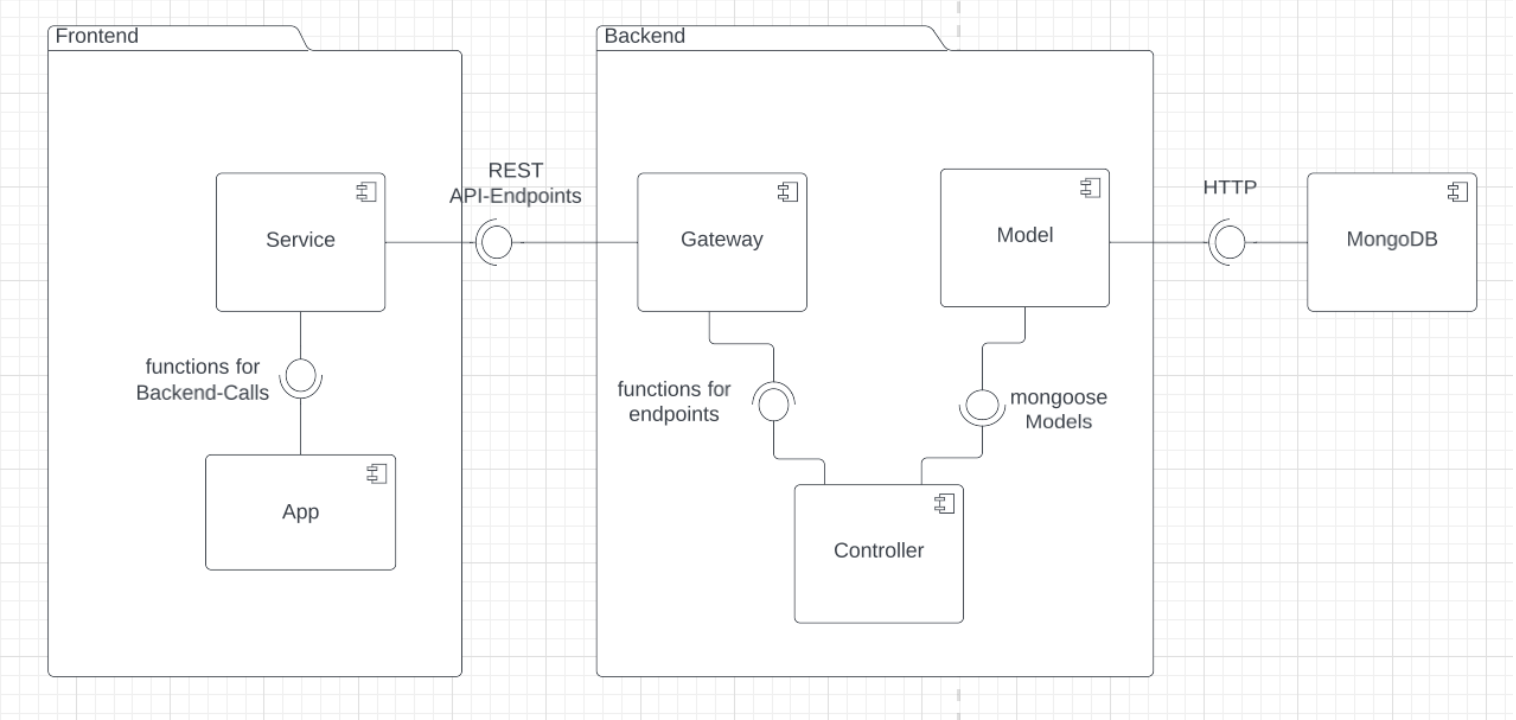
Be-Aware - Architektur

Binary Beasts

Komponenten

Ebene 1

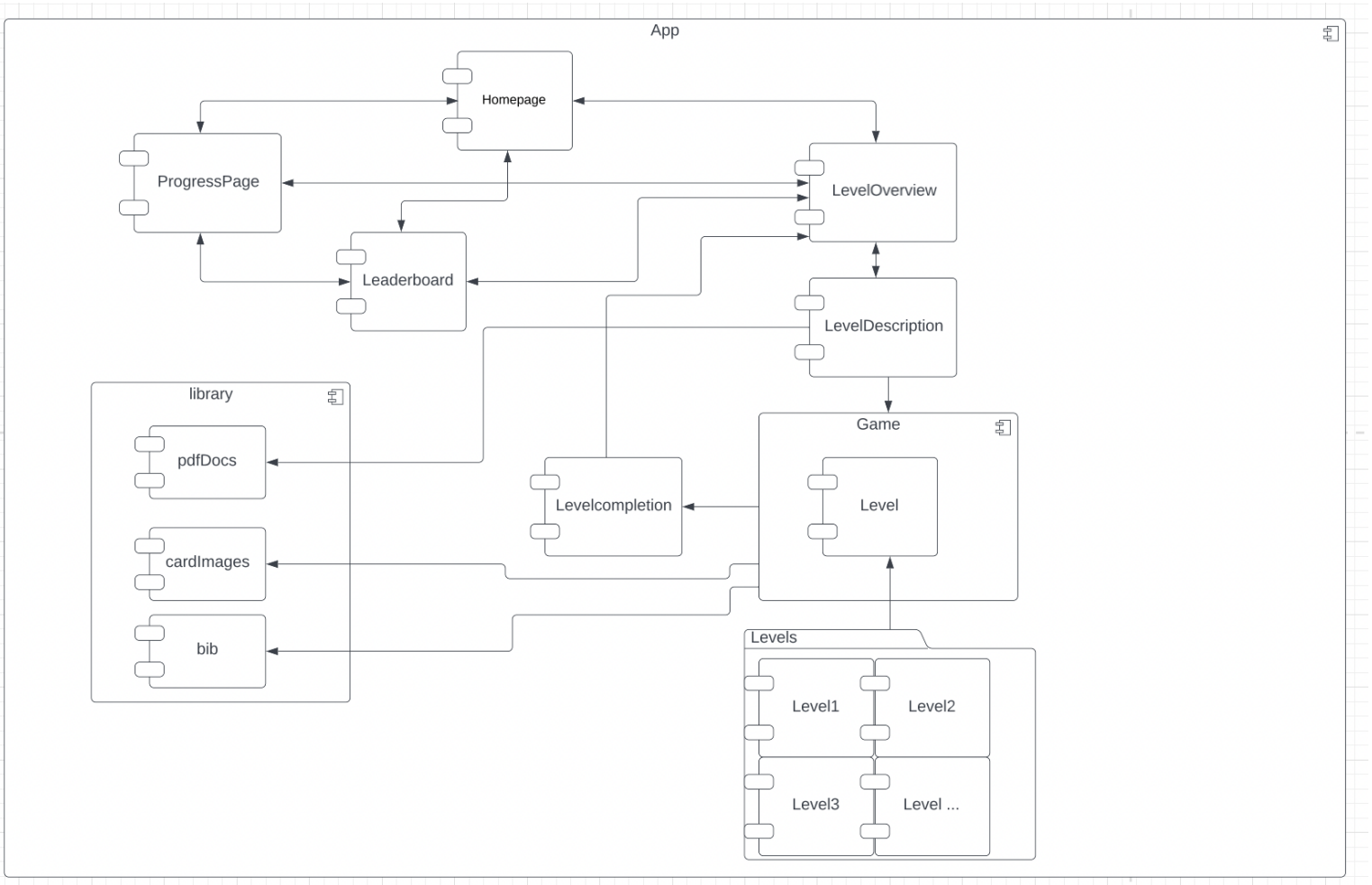


Auf der ersten Ebene läuft die Kommunikation der Komponenten folgendermaßen ab: Im Frontend liegt die App-Komponente, in der alle Seiten liegen (siehe Ebene 2). Diese Seiten benötigen für die dynamische Darstellung bestimmte Informationen aus dem Backend, welche über die Service-Komponente angefragt werden können. Die entsprechende Methode in der Service-Komponente führt dann den REST-Call an das Backend aus. Die Calls aus dem Frontend landen dann im Backend in der Gateway-Komponente, die in mehrere Komponenten untergliedert ist (siehe Ebene 2), je nach Art der Daten. Diese Endpoints aus dem Gateway rufen die entsprechende Funktion im Controller auf, welche wiederum Models für das Holen der Daten bzw. die Änderung von Daten verantwortlich sind.

Ebene 2

Auf der zweiten Ebene werden die Beziehungen zwischen den Komponenten innerhalb der App-, Gateway-, Controller- und Model-Komponente konkretisiert.

App-Komponente



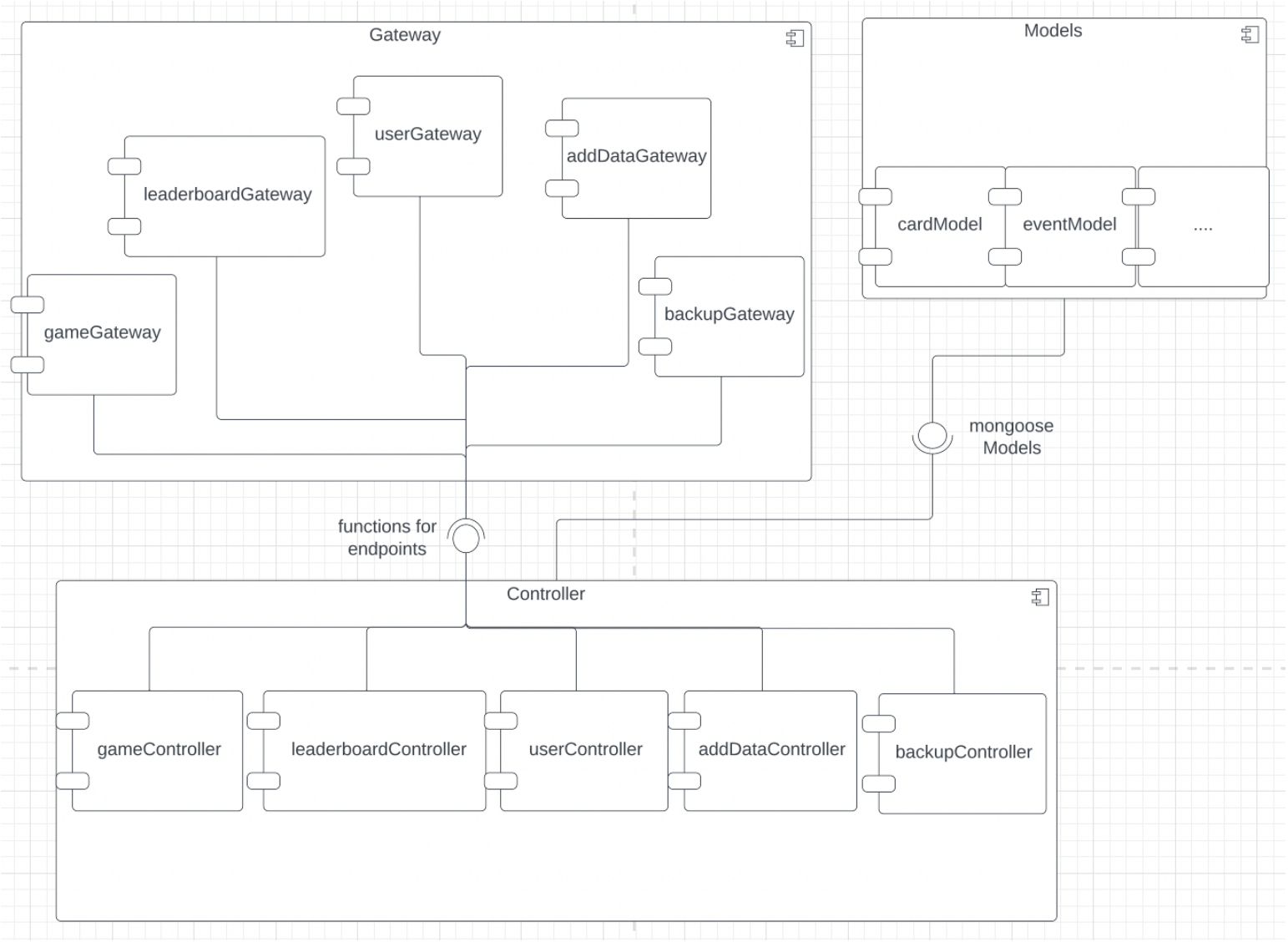
Die Pfeile zwischen den Komponenten, die eine Seite darstellen, kennzeichnen, dass man zwischen diesen Seiten navigieren kann. Für die Spiellogik ist der folgende Ablauf festgelegt: Man navigiert von einer beliebigen Seite auf die LevelOverview-Seite, in der man über Buttons die verschiedenen Levels auswählen kann. Klickt man auf einen Button für ein Level, wird man zu der LevelDescription-Seite weitergeleitet. Hier steht die Beschreibung des Levels, man kann sich das PDF-Dokument für die Lerninhalte runterladen, und man kann das Level starten.

“library”:
Für den PDF-Download wird ein Funktionsaufruf an die pdfDocs-Komponente ausgeführt, welche das jeweilige PDF-Dokument für das Level zurückliefert. Startet man das Level, so wird man zur Game-Komponente weitergeleitet.

“Game”:
Die Game-Komponente stellt fest, welches Level ausgewählt wurde und lädt die entsprechende Level-Komponente mit ihrer zugehörigen Spiellogik. Somit wird das Prinzip des Separation of Concerns sichergestellt, da die Game-Komponente universell für alle Level die Game-Navbar anzeigt (mit Level-Name, Punktzahl/Gesamtpunktzahl) und die Level-Komponente sich nur um die Karten, Ereignisse und Spiellogik kümmert. Hierbei wird

zur Kommunikation zwischen Game und Level (z.B. für die Punktzahl) die bib-Komponente verwendet, und für die Kartenbilder wird die cardImages-Komponente benutzt. Ist das Level abgeschlossen, navigiert man zur LevelCompletion-Komponente, in der der Nutzer Feedback erhält und zurück zur LevelOverview-Seite navigieren kann.

Backend-Komponenten



Die wesentlichen Gateways, die die Anwendung benötigt, sind für Game, Leaderboard und User. Die Gateways für addData und Backup sind für die Daten-Einpflegung bzw. Datensicherung. Die Gateways legen die API-Endpoints für das Frontend fest und bestimmen, welcher Endpoint welche Funktion in der Controller-Komponente aufruft.
Die Controller-Komponente ist hierbei analog zur Gateway-Komponente in mehreren Komponenten untergliedert. Die jeweilige Controller-Komponente legt schlussendlich fest, wie die Daten aus der Datenbank geholt und zurückgeliefert werden. Damit die Controller sich die Daten aus der Datenbank holen können und diese dann entsprechend im Code darstellen können, werden Mongoose-Models verwendet. Was genau Mongoose ist, wird im Abschnitt “Schnittstellen” näher behandelt. Es existiert in der Models-Komponente jeweils

ein Model für jede Collection in der MongoDB, die die jeweilige Struktur der Collection im Code darstellt. Zudem bieten die Models durch die Verwendung von Mongoose Schnittstellen an, mit denen man die Daten holen bzw. neue Daten in die MongoDB schreiben kann.

Schnittstellen

Komponente	**Daten**	**Funktionen**
Controller	Input: - aktuelles Level - Nickname - Punktzahl Output: - Level-Informationen (Ereignisse, Aktionen, …) - Leaderboard-Informationen	- *getLevel():* Rückgabe der Level-Informationen als JSON anhand der Level-Eingabe - *postLeaderboard():* Bei abgeschlossenem Level wird das Level mit den Meta-Daten wie z.B. Nickname und Punkte für das Leaderboard abgespeichert - …
Model	Input: - Nutzer-Eingaben Output: - Datenbank-Informationen	- Für jede Collection in der Datenbank gibt es ein Mongoose-Schema - Das Schema enthält alle Datenfelder der jeweiligen Collection - Im Mongoose-Schema sind die Funktionen für das Holen der Daten sowie das Schreiben der Daten in die Datenbank vordefiniert
Gateway	Input: - API-Aufruf mit Endpoint Output: - Weitergabe der Anfrage zum Controller für weitere Verarbeitung	bietet die Endpunkte für die Verarbeitung im Backend mit ihren entsprechenden HTTP-Verben (GET, POST, ..) an Bsp: *router.get('/getLevelData/:Level', controller.getLevelData, () => {...})*
Service	Input: - Nutzer-Eingaben Output: - Spielinformationen (Karten/Ereignisse) - User-Authentifikation	- Nutzer-Eingaben empfangen und mit entsprechenden API-Endpunkt an das Backend schicken - Vom Backend verarbeitete Spielinformationen an die React-Seiten zurückliefern

App	Input: - URL (mit Pfad zum Endpoint) Output: - React-Komponenten bzw. Seiten	bietet mithilfe von BrowserRouter-Tag die URL-Endpunkte zu den entsprechenden Seiten an Bsp: *<BrowserRouter> <Routers> <Route path="/" element={<Home />} /> …. </Routers> </BrowserRouter>*
Homepage	Input: - Nickname - Pin Output: - User-Zugang	*checkInput():* prüft, ob die Eingaben in der Login-Maske mit einem User in der DB übereinstimme, ansonsten wird ein neuer User angelegt
ProgressPage	Input: - Output: - Abgeschlossene Level - Punktzahl - Maximale Punktzahl	*calcProgress():* Berechnet die Prozentzahl der abgeschlossenen Level
Leaderboard	Input: - Output: - Nickname - Punktzahl insgesamt	holt sich alle DB-Einträge von der Leaderboard Collection und mappt diese auf eine Tabelle
LevelOverview	Input: - Levelauswahl Output: - Alle Level mit ihren Namen	Leitet je nach Levelauswahl den Nutzer auf die jeweilige Level-Beschreibungsseite weiter
LevelDescription	Input: - Nutzer-Auswahl (PDF-Download, Level Starten, Zurück Output: - PDF-Dokument	Leitet den Nutzer nach dem Levelstart zur Game-Komponente weiter
Game	Input: - Aktuelle Punktzahl - Vorherige Punktzahl - Levelname - Maximale Punktzahl Output: - Punktzahl - Levelname	- *passCurrentScore()* - *passPreviousScore()* - *passLevelname()* - *passMaxScore()* → Diese Funktionen werden vom Level aufgerufen, damit Game die spezifischen Level-Informationen in der Game-Navbar anzeigen kann
Level	Input: - Nutzer-Eingaben (Ausgewählte Aktionen) Output: - Reaktionen auf Aktionen	*handleAnswerButtonClick():* Erhält die Karte, die ausgewählt wurde, und ändert das Ereignis/ den Text je nach Levellogik

	(Punkte, Ereignisse)	*React.useEffect():* Rendert die Karten/Ereignisse bei Änderungen
LevelCompletion	Input: - Abgeschlossenes Level - Alle gespielten Karten Output: - Feedback	*useLocation():* über die useLocation übergibt das Level alle gespielten Karten mit dem jeweiligen Feedback und die Karteneigenschaft (gut/schlecht)
pdfDocs	Input: - Levelname Output: - PDF-Dokument mit Lerninhalten	*getPdfDoc():* Lierfert je nach Level das entsprechende Dokument
cardImages	Input: - Image-Value Output: - Kartenbild	*getCardImage():* Liefert je nach Image-Value (z.B. ‘network’) das jeweilige Bild für die Karte
bib	Input: - Nickname - Score Output: - Nickname - Score	*setNickname():* Setzt den aktuellen Nutzer-Nickname in den localStorage *getNickname()* *setLevelStartScore():* Setzt die Anfangspunktzahl in den localStorage *updateScore()* getScore() shuffle() Shuffelt die Karten zufällig

Protokolle

Angaben zum Protokoll der wichtigsten Komponenten und die Begründung für die Auswahl:

Komponente	**Protokoll**	**Begründung**
Controller	lokaler Methodenaufruf	Der Controller bietet Callback-Funktionen, die den Pfaden bzw. den Routern vom Gateway entsprechen, an, um Anfragen zu bearbeiten. Die Trennung zwischen dem Controller und dem Gateway ist ein gutes Designprinzip, um den Code kurz und lesbar zu halten.
Model	TCP/HTTP	Von Mongoose vorgegeben
Gateway	HTTP/REST	Gateway mit REST stellt Pfade (mit einem

		aussagekräftigen Namen) zum Controller zur Verfügung, empfängt clientseitige Anfragen, leitet sie an den entsprechenden Controller weiter und schließlich gibt dem Client die Antwort zurück.
Service	lokaler Methodenaufruf	Dadurch werden CRUD-Operationen basierend auf der Anfrage des Benutzers ans Backend gesendet und Antworten werden basierend auf dem Ergebnis an die entsprechende React-Komponente weitergegeben
ReactRouter	React-Komponenten Aufruf	React-Komponenten Aufruf bietet verschiedene Möglichkeiten an, wie das Spiel dargestellt werden kann, wie zum Beispiel: ● Konstruktor-Funktionen: Diese werden als erstes aufgerufen und durch sie können Eigenschaften von Komponenten initialisiert werden. ● Verschachtelung: Komponenten können innerhalb von anderen Komponenten verschachtelt werden ● Parameter: An sie können Parameter übergeben werden.
Homepage	React-Komponenten Aufruf	
ProgressPage	React-Komponenten Aufruf	
Leaderboard	React-Komponenten Aufruf	
LevelOverview	React-Komponenten Aufruf	
Game	React-Komponenten Aufruf	
Level	React-Komponenten Aufruf	
MongoDB	TCP/HTTP	Definiert von der externen Komponente (MongoDB)

Angaben zum Datenformat und Kommunikationsstil:

Komponente	**Datenformat**	**Kommunikationsstil**
Controller	JSON	Kommunikationsstil, der sich an Methodenaufrufen orientiert
Model	JSON	Von Mongoose vorgegeben
Gateway	JSON	REST
Service	JSON	Kommunikationsstil, der sich an Methodenaufrufen orientiert
ReactRouter	JSX (Javascript XML bzw. Javascript Syntax Extension)	Kommunikationsstil, der sich an React-Komponenten-Aufrufen orientiert
Homepage		
ProgressPage		
Leaderboard		

LevelOverview		
Game		
Level		
MongoDB	JSON	REST

Technologien

Für die Entwicklung der Webanwendung wurde der MERN-Stack verwendet. Dieser Stack setzt sich aus den folgenden Technologien zusammen:

M - MongoDB:
Für die Datenhaltung wurde die NoSQL-artige Datenbank MongoDB verwendet. Der Vorteil hierbei liegt darin, dass die Daten im JSON-Format gespeichert werden, welches ebenfalls von den folgenden Technologien verwendet. Somit müssen die Daten nicht umgeformt werden.

E - Express:
Express ist ein unterstützendes Framework für den NodeJS-Server, welches die Programmierung des Backends einfacher macht.

R - React:
Das React-Framework wird für das Frontend entwickelt. React bietet durch die Verwendung eines Virtual-DOMs eine sehr hohe Performanz und ermöglicht durch die Verwendung von JSX eine sehr flexible Programmierung. Dadurch ist es möglich, JavaScript Ausdrücke in geschweiften Klammern direkt in HTML zu verwenden.

N - NodeJS:
Das NodeJS-Framework bietet einen JavaScript-basierten Server, auf dem das Backend implementiert werden kann. Dadurch ist es möglich, für das Frontend eine REST-API zu entwickeln, um mit der Datenbank kommunizieren zu können.

Das Zusammenspiel dieser Technologien macht diesen Stack sehr mächtig und populär, da die Entwicklung einer Webanwendung hervorragend umzusetzen ist und richtig eingesetzt eine hohe Performanz bietet.

Angaben zu den eingesetzten Technologien die Begründung für die Auswahl:

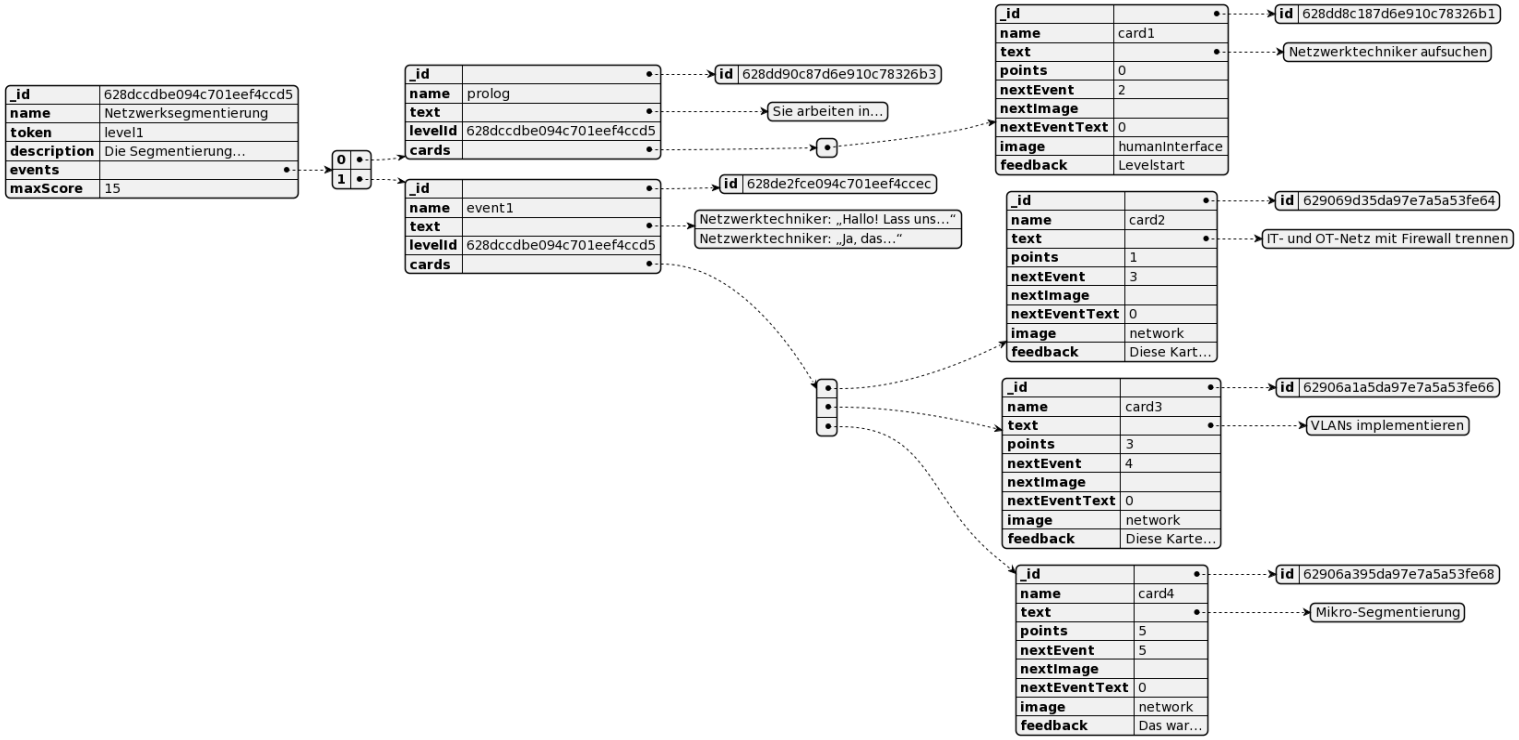
Komponente	**eingesetzte Technologien**	**Begründung**
Controller	Programmiersprache: JavaScript Ablaufumgebung: NodeJS Frameworks: Express	Mithilfe von Mongoose ist das Schreiben/Lesen der Daten durch den Controller einfach zu realisieren

Model	Programmiersprache: JavaScript Ablaufumgebung: NodeJS Frameworks: Express	Mithilfe von Mongoose ist die Darstellung der Daten im Code einfach zu realisieren
Gateway	Programmiersprache: JavaScript Ablaufumgebung: NodeJS Frameworks: Express	Express ermöglicht es, alle benötigten API-Endpoints in einer Datei zu definieren und durch den Node Server für das Frontend ansprechbar zu machen
Service	Programmiersprache: JavaScript Ablaufumgebung: Browser Frameworks: React	Datentransfer zwischen Front- und Backend erfolgt als JSON, da auf beiden Seiten JavaScript-basierte Frameworks verwendet werden, dadurch ist auf keiner Seite ein Parsen der Daten nötig
ReactRouter	Programmiersprache: JavaScript Ablaufumgebung: Browser Frameworks: React	Für das Routing der Seiten bietet sich der React-Router-Dom an, da hierdurch die Frontend-Endpunkte nur einmal in einer Datei definiert werden müssen und durch Link-Tags auf der gesamten Applikation ansprechbar sind
Homepage	Programmiersprache: JavaScript Ablaufumgebung: Browser Frameworks: React	Durch den React-Router-Dom wird die Homepage als Ausgangspunkt festgelegt (“/”-Pfad), durch den man dann auf die restlichen Seiten navigieren kann
ProgressPage	Programmiersprache: JavaScript Ablaufumgebung: Browser Frameworks: React	Die ProgessPage kann mit den oben beschriebenen Komponenten sehr einfach dynamisch mit den individuellen Daten dargestellt werden
Leaderboard	Programmiersprache: JavaScript Ablaufumgebung: Browser Frameworks: React	Auch das Leaderboard kann mit den oben beschriebenen Komponenten dynamisch mit den Daten dargestellt werden
LevelOverview	Programmiersprache: JavaScript Ablaufumgebung: Browser Frameworks: React	Mithilfe des React-Router-Dom lässt sich die LevelOverview-Seite sehr simpel mit den bereitgestellten Link-Tags realisieren
Game	Programmiersprache: JavaScript Ablaufumgebung: Browser Frameworks: React	Die gesamte Game-Komponente lässt sich durch React als Template für die Levels realisieren, in welchem die Levels nur reingeladen werden müssen → einheitliches Layout für alle Levels
Level	Programmiersprache: JavaScript	Das Level lässt sich durch React als

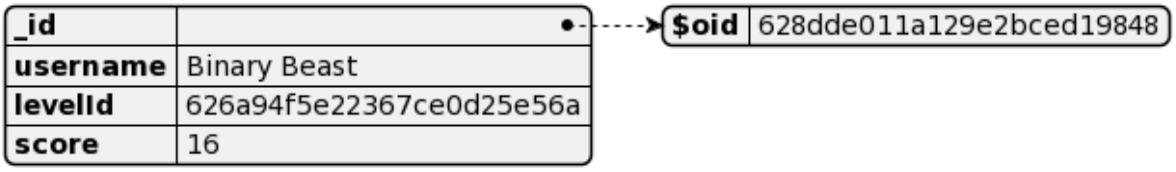
	Ablaufumgebung: Browser Frameworks: React	Komponente in die Game-Komponente integrieren
MongoDB	Ablaufumgebung: Cloud (Atlas)	Die MongoDB Datenbank ist Teil des populären “MERN-Stacks”, der sich zusätzlich aus den Frameworks Express, React und Node zusammensetzt. Das Zusammenspiel dieser Frameworks zusammen mit MongoDB bietet eine hervorragende Kombination, da die Daten zu jeder Zeit als JSON verarbeitet werden

6.3 Datenmodelle

Da die Daten in einer MongoDB gespeichert werden, liegen alle Daten in JSON vor. Zur Veranschaulichung der JSON-Struktur der Levels folgt ein Beispieldiagramm des ersten Levels mit den ersten zwei Ereignissen:

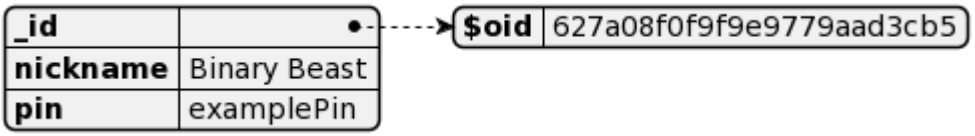


Die Leaderboard-Collection ist folgendermaßen aufgebaut:

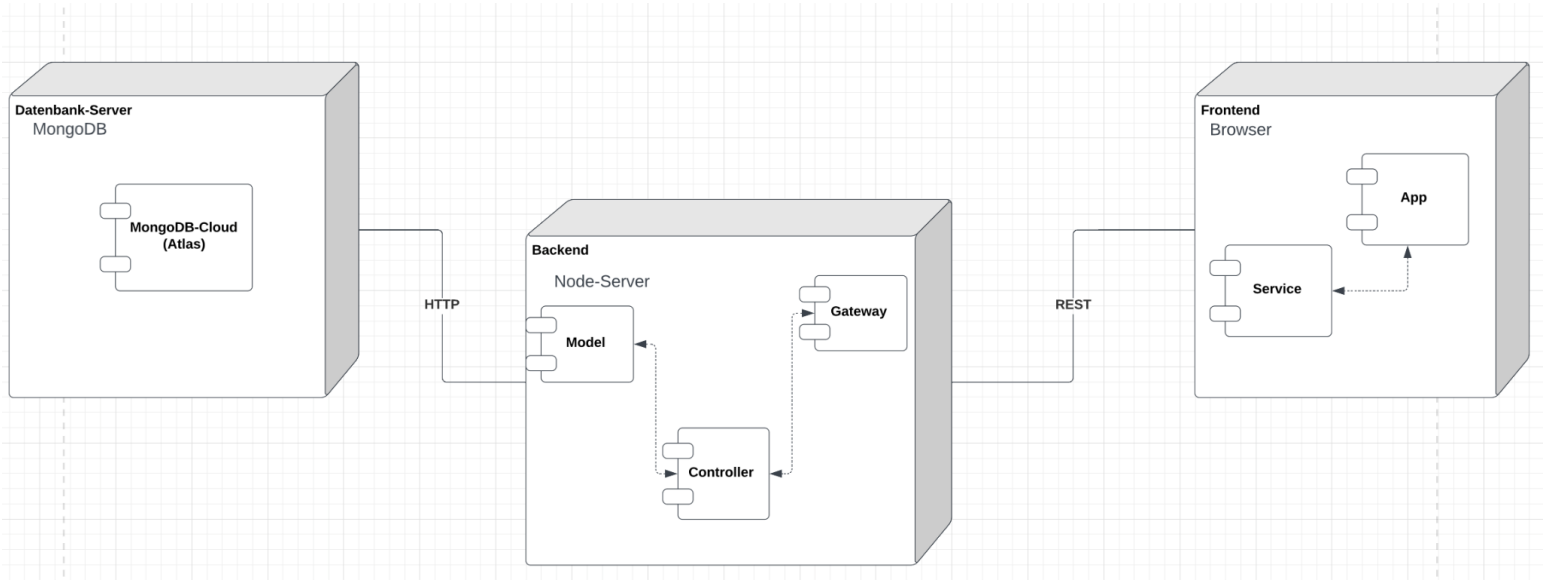


Die levelId referenziert hierbei das jeweilige Level, in der diese Punktzahl erreicht wurde. Die Collection wird sowohl für ProgressPage als auch für die Leaderboard-Seite verwendet. Bei der Leaderboard-Seite jedoch werden alle zusammengehörigen Einträge eines Nutzers gesammelt und nur die Summe der Punktzahl angezeigt.

Für den Nutzer wurde eine simple User-Collection verwendet:



Verteilungssicht



Für die Datenhaltung wird der MongoDB-Cloud-Dienst Atlas verwendet. Das Backend kann auf diesen mit den entsprechenden Zugangsdaten zugreifen. Das Backend läuft mit einem eigenem Server getrennt vom Frontend als REST-API. Diese REST-API kann das Frontend für ihre Funktionen aufrufen.

Gesamtüberblick - FMC-Diagramm

Das folgende Diagramm fasst den gesamten Ablauf der Software zusammen: