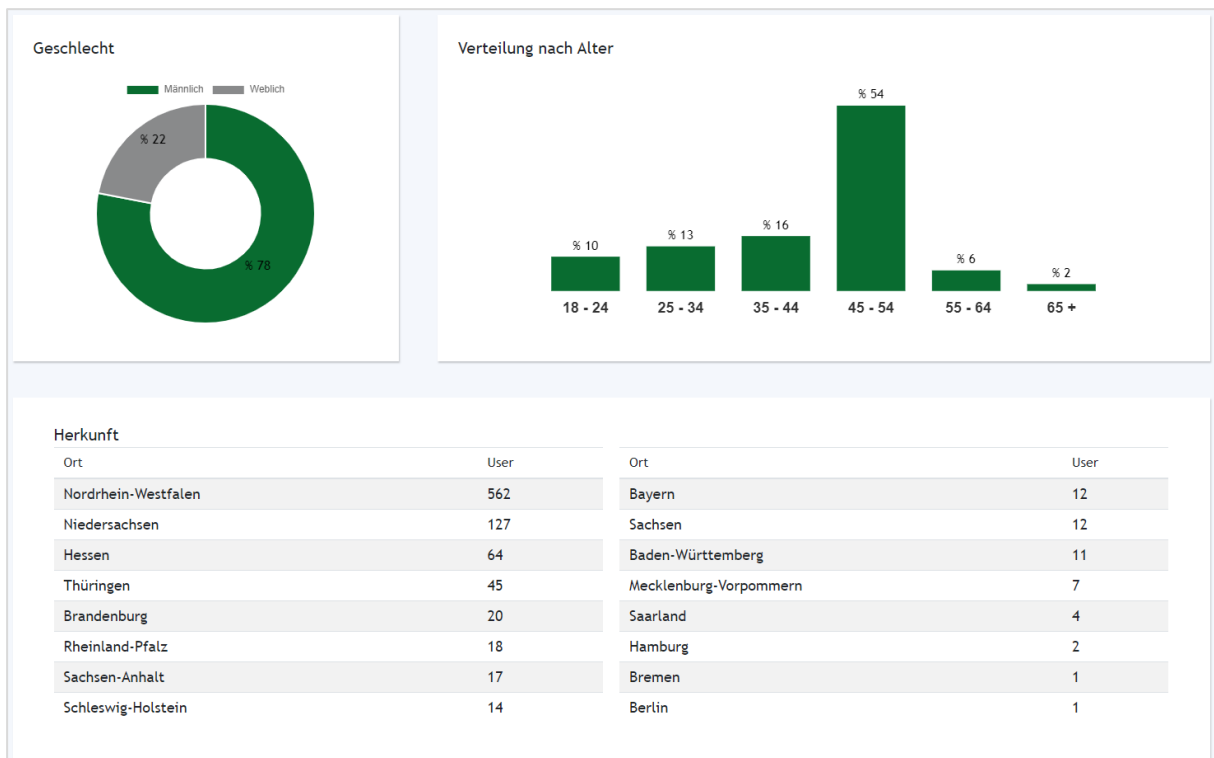


Abbildung 5 – Geschlechter - Alter Verhältnis, Herkunft-Tabelle

Am Ende des Dashboards befindet sich ein Abschnitt zum Aktionsvergleich. Alle Aktionen in der Datenbank werden Jahr für Jahr miteinander verglichen, indem die Zahlen in den beiden linearen Diagrammen von Benutzer und Gerubbelter Felder verglichen werden.

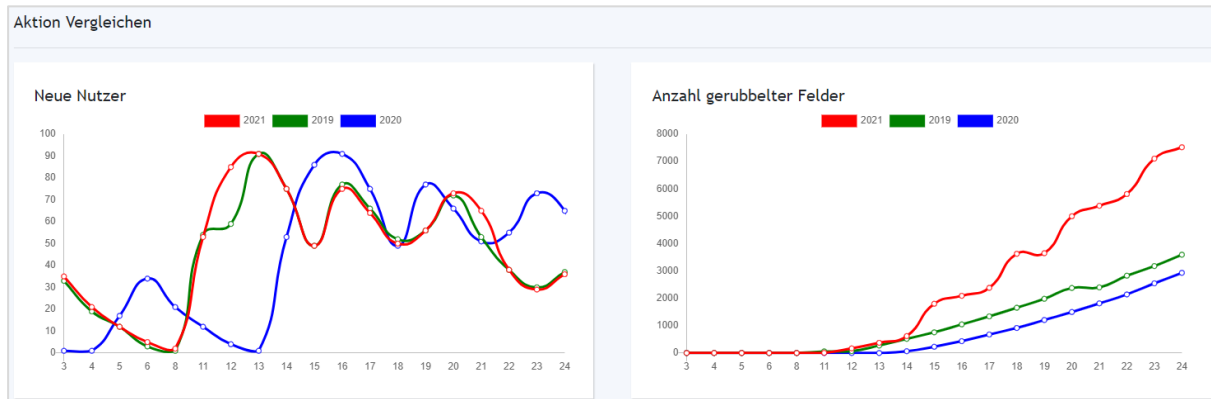


Abbildung 6 - Aktionsvergleich

Die Grundstruktur der Benutzeroberfläche wird mit HTML, CSS, Bootstrap, dynamische Abschnitte wie Tabelle werden mit der JavaScript Programmiersprache erstellt. Die Bibliothek JQuery und Moment.js wird für die Zeitintervallauswahlfunktion verwendet. Chart.js, eine kostenlose Open-Source-JavaScript-Bibliothek, wird für die Datenvisualisierung verwendet.

Das Design der Benutzeroberfläche wurde von einem Mediengestalter im Voraus mit Figma, einem webbasierten Vektorgrafik-Editor und Prototypen-Tool, gemäß den Kundenanforderungen vorbereitet. In Übereinstimmung mit diesem Design wird eine Benutzeroberfläche erstellt. Das verwendete Thema wird mit dem in anderen Veltins-Anwendungen verwendeten Thema kompatibel. Ein Screenshot der Benutzeroberfläche findet sich im Anhang unter A.6 Screenshots der Benutzeroberfläche.

3.2 Datenbankkommunikation

In dieser Phase wird die Verbindung der Dashboard-Anwendung mit MySQL-Datenbanken hergestellt. Die Struktur, die hier erstellt werden muss, sollte sowohl gegen SQL-Injektionen sicher als auch flexibel genug sein, um einen einfachen Zugriff auf mehr als eine Datenbank zu ermöglichen. Hierzu wird das MySQLi-Objekt von PHP verwendet, das als Datenbankabstraktionsschicht fungiert. MySQLi ist eine verbesserte Erweiterung von PHP zum Zugriff auf MySQL-Datenbanken und bietet die Freiheit mit mehr als einer Datenbank zu arbeiten.

Mit der PHP und MySQLi-Objekt wird eine Verbindungsklasse (Model) erstellt, um eine Verbindung zu Datenbanken auf dem MySQL-Server herzustellen. Für diese Klasse wird eine Methode erstellt, die den Datenbanknamen als Parameter verwendet, eine SQL-Abfrage an die Datenbank sendet und das Abfrageergebnis zurückgibt. In dieser Funktion wird die Geschlechtsbestimmung durchgeführt, die durch Erstellen eines Objekts der Gender-Detector-Klasse erfolgt.

Dann wird in einer anderen PHP-Datei (Control) ein Objekt dieser Linkklasse erstellt, die Methode der Klasse wird über das Objekt aufgerufen und das Ergebnis wird in das JSON-Format konvertiert. Damit die aktuelle Datenbank beim ersten Öffnen der Seite standardmäßig angezeigt wird, wird eine andere Methode erstellt, um die Namen der Datenbanken in MySQL Server abzufragen.

3.3 Geschäftslogik

Um beim Start standardmäßig die aktuelle Gewinnspieldaten zu laden, wird in der JavaScript-Datei (Control) eine Funktion erstellt, die beim erstmaligen Laden des Control Panels ausgeführt wird und eine AJAX-Abfrage an die entsprechende ausführt PHP-Datei.

Diese Funktion übernimmt Datenbanknamen aus der Datenbank und gibt die aktuelle Datenbank zurück. Es wird eine weitere Funktion geschrieben, die diesen zurückgegebenen aktuellen Datenbanknamen als Parameter verwendet. Diese Funktion ruft die erforderlichen Daten über eine andere AJAX-Abfrage aus der Datenbank ab und ermöglicht die Ausführung anderer Funktionen. Nachfolgende Datenanforderungen werden basierend auf der vom Benutzer gewählten Aktion gestellt.

Die Datenbanknamen von Gewinnspiel in MySQL Server werden mit einem Zeitstempel im Epochenformat am Anfang und einer benutzerdefinierten Erweiterung am Ende gespeichert. Dies geschieht, um die für das Dashboard erforderlichen Datenbanken von anderen zu unterscheiden und festzustellen, was aktuell ist. Die Anfrage wird unter Berücksichtigung dieser Anhänge gestellt.

Andere Funktionen zum Analysieren, Formatieren und Interpretieren von Daten werden erstellt, um die erforderlichen Datenobjekte für grafische Elemente zu erstellen. Mithilfe der

Chart.js-Bibliothek werden dann Objekte aus der Chart-Klasse erstellt und diese Datenobjekte verwendet, um die Visualisierung der Daten zu ermöglichen.

In dieser Phase werden auch die Methoden erstellt, die für einige benutzerfreundliche Oberflächenfunktionen erforderlich sind. Beispiel: Ändern der Farbe der aktiven Schaltfläche, Schreiben des ausgewählten Datumsbereichs als Text der Schaltfläche usw. JavaScript und eine freie JavaScript-Bibliothek jQuery wird für Frontend-Funktionen verwendet. JQuery bietet eine einfache Syntax für XMLHttpRequest, ein JavaScript-Objekt, das zum Übertragen von Daten über HTTP verwendet wird, und viele andere JS-Funktionen. Auf diese Weise können Inhalte asynchron hochgeladen werden.

Die Postleitzahlen der Teilnehmer sind in der Datenstruktur verfügbar. Das Dashboard sollte jedoch eine Tabelle enthalten, die die Verteilung der Teilnehmer nach Bundesländern zeigt. Zu diesem Zweck wurde eine Open Source JSON-Datei, die alle Postleitzahlen und Bundesländer für Deutschland enthält, aus dem Internet heruntergeladen und dem Projekt hinzugefügt. In dieser JSON-Datei wurden die Bundesländer der Teilnehmer durch Vergleich der Bundesländer und Postleitzahlen mit den Postleitzahlen in der Datenbank ermittelt und eine Verteilungstabelle nach Bundesländern erstellt.

4 Test/Qualitätssicherung

4.1 Genauigkeit

Mehrere Beispieldatenbanken und Daten wurden zum Testen der Anwendung erstellt. Es wurde getestet, ob die aus der Datenbank erhaltenen Daten berechnet und die korrekten Ergebnisse auf die Benutzeroberfläche übertragen wurden. Die Daten hierzu wurden mit der Console.log-Methode auf die Konsole gedruckt, manche Berechnungen wurden manuell durchgeführt und die Richtigkeit von den Berechnungen überprüft.

4.2 Benutzerfreundlichkeit & Verständlichkeit

Es wird sichergestellt, dass alle Schaltflächen, Menüs, Tabellen, Grafiken und Informationen auf der Benutzeroberfläche so gestaltet sind, dass sie für den Benutzer leicht verständlich sind, und dass informative Notizen oder Abbildungen verwendet werden. Zu diesem Zweck wurden die erforderlichen Erklärungen unter Verwendung des Attributs "title" in den HTML-Tags abgegeben, die aktiv sind, wenn der Benutzer den Mauszeiger über das Objekt bewegt.

4.3 Ausnahmebehandlung

Es wird sichergestellt, dass eine Fehlermeldung in der gesprochenen Sprache für alle Ausnahmen generiert wird, auf die der Benutzer möglicherweise stößt, einschließlich der Anmelde- und Abmeldeschritte. Dies sind Situationen, in denen der Benutzer keine Verbindung zum Internet herstellen kann, falsche Eingabe von Benutzername und Kennwort, Serverfehler, Probleme mit der Datenbankverbindung, die zu Beginn und während des Aktionswechsels auftreten.

5 Dokumentation

Die Projektdokumentation wurde als prozessorientierter Projektbericht fortwährend innerhalb des Projektes vom Autor manuell erstellt. Dieser beschreibt die einzelnen Phasen und deren Inhalte, die im Laufe des Projektes durchlaufen wurden.

6 Fazit

6.1 Ist-Soll Vergleich

Rückblickend wurden alle im Pflichtenheft definierten Anforderungen umgesetzt. Der zu Anfang erstellte Projektplan konnte ebenfalls eingehalten werden. In der Tabelle Soll/ Ist-Vergleich ist der benötigte Zeitaufwand gegenübergestellt. Es ist zu erkennen, dass es eine gewisse Zeitdifferenz bei der Implementationsphase gegeben hat. Da die Implementation-Phase des Dashboards mehr Zeit benötigte, konnte die Pufferzeit für diese Abweichung verwendet werden. Somit wurde das Projekt in den festgelegten 70 Stunden umgesetzt.

Projektphase	Geplante Zeit	Tatsächliche Zeit	Differenz
A - Analyse & Planung	20h	20h	0h
B - Implementation	32h	34h	+2h
C - Test/Qualitätssicherung	4h	4h	0h
D - Dokumentation	13h	13h	0h
E - Nachbearbeitung	2h	0h	-2h
Gesamt			70h

Tabelle 3 - Soll/Ist Vergleich

6.2 Gewonnene Erkenntnisse

Das Projekt ermöglichte es dem Autor, den Ablauf eines Softwareprojekts von Anfang bis Ende zu betrachten. Der wichtigste Faktor war nicht nur, in direkten Programmieraufgaben aktiv zu sein, sondern auch in erster Linie für die genaue Definition der technischen Anforderungen, die Auswahl des Entwicklungsprozesses und das Management der einzelnen Projektphasen verantwortlich zu sein. Darüber hinaus konnte der Autor wichtige Erfahrung im Bereich PHP-Programmierung und der MySQL-Datenbank sammeln.

Literaturverzeichnis

Michael, J. (31.12.2020). Gender Detector

<https://github.com/tuqqu/gender-detector>

Rhymes, C.S. (31.03.2018). What is NPM and why should I use it?

<https://medium.com/@chrisrhymes/what-is-npm-and-why-should-i-use-it-ef15de78e305>

Thattil, S. (19.03.2015). 13 Vorteile von PHP

<https://www.yuhiro.de/13-vorteile-von-php/>

Timotic, M. (05.11.2018). 14 Best Web Development IDE

<https://tms-outsource.com/blog/posts/web-development-ide>

Wikipedia, (03.11.2020) Composer (Paketverwaltung)

[https://de.wikipedia.org/wiki/Composer_\(Paketverwaltung\)](https://de.wikipedia.org/wiki/Composer_(Paketverwaltung))

Anhang

A.1 Detaillierte Zeitplanung

Projektphase	Geplante Zeit
A - Analyse & Planung	19h
Projektphasen und Ressourcenplanung	3h
Durchführung einer Ist-Analyse	2h
Erstellung eines Soll-Konzeptes	2h
Durchführung der Wirtschaftlichkeitsanalyse	3h
Erarbeitung eines Pflichtenheftes	4h
Erstellung des Programmstrukturplan	3h
Auswählen der Entwicklungsumgebung und die Bibliotheken	2h
B - Implementation	32h
Aufbau der Kommunikation mit Datenbanken	2h
Erstellung der SQL-Abfragen	2h
Erstellung der Elemente zur Datenspeicherung	1h
Konzipierung der Struktur von Benutzeroberflächen	3h
Erstellung der Logik für Login	3h
Erstellung der Logik für Aktivitäten	3h
Erstellung der Logik zur Auswertung von Geschlecht	3h
Erstellung der Logik zur Auswertung von Altersgruppen	3h
Erstellung grafischer Elemente	2h
Verbindung von Daten und grafischen Elemente	2h
Erstellung der Logik zum Wechsel von Aktionen	2h
Erstellung der Logik zum Wechsel von Datum	3h
Styling der Benutzeroberfläche	3h
C - Test/Qualitätssicherung	4h
Genauigkeit von Statistiken und Berechnungen	1h
Benutzerfreundlichkeit aller Funktionen	1h
Verständlichkeit von Tabellen und Grafiken	1h
Eine eindeutige Fehlermeldung für jeden möglichen Fehler	1h
D - Dokumentation	13h
E - Nachbearbeitung	2h
Gesamt	70h

A.2 Verwendete Ressourcen

Hardware

HP-Laptop	: Rechner
Samsung Bildschirm	: Zusätzlicher Bildschirm

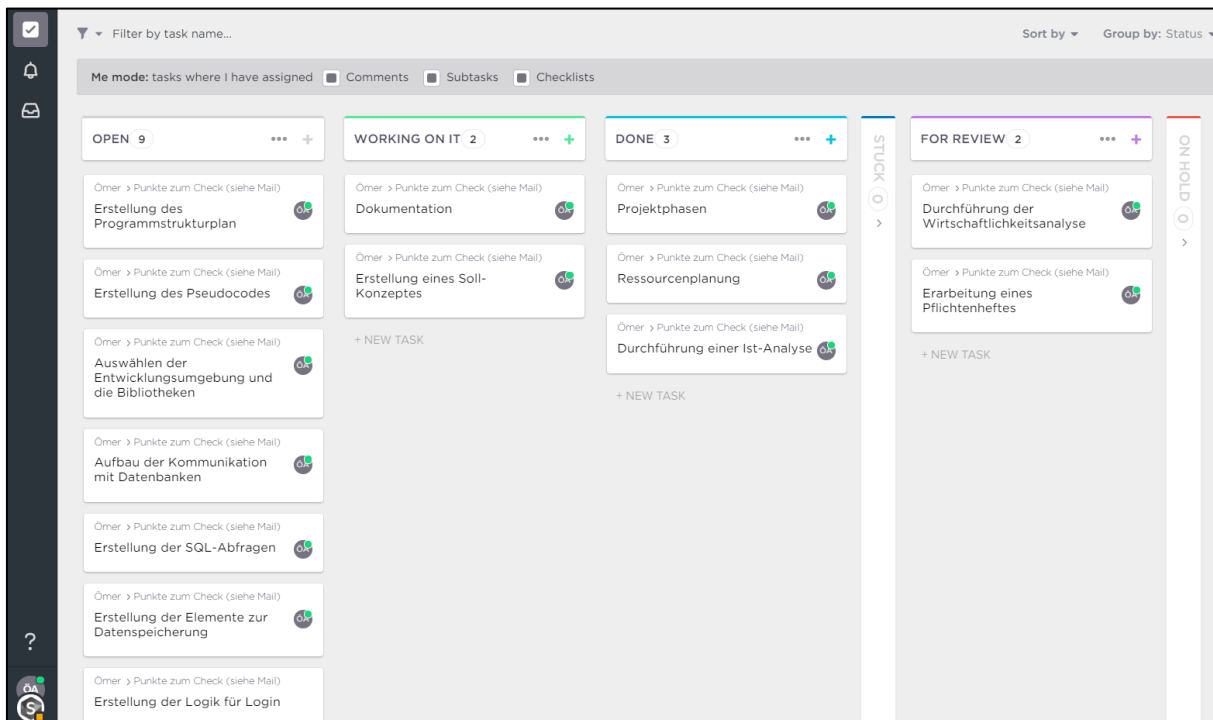
Software

Windows 10 Professional	: Betriebssystem
Visual Studio Code	: Entwicklungsumgebung
Git / GitHub	: Versionskontrolle
Npm	: Paketmanager für Frontend
Composer	: Paketmanager für PHP
Ampmps	: Apache Server, MySQL Datenbank, PHP 7.3
ClickUp	: Projektmanagement-Tool / Kanban-Board
Figma	: Designentwurf
Office 365	: Dokumentation
PapDesigner	: Programmablaufplan
Greenshot	: Screenshot

Personal

Anwendungsentwickler	: Umsetzung des Projektes
Ausbilder	: Review der Code und Abnahme

A.3 Ausschnitt des Kanban-Boards



The screenshot shows a Kanban board interface with the following columns and tasks:

- OPEN (9):**
 - Erstellung des Programmstrukturplan
 - Erstellung des Pseudocodes
 - Auswählen der Entwicklungsumgebung und die Bibliotheken
 - Aufbau der Kommunikation mit Datenbanken
 - Erstellung der SQL-Abfragen
 - Erstellung der Elemente zur Datenspeicherung
 - Erstellung der Logik für Login
- WORKING ON IT (2):**
 - Dokumentation
 - Erstellung eines Soll-Konzeptes
- DONE (3):**
 - Projektphasen
 - Ressourcenplanung
 - Durchführung einer Ist-Analyse
- STUCK (0):** (Empty column)
- FOR REVIEW (2):**
 - Durchführung der Wirtschaftlichkeitsanalyse
 - Erarbeitung eines Pflichtenheftes
- ON HOLD (0):** (Empty column)

Each task card includes a title, a progress indicator (a small green circle with a checkmark), and a link to the task details (e.g., 'Omer > Punkte zum Check (siehe Mail)').

A.4 Auszug aus dem Pflichtenheft

3 Beschreibung der Anforderungen

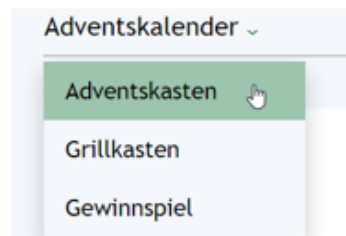
Für das Projekt besteht ein Design-Entwurf, welcher als Grundlage für die kommenden Funktionen gilt.

Link Designentwurf:

<https://www.figma.com/file/36qHU1q2TiBXTor9uBb5zm/Veltins?node-id=0%3A1>

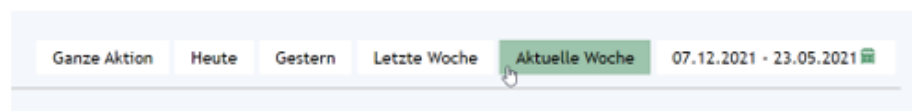
3.1 Wesentliche Teilfunktionen

a) Wechsel zwischen Aktionen



Es muss möglich sein, zwischen verschiedenen Aktionen und somit auch zwischen verschiedenen Datenbanken zu wechseln.

b) Datumsauswahl

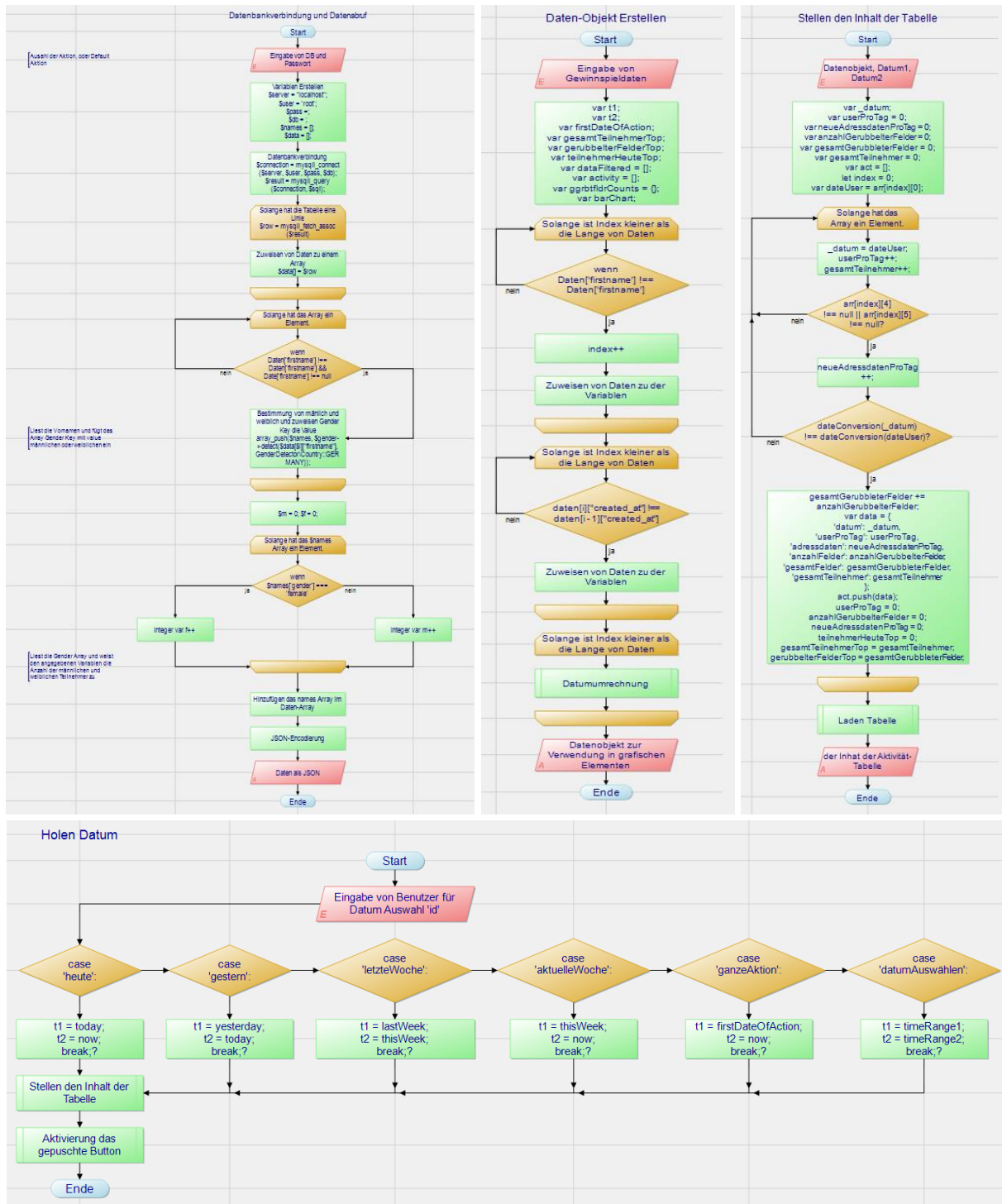


Es muss gewährleistet werden, dass auch individuelle Zeiträume innerhalb einer Aktion miteinander verglichen werden können.

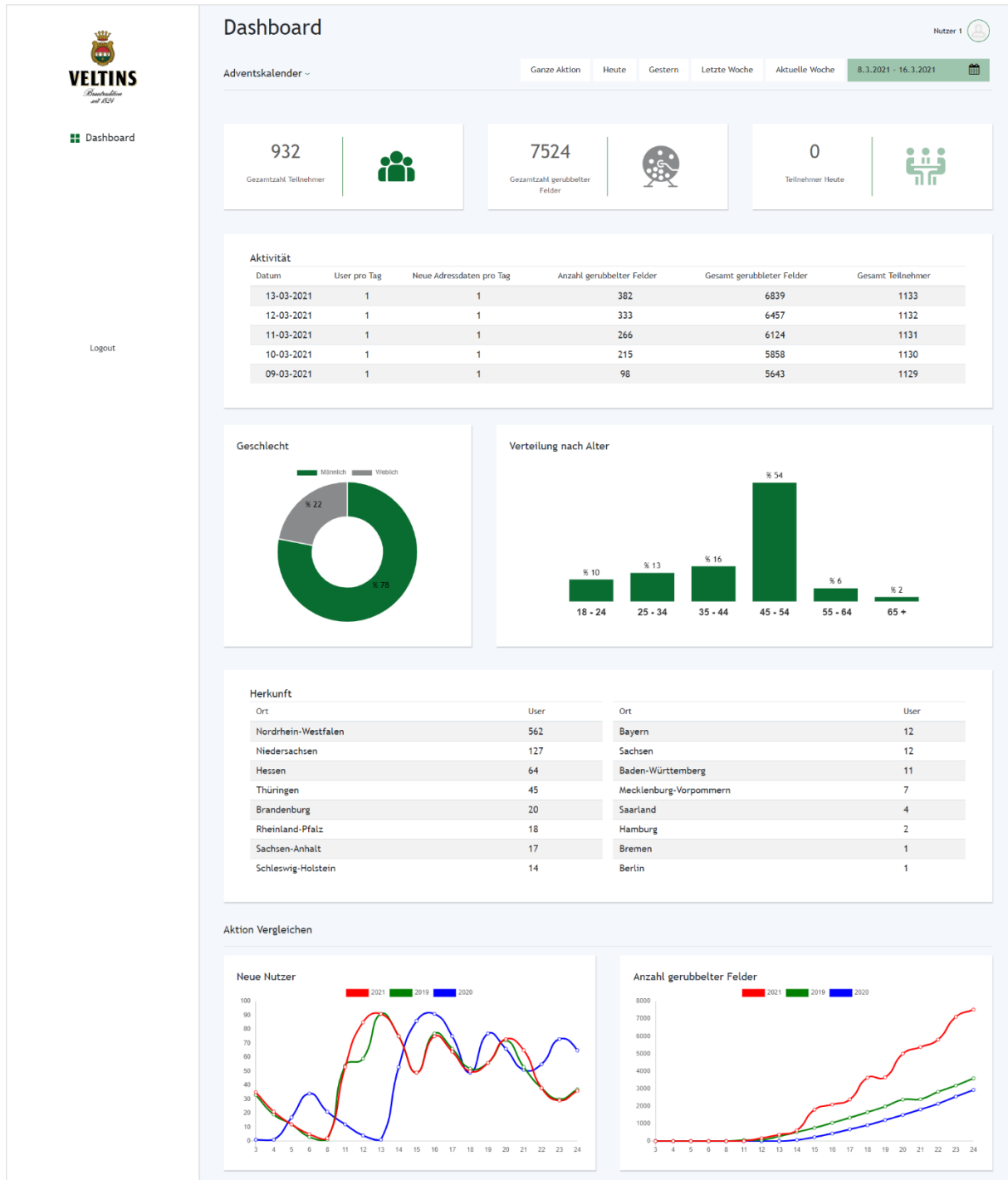
c) Modul Aktivitäten

Es soll möglich sein für jeden Tag die Anzahl der Nutzer, Anzahl der neuen Adressdaten, Anzahl der freigerubbelten Felder im Rubbel-Gewinnspiel, Gesamtzahl der gerubbelten Felder bis zu diesem Tag sowie die Gesamtzahl der Teilnehmer bis zu diesem Tag zu sehen und in tabellarischer Form gemäß Design darzustellen.

A.5 Beispiel Screenshots der Programmablaufplan



A.6 Ein Screenshot der Benutzeroberfläche



A.7 Screenshots der Quellcode

```

488 //get states from zip number
489 function getStates(arr) {
490     if (arr !== undefined) {
491         var stateArray = [];
492         var items = [];
493         $.getJSON("plz.json", function (data) {
494             $.each(data, function (key, val) {
495                 items.push([val["zipcode"], val["state"]]);
496             });
497             //make an array with states
498             for (let i = 0; i < arr.length; i++) {
499                 for (let a = 0; a < items.length; a++) {
500                     if (arr[i][4] === items[a][0]) {
501                         stateArray[i] = items[a][1];
502                     }
503                 }
504             }
505             //get how many time states repeat
506             var counts = {};
507             stateArray.forEach(function (x) {
508                 counts[x] = (counts[x] || 0) + 1;
509             });
510             //make it sortable array
511             var sortable = [];
512             for (var c in counts) {
513                 sortable.push([c, counts[c]]);
514             }
515             //sort descending
516             sortable.sort(function (a, b) {
517                 return b[1] - a[1];
518             });
519             //table variables
520             var herkunft = [];
521             var user = [];
522             //table variables
523             var herkunft = [];
524             var user = [];
525             for (let i = 0; i < sortable.length; i++) {
526                 const element = sortable[i];
527                 herkunft.push(element[0]);
528                 user.push(element[1]);
529             }
530             //control group of nonexistent states
531             var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
532             //check which states are not in the game,
533             //and add them with 0 value
534             var a = [];
535             for (var i = 0; i < bundesland.length; i++) {
536                 a[bundesland[i]] = true;
537             }
538             for (var i = 0; i < herkunft.length; i++) {
539                 if (a[herkunft[i]]) {
540                     delete a[herkunft[i]];
541                 } else {
542                     a[herkunft[i]] = true;
543                 }
544             }
545             for (var k in a) {
546                 herkunft.push(k);
547                 user.push(0);
548             }
549             loadHerkunftTable(herkunft, user);
550             herkunft = []; //reset to null again
551             user = []; //reset to null again
552         });
553     }
554 }
555
556 //table variables
557 var herkunft = [];
558 var user = [];
559
560 for (let i = 0; i < sortable.length; i++) {
561     const element = sortable[i];
562     herkunft.push(element[0]);
563     user.push(element[1]);
564 }
565
566 //control group of nonexistent states
567 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
568 //check which states are not in the game,
569 //and add them with 0 value
570 var a = [];
571 for (var i = 0; i < bundesland.length; i++) {
572     a[bundesland[i]] = true;
573 }
574 for (var i = 0; i < herkunft.length; i++) {
575     if (a[herkunft[i]]) {
576         delete a[herkunft[i]];
577     } else {
578         a[herkunft[i]] = true;
579     }
580 }
581 for (var k in a) {
582     herkunft.push(k);
583     user.push(0);
584 }
585 loadHerkunftTable(herkunft, user);
586 herkunft = []; //reset to null again
587 user = []; //reset to null again
588
589 //table variables
590 var herkunft = [];
591 var user = [];
592
593 for (let i = 0; i < sortable.length; i++) {
594     const element = sortable[i];
595     herkunft.push(element[0]);
596     user.push(element[1]);
597 }
598 //control group of nonexistent states
599 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
600 //check which states are not in the game,
601 //and add them with 0 value
602 var a = [];
603 for (var i = 0; i < bundesland.length; i++) {
604     a[bundesland[i]] = true;
605 }
606 for (var i = 0; i < herkunft.length; i++) {
607     if (a[herkunft[i]]) {
608         delete a[herkunft[i]];
609     } else {
610         a[herkunft[i]] = true;
611     }
612 }
613 for (var k in a) {
614     herkunft.push(k);
615     user.push(0);
616 }
617 loadHerkunftTable(herkunft, user);
618 herkunft = []; //reset to null again
619 user = []; //reset to null again
620
621 //table variables
622 var herkunft = [];
623 var user = [];
624
625 for (let i = 0; i < sortable.length; i++) {
626     const element = sortable[i];
627     herkunft.push(element[0]);
628     user.push(element[1]);
629 }
630 //control group of nonexistent states
631 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
632 //check which states are not in the game,
633 //and add them with 0 value
634 var a = [];
635 for (var i = 0; i < bundesland.length; i++) {
636     a[bundesland[i]] = true;
637 }
638 for (var i = 0; i < herkunft.length; i++) {
639     if (a[herkunft[i]]) {
640         delete a[herkunft[i]];
641     } else {
642         a[herkunft[i]] = true;
643     }
644 }
645 for (var k in a) {
646     herkunft.push(k);
647     user.push(0);
648 }
649 loadHerkunftTable(herkunft, user);
650 herkunft = []; //reset to null again
651 user = []; //reset to null again
652
653 //table variables
654 var herkunft = [];
655 var user = [];
656
657 for (let i = 0; i < sortable.length; i++) {
658     const element = sortable[i];
659     herkunft.push(element[0]);
660     user.push(element[1]);
661 }
662 //control group of nonexistent states
663 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
664 //check which states are not in the game,
665 //and add them with 0 value
666 var a = [];
667 for (var i = 0; i < bundesland.length; i++) {
668     a[bundesland[i]] = true;
669 }
670 for (var i = 0; i < herkunft.length; i++) {
671     if (a[herkunft[i]]) {
672         delete a[herkunft[i]];
673     } else {
674         a[herkunft[i]] = true;
675     }
676 }
677 for (var k in a) {
678     herkunft.push(k);
679     user.push(0);
680 }
681 loadHerkunftTable(herkunft, user);
682 herkunft = []; //reset to null again
683 user = []; //reset to null again
684
685 //table variables
686 var herkunft = [];
687 var user = [];
688
689 for (let i = 0; i < sortable.length; i++) {
690     const element = sortable[i];
691     herkunft.push(element[0]);
692     user.push(element[1]);
693 }
694 //control group of nonexistent states
695 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
696 //check which states are not in the game,
697 //and add them with 0 value
698 var a = [];
699 for (var i = 0; i < bundesland.length; i++) {
700     a[bundesland[i]] = true;
701 }
702 for (var i = 0; i < herkunft.length; i++) {
703     if (a[herkunft[i]]) {
704         delete a[herkunft[i]];
705     } else {
706         a[herkunft[i]] = true;
707     }
708 }
709 for (var k in a) {
710     herkunft.push(k);
711     user.push(0);
712 }
713 loadHerkunftTable(herkunft, user);
714 herkunft = []; //reset to null again
715 user = []; //reset to null again
716
717 //table variables
718 var herkunft = [];
719 var user = [];
720
721 for (let i = 0; i < sortable.length; i++) {
722     const element = sortable[i];
723     herkunft.push(element[0]);
724     user.push(element[1]);
725 }
726 //control group of nonexistent states
727 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
728 //check which states are not in the game,
729 //and add them with 0 value
730 var a = [];
731 for (var i = 0; i < bundesland.length; i++) {
732     a[bundesland[i]] = true;
733 }
734 for (var i = 0; i < herkunft.length; i++) {
735     if (a[herkunft[i]]) {
736         delete a[herkunft[i]];
737     } else {
738         a[herkunft[i]] = true;
739     }
740 }
741 for (var k in a) {
742     herkunft.push(k);
743     user.push(0);
744 }
745 loadHerkunftTable(herkunft, user);
746 herkunft = []; //reset to null again
747 user = []; //reset to null again
748
749 //table variables
750 var herkunft = [];
751 var user = [];
752
753 for (let i = 0; i < sortable.length; i++) {
754     const element = sortable[i];
755     herkunft.push(element[0]);
756     user.push(element[1]);
757 }
758 //control group of nonexistent states
759 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
760 //check which states are not in the game,
761 //and add them with 0 value
762 var a = [];
763 for (var i = 0; i < bundesland.length; i++) {
764     a[bundesland[i]] = true;
765 }
766 for (var i = 0; i < herkunft.length; i++) {
767     if (a[herkunft[i]]) {
768         delete a[herkunft[i]];
769     } else {
770         a[herkunft[i]] = true;
771     }
772 }
773 for (var k in a) {
774     herkunft.push(k);
775     user.push(0);
776 }
777 loadHerkunftTable(herkunft, user);
778 herkunft = []; //reset to null again
779 user = []; //reset to null again
780
781 //table variables
782 var herkunft = [];
783 var user = [];
784
785 for (let i = 0; i < sortable.length; i++) {
786     const element = sortable[i];
787     herkunft.push(element[0]);
788     user.push(element[1]);
789 }
790 //control group of nonexistent states
791 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
792 //check which states are not in the game,
793 //and add them with 0 value
794 var a = [];
795 for (var i = 0; i < bundesland.length; i++) {
796     a[bundesland[i]] = true;
797 }
798 for (var i = 0; i < herkunft.length; i++) {
799     if (a[herkunft[i]]) {
800         delete a[herkunft[i]];
801     } else {
802         a[herkunft[i]] = true;
803     }
804 }
805 for (var k in a) {
806     herkunft.push(k);
807     user.push(0);
808 }
809 loadHerkunftTable(herkunft, user);
810 herkunft = []; //reset to null again
811 user = []; //reset to null again
812
813 //table variables
814 var herkunft = [];
815 var user = [];
816
817 for (let i = 0; i < sortable.length; i++) {
818     const element = sortable[i];
819     herkunft.push(element[0]);
820     user.push(element[1]);
821 }
822 //control group of nonexistent states
823 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
824 //check which states are not in the game,
825 //and add them with 0 value
826 var a = [];
827 for (var i = 0; i < bundesland.length; i++) {
828     a[bundesland[i]] = true;
829 }
830 for (var i = 0; i < herkunft.length; i++) {
831     if (a[herkunft[i]]) {
832         delete a[herkunft[i]];
833     } else {
834         a[herkunft[i]] = true;
835     }
836 }
837 for (var k in a) {
838     herkunft.push(k);
839     user.push(0);
840 }
841 loadHerkunftTable(herkunft, user);
842 herkunft = []; //reset to null again
843 user = []; //reset to null again
844
845 //table variables
846 var herkunft = [];
847 var user = [];
848
849 for (let i = 0; i < sortable.length; i++) {
850     const element = sortable[i];
851     herkunft.push(element[0]);
852     user.push(element[1]);
853 }
854 //control group of nonexistent states
855 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
856 //check which states are not in the game,
857 //and add them with 0 value
858 var a = [];
859 for (var i = 0; i < bundesland.length; i++) {
860     a[bundesland[i]] = true;
861 }
862 for (var i = 0; i < herkunft.length; i++) {
863     if (a[herkunft[i]]) {
864         delete a[herkunft[i]];
865     } else {
866         a[herkunft[i]] = true;
867     }
868 }
869 for (var k in a) {
870     herkunft.push(k);
871     user.push(0);
872 }
873 loadHerkunftTable(herkunft, user);
874 herkunft = []; //reset to null again
875 user = []; //reset to null again
876
877 //table variables
878 var herkunft = [];
879 var user = [];
880
881 for (let i = 0; i < sortable.length; i++) {
882     const element = sortable[i];
883     herkunft.push(element[0]);
884     user.push(element[1]);
885 }
886 //control group of nonexistent states
887 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
888 //check which states are not in the game,
889 //and add them with 0 value
890 var a = [];
891 for (var i = 0; i < bundesland.length; i++) {
892     a[bundesland[i]] = true;
893 }
894 for (var i = 0; i < herkunft.length; i++) {
895     if (a[herkunft[i]]) {
896         delete a[herkunft[i]];
897     } else {
898         a[herkunft[i]] = true;
899     }
900 }
901 for (var k in a) {
902     herkunft.push(k);
903     user.push(0);
904 }
905 loadHerkunftTable(herkunft, user);
906 herkunft = []; //reset to null again
907 user = []; //reset to null again
908
909 //table variables
910 var herkunft = [];
911 var user = [];
912
913 for (let i = 0; i < sortable.length; i++) {
914     const element = sortable[i];
915     herkunft.push(element[0]);
916     user.push(element[1]);
917 }
918 //control group of nonexistent states
919 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
920 //check which states are not in the game,
921 //and add them with 0 value
922 var a = [];
923 for (var i = 0; i < bundesland.length; i++) {
924     a[bundesland[i]] = true;
925 }
926 for (var i = 0; i < herkunft.length; i++) {
927     if (a[herkunft[i]]) {
928         delete a[herkunft[i]];
929     } else {
930         a[herkunft[i]] = true;
931     }
932 }
933 for (var k in a) {
934     herkunft.push(k);
935     user.push(0);
936 }
937 loadHerkunftTable(herkunft, user);
938 herkunft = []; //reset to null again
939 user = []; //reset to null again
940
941 //table variables
942 var herkunft = [];
943 var user = [];
944
945 for (let i = 0; i < sortable.length; i++) {
946     const element = sortable[i];
947     herkunft.push(element[0]);
948     user.push(element[1]);
949 }
950 //control group of nonexistent states
951 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
952 //check which states are not in the game,
953 //and add them with 0 value
954 var a = [];
955 for (var i = 0; i < bundesland.length; i++) {
956     a[bundesland[i]] = true;
957 }
958 for (var i = 0; i < herkunft.length; i++) {
959     if (a[herkunft[i]]) {
960         delete a[herkunft[i]];
961     } else {
962         a[herkunft[i]] = true;
963     }
964 }
965 for (var k in a) {
966     herkunft.push(k);
967     user.push(0);
968 }
969 loadHerkunftTable(herkunft, user);
970 herkunft = []; //reset to null again
971 user = []; //reset to null again
972
973 //table variables
974 var herkunft = [];
975 var user = [];
976
977 for (let i = 0; i < sortable.length; i++) {
978     const element = sortable[i];
979     herkunft.push(element[0]);
980     user.push(element[1]);
981 }
982 //control group of nonexistent states
983 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
984 //check which states are not in the game,
985 //and add them with 0 value
986 var a = [];
987 for (var i = 0; i < bundesland.length; i++) {
988     a[bundesland[i]] = true;
989 }
990 for (var i = 0; i < herkunft.length; i++) {
991     if (a[herkunft[i]]) {
992         delete a[herkunft[i]];
993     } else {
994         a[herkunft[i]] = true;
995     }
996 }
997 for (var k in a) {
998     herkunft.push(k);
999     user.push(0);
1000 }
1001 loadHerkunftTable(herkunft, user);
1002 herkunft = []; //reset to null again
1003 user = []; //reset to null again
1004
1005 //table variables
1006 var herkunft = [];
1007 var user = [];
1008
1009 for (let i = 0; i < sortable.length; i++) {
1010     const element = sortable[i];
1011     herkunft.push(element[0]);
1012     user.push(element[1]);
1013 }
1014 //control group of nonexistent states
1015 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1016 //check which states are not in the game,
1017 //and add them with 0 value
1018 var a = [];
1019 for (var i = 0; i < bundesland.length; i++) {
1020     a[bundesland[i]] = true;
1021 }
1022 for (var i = 0; i < herkunft.length; i++) {
1023     if (a[herkunft[i]]) {
1024         delete a[herkunft[i]];
1025     } else {
1026         a[herkunft[i]] = true;
1027     }
1028 }
1029 for (var k in a) {
1030     herkunft.push(k);
1031     user.push(0);
1032 }
1033 loadHerkunftTable(herkunft, user);
1034 herkunft = []; //reset to null again
1035 user = []; //reset to null again
1036
1037 //table variables
1038 var herkunft = [];
1039 var user = [];
1040
1041 for (let i = 0; i < sortable.length; i++) {
1042     const element = sortable[i];
1043     herkunft.push(element[0]);
1044     user.push(element[1]);
1045 }
1046 //control group of nonexistent states
1047 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1048 //check which states are not in the game,
1049 //and add them with 0 value
1050 var a = [];
1051 for (var i = 0; i < bundesland.length; i++) {
1052     a[bundesland[i]] = true;
1053 }
1054 for (var i = 0; i < herkunft.length; i++) {
1055     if (a[herkunft[i]]) {
1056         delete a[herkunft[i]];
1057     } else {
1058         a[herkunft[i]] = true;
1059     }
1060 }
1061 for (var k in a) {
1062     herkunft.push(k);
1063     user.push(0);
1064 }
1065 loadHerkunftTable(herkunft, user);
1066 herkunft = []; //reset to null again
1067 user = []; //reset to null again
1068
1069 //table variables
1070 var herkunft = [];
1071 var user = [];
1072
1073 for (let i = 0; i < sortable.length; i++) {
1074     const element = sortable[i];
1075     herkunft.push(element[0]);
1076     user.push(element[1]);
1077 }
1078 //control group of nonexistent states
1079 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1080 //check which states are not in the game,
1081 //and add them with 0 value
1082 var a = [];
1083 for (var i = 0; i < bundesland.length; i++) {
1084     a[bundesland[i]] = true;
1085 }
1086 for (var i = 0; i < herkunft.length; i++) {
1087     if (a[herkunft[i]]) {
1088         delete a[herkunft[i]];
1089     } else {
1090         a[herkunft[i]] = true;
1091     }
1092 }
1093 for (var k in a) {
1094     herkunft.push(k);
1095     user.push(0);
1096 }
1097 loadHerkunftTable(herkunft, user);
1098 herkunft = []; //reset to null again
1099 user = []; //reset to null again
1100
1101 //table variables
1102 var herkunft = [];
1103 var user = [];
1104
1105 for (let i = 0; i < sortable.length; i++) {
1106     const element = sortable[i];
1107     herkunft.push(element[0]);
1108     user.push(element[1]);
1109 }
1110 //control group of nonexistent states
1111 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1112 //check which states are not in the game,
1113 //and add them with 0 value
1114 var a = [];
1115 for (var i = 0; i < bundesland.length; i++) {
1116     a[bundesland[i]] = true;
1117 }
1118 for (var i = 0; i < herkunft.length; i++) {
1119     if (a[herkunft[i]]) {
1120         delete a[herkunft[i]];
1121     } else {
1122         a[herkunft[i]] = true;
1123     }
1124 }
1125 for (var k in a) {
1126     herkunft.push(k);
1127     user.push(0);
1128 }
1129 loadHerkunftTable(herkunft, user);
1130 herkunft = []; //reset to null again
1131 user = []; //reset to null again
1132
1133 //table variables
1134 var herkunft = [];
1135 var user = [];
1136
1137 for (let i = 0; i < sortable.length; i++) {
1138     const element = sortable[i];
1139     herkunft.push(element[0]);
1140     user.push(element[1]);
1141 }
1142 //control group of nonexistent states
1143 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1144 //check which states are not in the game,
1145 //and add them with 0 value
1146 var a = [];
1147 for (var i = 0; i < bundesland.length; i++) {
1148     a[bundesland[i]] = true;
1149 }
1150 for (var i = 0; i < herkunft.length; i++) {
1151     if (a[herkunft[i]]) {
1152         delete a[herkunft[i]];
1153     } else {
1154         a[herkunft[i]] = true;
1155     }
1156 }
1157 for (var k in a) {
1158     herkunft.push(k);
1159     user.push(0);
1160 }
1161 loadHerkunftTable(herkunft, user);
1162 herkunft = []; //reset to null again
1163 user = []; //reset to null again
1164
1165 //table variables
1166 var herkunft = [];
1167 var user = [];
1168
1169 for (let i = 0; i < sortable.length; i++) {
1170     const element = sortable[i];
1171     herkunft.push(element[0]);
1172     user.push(element[1]);
1173 }
1174 //control group of nonexistent states
1175 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1176 //check which states are not in the game,
1177 //and add them with 0 value
1178 var a = [];
1179 for (var i = 0; i < bundesland.length; i++) {
1180     a[bundesland[i]] = true;
1181 }
1182 for (var i = 0; i < herkunft.length; i++) {
1183     if (a[herkunft[i]]) {
1184         delete a[herkunft[i]];
1185     } else {
1186         a[herkunft[i]] = true;
1187     }
1188 }
1189 for (var k in a) {
1190     herkunft.push(k);
1191     user.push(0);
1192 }
1193 loadHerkunftTable(herkunft, user);
1194 herkunft = []; //reset to null again
1195 user = []; //reset to null again
1196
1197 //table variables
1198 var herkunft = [];
1199 var user = [];
1200
1201 for (let i = 0; i < sortable.length; i++) {
1202     const element = sortable[i];
1203     herkunft.push(element[0]);
1204     user.push(element[1]);
1205 }
1206 //control group of nonexistent states
1207 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1208 //check which states are not in the game,
1209 //and add them with 0 value
1210 var a = [];
1211 for (var i = 0; i < bundesland.length; i++) {
1212     a[bundesland[i]] = true;
1213 }
1214 for (var i = 0; i < herkunft.length; i++) {
1215     if (a[herkunft[i]]) {
1216         delete a[herkunft[i]];
1217     } else {
1218         a[herkunft[i]] = true;
1219     }
1220 }
1221 for (var k in a) {
1222     herkunft.push(k);
1223     user.push(0);
1224 }
1225 loadHerkunftTable(herkunft, user);
1226 herkunft = []; //reset to null again
1227 user = []; //reset to null again
1228
1229 //table variables
1230 var herkunft = [];
1231 var user = [];
1232
1233 for (let i = 0; i < sortable.length; i++) {
1234     const element = sortable[i];
1235     herkunft.push(element[0]);
1236     user.push(element[1]);
1237 }
1238 //control group of nonexistent states
1239 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1240 //check which states are not in the game,
1241 //and add them with 0 value
1242 var a = [];
1243 for (var i = 0; i < bundesland.length; i++) {
1244     a[bundesland[i]] = true;
1245 }
1246 for (var i = 0; i < herkunft.length; i++) {
1247     if (a[herkunft[i]]) {
1248         delete a[herkunft[i]];
1249     } else {
1250         a[herkunft[i]] = true;
1251     }
1252 }
1253 for (var k in a) {
1254     herkunft.push(k);
1255     user.push(0);
1256 }
1257 loadHerkunftTable(herkunft, user);
1258 herkunft = []; //reset to null again
1259 user = []; //reset to null again
1260
1261 //table variables
1262 var herkunft = [];
1263 var user = [];
1264
1265 for (let i = 0; i < sortable.length; i++) {
1266     const element = sortable[i];
1267     herkunft.push(element[0]);
1268     user.push(element[1]);
1269 }
1270 //control group of nonexistent states
1271 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1272 //check which states are not in the game,
1273 //and add them with 0 value
1274 var a = [];
1275 for (var i = 0; i < bundesland.length; i++) {
1276     a[bundesland[i]] = true;
1277 }
1278 for (var i = 0; i < herkunft.length; i++) {
1279     if (a[herkunft[i]]) {
1280         delete a[herkunft[i]];
1281     } else {
1282         a[herkunft[i]] = true;
1283     }
1284 }
1285 for (var k in a) {
1286     herkunft.push(k);
1287     user.push(0);
1288 }
1289 loadHerkunftTable(herkunft, user);
1290 herkunft = []; //reset to null again
1291 user = []; //reset to null again
1292
1293 //table variables
1294 var herkunft = [];
1295 var user = [];
1296
1297 for (let i = 0; i < sortable.length; i++) {
1298     const element = sortable[i];
1299     herkunft.push(element[0]);
1300     user.push(element[1]);
1301 }
1302 //control group of nonexistent states
1303 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1304 //check which states are not in the game,
1305 //and add them with 0 value
1306 var a = [];
1307 for (var i = 0; i < bundesland.length; i++) {
1308     a[bundesland[i]] = true;
1309 }
1310 for (var i = 0; i < herkunft.length; i++) {
1311     if (a[herkunft[i]]) {
1312         delete a[herkunft[i]];
1313     } else {
1314         a[herkunft[i]] = true;
1315     }
1316 }
1317 for (var k in a) {
1318     herkunft.push(k);
1319     user.push(0);
1320 }
1321 loadHerkunftTable(herkunft, user);
1322 herkunft = []; //reset to null again
1323 user = []; //reset to null again
1324
1325 //table variables
1326 var herkunft = [];
1327 var user = [];
1328
1329 for (let i = 0; i < sortable.length; i++) {
1330     const element = sortable[i];
1331     herkunft.push(element[0]);
1332     user.push(element[1]);
1333 }
1334 //control group of nonexistent states
1335 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1336 //check which states are not in the game,
1337 //and add them with 0 value
1338 var a = [];
1339 for (var i = 0; i < bundesland.length; i++) {
1340     a[bundesland[i]] = true;
1341 }
1342 for (var i = 0; i < herkunft.length; i++) {
1343     if (a[herkunft[i]]) {
1344         delete a[herkunft[i]];
1345     } else {
1346         a[herkunft[i]] = true;
1347     }
1348 }
1349 for (var k in a) {
1350     herkunft.push(k);
1351     user.push(0);
1352 }
1353 loadHerkunftTable(herkunft, user);
1354 herkunft = []; //reset to null again
1355 user = []; //reset to null again
1356
1357 //table variables
1358 var herkunft = [];
1359 var user = [];
1360
1361 for (let i = 0; i < sortable.length; i++) {
1362     const element = sortable[i];
1363     herkunft.push(element[0]);
1364     user.push(element[1]);
1365 }
1366 //control group of nonexistent states
1367 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1368 //check which states are not in the game,
1369 //and add them with 0 value
1370 var a = [];
1371 for (var i = 0; i < bundesland.length; i++) {
1372     a[bundesland[i]] = true;
1373 }
1374 for (var i = 0; i < herkunft.length; i++) {
1375     if (a[herkunft[i]]) {
1376         delete a[herkunft[i]];
1377     } else {
1378         a[herkunft[i]] = true;
1379     }
1380 }
1381 for (var k in a) {
1382     herkunft.push(k);
1383     user.push(0);
1384 }
1385 loadHerkunftTable(herkunft, user);
1386 herkunft = []; //reset to null again
1387 user = []; //reset to null again
1388
1389 //table variables
1390 var herkunft = [];
1391 var user = [];
1392
1393 for (let i = 0; i < sortable.length; i++) {
1394     const element = sortable[i];
1395     herkunft.push(element[0]);
1396     user.push(element[1]);
1397 }
1398 //control group of nonexistent states
1399 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1400 //check which states are not in the game,
1401 //and add them with 0 value
1402 var a = [];
1403 for (var i = 0; i < bundesland.length; i++) {
1404     a[bundesland[i]] = true;
1405 }
1406 for (var i = 0; i < herkunft.length; i++) {
1407     if (a[herkunft[i]]) {
1408         delete a[herkunft[i]];
1409     } else {
1410         a[herkunft[i]] = true;
1411     }
1412 }
1413 for (var k in a) {
1414     herkunft.push(k);
1415     user.push(0);
1416 }
1417 loadHerkunftTable(herkunft, user);
1418 herkunft = []; //reset to null again
1419 user = []; //reset to null again
1420
1421 //table variables
1422 var herkunft = [];
1423 var user = [];
1424
1425 for (let i = 0; i < sortable.length; i++) {
1426     const element = sortable[i];
1427     herkunft.push(element[0]);
1428     user.push(element[1]);
1429 }
1430 //control group of nonexistent states
1431 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1432 //check which states are not in the game,
1433 //and add them with 0 value
1434 var a = [];
1435 for (var i = 0; i < bundesland.length; i++) {
1436     a[bundesland[i]] = true;
1437 }
1438 for (var i = 0; i < herkunft.length; i++) {
1439     if (a[herkunft[i]]) {
1440         delete a[herkunft[i]];
1441     } else {
1442         a[herkunft[i]] = true;
1443     }
1444 }
1445 for (var k in a) {
1446     herkunft.push(k);
1447     user.push(0);
1448 }
1449 loadHerkunftTable(herkunft, user);
1450 herkunft = []; //reset to null again
1451 user = []; //reset to null again
1452
1453 //table variables
1454 var herkunft = [];
1455 var user = [];
1456
1457 for (let i = 0; i < sortable.length; i++) {
1458     const element = sortable[i];
1459     herkunft.push(element[0]);
1460     user.push(element[1]);
1461 }
1462 //control group of nonexistent states
1463 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1464 //check which states are not in the game,
1465 //and add them with 0 value
1466 var a = [];
1467 for (var i = 0; i < bundesland.length; i++) {
1468     a[bundesland[i]] = true;
1469 }
1470 for (var i = 0; i < herkunft.length; i++) {
1471     if (a[herkunft[i]]) {
1472         delete a[herkunft[i]];
1473     } else {
1474         a[herkunft[i]] = true;
1475     }
1476 }
1477 for (var k in a) {
1478     herkunft.push(k);
1479     user.push(0);
1480 }
1481 loadHerkunftTable(herkunft, user);
1482 herkunft = []; //reset to null again
1483 user = []; //reset to null again
1484
1485 //table variables
1486 var herkunft = [];
1487 var user = [];
1488
1489 for (let i = 0; i < sortable.length; i++) {
1490     const element = sortable[i];
1491     herkunft.push(element[0]);
1492     user.push(element[1]);
1493 }
1494 //control group of nonexistent states
1495 var bundesland = ["Bayern", "Berlin", "Brandenburg", ...];
1496 //check which states are not in the game,
1497 //and add them with 0 value
1498 var a = [];
1499 for (var i = 0; i < bundesland.length; i++) {
1500     a[bundesland[i]] = true;
1501 }
1502 for (var i = 0; i < herkunft.length; i++) {
1503     if (a[herkunft[i]]) {
1504         delete a[herkunft[i]];
1505     } else {
1506         a[herkunft[i]] = true;
1507     }
1508 }
1509 for (var k in a) {
1510     herkunft.push(k);
1511     user.push(0);
1512 }
1513 loadHerkunftTable(herkunft, user);
1514 herkunft = []; //reset to null again
1515 user = []; //reset to null again
1516
1517 //table variables
1518 var herkunft = [];
1519 var user = [];
1520
1521 for (let i = 0; i < sortable.length; i++) {
1522     const element = sortable[i];
1523     herkunft.push(element[0]);
1524     user.push(element[1]);
1525 }
1526 //control
```