/* Backend-Dokumentation «Covid» */

</Technologie-Stack>

//Frontend HMTL, CSS, JavaScript, D3.js

Das Frontend wird aus einem Zusammenspiel von HTML, CSS, JavaScript und der JavaScript-Library D3.js realisiert. JavaScript-Code hat den Vorteil, dass er in externe Skriptdateien ausgelagert werden kann, was dabei hilft, den Code schlank zu halten und trotzdem die vorteilhaften Funktionalitäten von JavaScript nutzen zu können. Diese Vorteile liegen insbesondere in der Flexibilität und Reaktionsfähigkeit bei Nutzer*innenaktionen, welche dadurch zustande kommt, dass JavaScript clientseitig läuft. JavaScript verleiht dem HTML-Dokument Dynamik und Interaktivität. Zur Visualisierung der Daten wird die JavaScript Open-Source-Bibliothek D3 eingesetzt. Dazu wird das Skript in den HTML-Code eingebunden.

//Backend Java-Applikation

Java enthält eine Bibliothek für WebSockets Clients und Server. Das WebSocket-Protokoll ist eine leichtgewichtige Alternative zu HTTP, weil es im Gegensatz dazu eine bidirektionale Kommunikation zwischen einer Webanwendung und dem Web-Socket-Server ermöglicht. Nach dem Verbindungsaufbau von Client und Server mittels einer klassischen Anfrage wie bei HTTP, bleibt der Kommunikationskanal danach geöffnet, wodurch es auch dem Server möglich ist, aktiv Informationen zur Verfügung zu stellen, ohne erneute Anfrage. Ein WebSocket eignet sich deshalb besonders dann, wenn viele kleinere Nachrichten zwischen Client und Server ausgetauscht werden müssen oder auch wenn eine Echtzeitverbindung von Vorteil ist. Deshalb eignet sich der Einsatz im Fall eines Online-Games besonders.

//Datenspeicher Bei diesem Online-Game wird der Server als Datenspeicher verwendet, auf welchem die Dateien abgelegt werden.

Im Falle des Einsatzes einer Datenbank, würde sich am ehesten eine relationale Datenbank anbieten, da sich die Daten aus den Spielen tabellarisch modellieren lassen und sich in ihrer Struktur auch nicht verändern, da bei jedem Spiel ein neuer Eintrag in der jeweiligen Tabelle hinzugefügt wird. Um die Statistik in diesem Game zu erheben, ist es deshalb wichtig, auf die Resultate vergangener Spiele zurückgreifen zu können. Eine populäre relationale Datenbank, v.a. auch für den Einsatz bei Webanwendungen, ist MySQL.

</Frontend-API>

/covid/games	{gamesid: integer, username: string, trackrecord: array [items: integer]} Requests: GET, POST, PUT
/covid/games/{gamesid}	{gamesid: integer} Requests: GET, PATCH
/covid/statistik	{statid: integer, statname: string, maincount: integer, (not required → att1-5: array [items: integer]} Requests: POST, PUT
/covid/statistik/{statid}	{statid: integer} Requests: GET, PATCH
/covid/results	{resultid: integer, resultname: string, resulttext: string} Requests: GET, POST, PUT
/covid/results/{resultid}	{resultid: integer} Requests: GET
/covid/fragen	{frageid: integer, fragetext: string, optnum: integer, opttexts: array [items: string], redirections: array [items: integer]} Requests: GET, POST, PUT
/covid/fragen/{frageid}	{frageid: integer} Requests: GET

</Datenbank>

Die Datenbank umfasst folgende Tabellen:

</Tabelle>	**</Inhalt>**
Games	gamesid ; username ; trackrecord
Statistik	statid ; statname ; maincount ; (not required att1-att5)
Results	resultid ; resultname ; resulttext
Fragen	frageid ; fragetext ; optnum ; opttexts ; redirections

</Backend>

Games

Über die API */covid/games* kann ein neues Spiel erstellt werden.

Im lokalen Speicher wird ein Eintrag mit folgenden Werten erstellt:

gamesid — Wird automatisch als fortlaufende Zahl generiert, sofern die *gamesid* noch nicht auf dem Server angelegt wurde. Falls bereits vorhanden, wird das auf dem Server abgelegte Spiel aufgerufen. Bei gewünschtem Restart, kann ein Spiel auch aus dem lokalen Speicher wieder gelöscht werden.

username — Wird gemeinsam mit der *gamesid* auf dem Server abgelegt und muss als Text erfasst werden. Ein Eintrag ist erforderlich, das Spiel wird ohne Eintrag nicht gestartet. Wird mit einer maximal Länge von 20 Zeichen in Klartext gespeichert.

trackrecord — Wird gemeinsam mit der *gamesid* und dem *username* auf dem Server abgelegt. Der Startwert ist immer 1 (erste Frage) und wird um die gewählten Antworten ergänzt. Dies erfolgt über eine Funktion, welche nach jeder Antwortauswahl ausgelöst wird und ein neues Element in das Array einfügt.

Die Daten werden jeweils in der lokalen Variable *currentgame* gespeichert und mittels einer fetch-Funktion werden somit Spielstand (*trackrecord*) und Spielstatistik (*scount*: *gestartete Spiele*) an den Server übermittelt. So können auch die Daten für ein bereits bestehendes Spiel vom Server abgerufen werden. Das Ganze ist synchronisiert mit der Visualisierung, welche das “Spielfeld” zeichnet und die Frage/Antworten darstellt. Die Daten eines Games (*gamesid*, *username*, *trackrecord*) werden ebenfalls eingeblendet.

Statistik

Über die API */covid/statistik* werden verschiedene Daten zu den erfolgten Nutzer*inneninteraktionen erhoben.

statid — Besteht aus einer Zahl, welche bereits vordefiniert und auf dem Server abgelegt ist. Wird während des Spiels nicht manipuliert. Anhand der *statid* kann die für die Datenvisualisierung gewünschte Substatistik vom Server abgerufen werden, gleichzeitig können so aber auch verschiedene Substatistiken anhand ihres *maincounts* verglichen werden.

statname — Besteht aus Text und ist ebenfalls bereits auf dem Server abgelegt. Wird während des Spiels nicht manipuliert.

maincount — Besteht aus einer Zahl, welche sich je nach *statid* bei jedem angefangenen oder abgeschlossenen Spiel und je nach erzieltem Resultat erhöht. Jedes Resultat (Pandemietyp) hat eine eigene *statid* erhalten.

Die bereits angelegten Statistiken werden während des Spiels mittels einer fetch-Funktion aktualisiert, indem deren *maincount* manipuliert wird

Results

Über die API */covid/results* werden während des Spiels die entsprechenden Inhalte, welche auf dem Server hinterlegt sind, abgerufen. So konnte der Code massiv schlanker gestaltet und es könnte nun einfach ein Quiz mit anderen Inhalten gestaltet werden.

resultid	Als Zahl definiert, anhand derer das entsprechende Resultat, nach Beenden des Quiz geladen werden kann. Je nach erreichtem Resultat, also *resultid*, wird die Statistik, also der *maincount*, des selbigen Resultats aktualisiert.
resultname	Hier ist die Bezeichnung des Typs als Text hinterlegt.
resulttext	Hier ist die Beschreibung des Typs als Text hinterlegt.

Bei der Ausgabe eines Resultats wird auch die Statistik (*maincount*) der abgeschlossenen Spiele (*ecount*) aktualisiert.

Fragen

Über die API */covid/fragen* werden während des Spiels die entsprechenden Frage-Antwort-Inhalte, welche auf dem Server hinterlegt sind, abgerufen. So konnte der Code massiv schlanker gestaltet und es könnte nun einfach ein Quiz mit anderen Inhalten gestaltet werden.

frageid	Als Zahl definiert, anhand derer alle benötigten Inhalte für einen Spielzug geladen werden können.
fragetext	Als Text definiert und wird als solcher vom Server abgerufen.
optnum	Als Zahl definiert und gibt die Anzahl möglicher Antworten pro Frage aus.
opttexts	Als Array definiert, welches Textelemente enthält und somit die Inhalte der Antwortmöglichkeiten ausgibt.
redirections	Als Array definiert, welches Zahlenelemente enthält und somit die passenden Weiterleitungen von der gewählten Antwort zur nächsten Frage/zum Resultat definiert.