Design Entscheidungen

Entscheidung: NX

Das Projekt verwendet das NX-Entwicklungs-Framework für die Verwaltung des Monorepo, der Codegenerierung und der Build- und Test-Workflows.

Grund: Monorepo-Verwaltung

NX erleichtert die Verwaltung von Code in einem Monorepo, da es eine konsistente Struktur von Code bietet und Abhängigkeiten zwischen Projekten innerhalb des Monorepo verwaltet.

Grund: Codegenerierung

NX bietet Tools für die automatisierte Generierung von Code, einschließlich der Generierung von Scaffold für neue Komponenten, Services und Libraries.

Grund: Skalierbarkeit

NX ermöglicht es eine monolithische Architektur für das Projekt zu verwenden, indem es eine klare Struktur für Code und Abhängigkeiten bietet. Im Rahmen unseres Projekts beinhaltet das Monorepo sowohl das Frontend als auch das Backend der Anwendung, sowie E2E-Tests.

Entscheidung: Client Server

Die Client-Server-Architektur wird für die Implementierung der Anwendung verwendet.

Grund: Skalierbarkeit

Durch die Trennung der Anwendung in Frontend- und Backend-Komponenten kann jeder Teil der Anwendung unabhängig skaliert werden, um eine höhere Last zu bewältigen, ohne die Leistung des anderen Teils zu beeinträchtigen.

Grund: Sicherheit

Die Verwendung einer Client-Server-Architektur ermöglicht es, Sicherheitsmaßnahmen wie Zugriffskontrolle und Verschlüsselung an der Backend-Seite zu implementieren, um die Anwendung vor Angriffen zu schützen. Wir können somit beispielsweise verhindern, das der Nutzer das zu erratende Wort aus dem Client-Source-Code ausließt.

Grund: Wiederverwendbarkeit

Die Trennung der Anwendung in Frontend- und Backend-Komponenten ermöglicht es, die Wiederverwendung von Komponenten besser zu organisieren. In unserem Projekt werden Frontend-Komponenten nur im Frontend wiederverwendet und Backend-Komponenten nur im Backend wiederverwendet.

Entscheidung: RESTful-Webservices

Die Anwendung verwendet RESTful-Webservices zur Kommunikation zwischen dem Frontend und dem Backend.

Grund: Interoperabilität

RESTful-Webservices verwenden standardisierte HTTP-Methoden und Datenformate, was es verschiedenen Systemen und Programmiersprachen ermöglicht, miteinander zu kommunizieren. In unserem Projekt muss ein Qwik Frontend (geschrieben in TypeScript) mit einem Kotlin Backend kommunizieren.

Grund: Skalierbarkeit

RESTful-Webservices ermöglichen die Skalierung der Anwendung, indem sie das Frontend und das Backend entkoppeln und es dem Backend ermöglichen, mehrere Anfragen gleichzeitig zu verarbeiten. Unabhängig davon wie viele Clients das Frontend der Anwendung verwenden, kann das Backend die Anfragen verarbeiten.

Grund: Einfachheit

RESTful-Webservices sind einfach zu implementieren und zu nutzen, da sie auf standardisierten Methoden und Datenformaten basieren.

Entscheidung: PlantUML

Das Projekt verwendet PlantUML als Werkzeug für die Erstellung von UML-Diagrammen.

Grund: Einfache Syntax

PlantUML verwendet eine einfache, textbasierte Syntax, um UML-Diagramme zu erstellen.

Grund: Flexibilität

PlantUML unterstützt verschiedene Arten von UML-Diagrammen, einschließlich Klassendiagrammen, Sequenzdiagrammen und Zustandsdiagrammen, sowie

benutzerdefinierte Diagrammtypen. In diesem Projekt wird mit PlantUML die Sequenz- und Ablaufdiagrammen der Ablauf von Programm-Abschnitten beschrieben.

Grund: Integration

PlantUML kann leicht in andere Tools und Workflows integriert werden, z.B. in IDEs oder Dokumentationsprozesse.

Grund: Export

PlantUML bietet eine Vielzahl von Exportoptionen, um die erstellten Diagramme in verschiedenen Formaten zu exportieren, z.B. als Bild- oder PDF-Datei. In diesem Projekt werden die Diagramme als PNG-Bilder exportiert, welche in der Bibliotheken-Dokumentation referenziert werden.

Entscheidung: GitHub

Das Projekt verwendet GitHub als Code-Repository und Kollaborationsplattform für Issue-Tracking.

Grund: Kollaboration

GitHub bietet eine benutzerfreundliche und intuitive Plattform für die Zusammenarbeit zwischen Entwicklern, die es einfach macht, Code zu teilen, zu kommentieren und zusammenzuarbeiten.

Grund: Versionskontrolle

GitHub bietet eine leistungsfähige Versionskontrollfunktion, die es ermöglicht, verschiedene Versionen des Codes zu speichern und bei Bedarf wiederherzustellen.

Grund: Issue-Tracking

GitHub bietet eine integrierte Issue-Tracking-Funktion, die es Entwicklern ermöglicht, Issues zu erstellen, zu kommentieren und zu verfolgen.

Entscheidung: Tailwind CSS

Das Projekt verwendet das Tailwind CSS Framework für die Gestaltung des Frontends. Tailwind CSS ist ein Utility-First CSS-Framework, das es Entwicklern ermöglicht, responsive Benutzeroberflächen zu erstellen.

Grund: Effizienz

Tailwind CSS bietet eine Vielzahl von vordefinierten Utility-Klassen.

Grund: Flexibilität

Tailwind CSS ist sehr flexibel und erlaubt es Designs zu erstellen, ohne aufwendige CSS-Regeln schreiben zu müssen.

Grund: Responsivität

Tailwind CSS verfolgt den Mobile-First Ansatz und erleichtert die Erstellung von responsiven Designs für verschiedene Geräte.

Grund: Design

Einheitliches Design für alle Frontend-Komponenten der Wordle Time Plattform.

Entscheidung: Cypress

Das Projekt verwendet das Cypress-Testing-Framework für die Automatisierung von E2E-Tests im Frontend.

Grund: Einfache Einrichtung

Cypress ist einfach einzurichten und zu verwenden, da es kaum zusätzlichen Abhängigkeiten oder Konfigurationen erfordert.

Grund: Dokumentation

Im Verlauf von Cypress Tests kann das Framework Screenshots und Videos erstellen, die der Dokumentation der Tests dienen. Außerdem wird eine bessere Nachvollziehbarkeit des Anwendungsverhaltens bezüglich der in UML Aktivitätsdiagramm beschriebenen Abläufen ermöglicht.

Entscheidung: Single-Page-Anwendung

Das Projekt implementiert eine Single-Page-Anwendungen (SPA) - Architektur für das Frontend.

Grund: Trennung des Frontends vom Backend

Wartbarkeit erhöht und die Entwicklung erleichtert. Die SPA kann unabhängig vom Backend entwickelt werden, solange die Schnittstellen zwischen Frontend und Backend definiert sind. In diesem Projekt werden die Schnittstellen mit Swagger definiert.

Grund: Verbesserte Benutzererfahrung

Laden und Navigieren zwischen den Seiten nicht zu vollständigen Neuladungen führt, ist die Anwendung reaktionsschneller und bietet eine flüssige Benutzererfahrung.

Entscheidung: Kotlin

Das Projekt verwendet Kotlin als die Programmiersprache für die Entwicklung von dem Backend. Kotlin ist eine moderne, statisch typisierte Programmiersprache, die auf der JVM (Java Virtual Machine) ausgeführt wird und vollständig interoperabel mit Java ist.

Grund: Verbesserte Lesbarkeit und Produktivität

Kotlin bietet eine klare und präzise Syntax, die die Lesbarkeit des Codes verbessert und die Produktivität der Entwickler steigert. Dies führt zu kürzerem und wartbarem Code.

Grund: Sicherheit und Stabilität

Kotlin bietet Funktionen wie Typinferenz und Nullsicherheit, die dazu beitragen, Laufzeitfehler zu reduzieren und die Stabilität der Anwendung zu verbessern.

Grund: Moderne Sprachmerkmale

Kotlin bietet moderne Sprachmerkmale wie Funktionen höherer Ordnung, Datenklassen und Erweiterungsfunktionen, die die Entwicklung erleichtern und den Code eleganter gestalten.

Entscheidung: Ktor Server

Ktor ist Framework für die Entwicklung asynchroner Server- und Client Anwendungen. Im Projekt wir das Framework für die Backend-Entwicklung verwendet.

Grund: Asynchronität

Ktor nutzt die Vorteile der asynchronen Programmierung mit Kotlin-Coroutines, um eine hohe Leistung und Skalierbarkeit zu gewährleisten.

Grund: Fokus auf Server-Framework

Ktor bietet in diesem Projekt nur das Framework für einen Web-Server und konzentriert sich dabei auf seine Stärken. Freiheiten in anderen Bereichen der Backend-Entwicklung werden nicht eingeschränkt.

Grund: Nach Bedarf erweiterbar mit Plug-Ins

Der geringe Memory-Footprint des Ktor Kern-Frameworks wird u. A. dadurch ermöglicht, das viele Funktionen bei Bedarf via Plugins nachgerüstet werden. So werden im Rahmen dieses Projekts beispielsweise CORS-Funktionalität, Serialisierung und die Netty-Server-Engine via Plugins ins Backend integriert.

Entscheidung: Gradle

Das Projekt verwendet das Build-Management-Tool Gradle zur Automatisierung des Build-Prozesses, zur Verwaltung von Abhängigkeiten und zur Konfiguration der Projektabläufe. Gradle ist ein leistungsstarkes, flexibles und plattformunabhängiges Build-Tool, das auf der Groovy- und Kotlin-Programmiersprache basiert und sowohl für Java- als auch für andere JVM-basierte Sprachen eingesetzt werden kann.

Grund: Flexibilität

Gradle ist äußerst flexibel und erlaubt die Definition von benutzerdefinierten Build-Skripten und Aufgaben. Dies ermöglicht es, den Build-Prozess genau an die Anforderungen des Projekts anzupassen.

Grund: Erweiterbarkeit

Gradle bietet eine Vielzahl von Plugins und Erweiterungen, die den Build-Prozess um zusätzliche Funktionen erweitern können. Dies erleichtert die Integration von Tools für Tests, statische Code-Analyse, Berichterstellung und mehr.

Grund: Plattformunabhängigkeit

Gradle ist plattformunabhängig und kann auf verschiedenen Betriebssystemen eingesetzt werden, einschließlich Windows, macOS und Linux.

Entscheidung: Qwik

Das Projekt verwendet Qwik als Webframework für die Entwicklung des Frontends. Qwik ist ein modernes, komponentenbasiertes Webframework, das auf dem Server-Side Rendering (SSR) Ansatz basiert und JavaScript erst auf dem Client lädt, wenn es benötigt wird.

Grund: Performance

Qwik bietet eine schnellere initiale Seitendarstellung, da der Client sofort vollständig gerenderte HTML-Seiten erhält. Dies verbessert die Benutzererfahrung, insbesondere auf langsamen oder instabilen Internetverbindungen.

Grund: Wiederverwendbarkeit

Qwik ermöglicht die Wiederverwendung von Komponenten, was die Entwicklung beschleunigt und die Wartbarkeit des Codes verbessert.

Grund: Resumability

Qwik bietet eine Resumability-Funktion, die es ermöglicht, den Zustand der Anwendung zwischen den Seitenaufrufen beizubehalten. Dies verbessert die Benutzererfahrung, da die Anwendung nicht jedes Mal neu geladen werden muss.

Entscheidung: PNPM

Das Projekt verwendet PNPM als Paketmanager für die Verwaltung von Abhängigkeiten und Paketen. PNPM ist ein alternatives Paketverwaltungstool, das speziell für die effiziente Verwaltung von JavaScript-Paketen und -Abhängigkeiten entwickelt wurde. Im Gegensatz zu anderen Paketmanagern verwendet PNPM einen link-basierten Ansatz, der Speicherplatz spart und die Installation von Paketen beschleunigt.

Grund: Effizienz

PNPM verwendet einen symbolischen Link-Ansatz, bei dem gemeinsame Abhängigkeiten zwischen Projekten nicht mehrfach heruntergeladen werden müssen. Dies spart Speicherplatz und beschleunigt die Installation von Paketen erheblich.

Grund: Plattformunabhängigkeit

PNPM ist plattformunabhängig und kann in verschiedenen Betriebssystemen und Entwicklungsumgebungen eingesetzt werden, was die Entwicklungsarbeit erleichtert.

Grund: Konsistenz

PNPM sorgt für konsistente Paketversionen und eine saubere Projektstruktur, da es alle Abhängigkeiten in einem zentralen Speicherort verwaltet.

Entscheidung: Swagger

Das Projekt verwendet Swagger, ein Open-Source-Toolset, um API-Dokumentation zu erstellen, zu veröffentlichen und zu verwalten. Swagger bietet eine automatisierte Möglichkeit, API-Endpunkte, Parameter, Anfragen, Antworten und Authentifizierungsinformationen in einem einheitlichen und interaktiven Format zu dokumentieren.

Grund: Automatisierte Dokumentation

Swagger ermöglicht es, API-Dokumentation direkt aus dem Quellcode zu generieren, was Zeit spart und die Genauigkeit der Dokumentation erhöht. Entwickler müssen die Dokumentation nicht manuell pflegen, da sie automatisch mit Änderungen am Code aktualisiert wird.

Grund: Interaktive Dokumentation

Swagger generiert eine interaktive API-Dokumentationsseite, auf der Entwickler API-Endpunkte erkunden, Testanfragen senden und Beispiele für Anfragen und Antworten anzeigen können. Dies erleichtert die Nutzung der API und verkürzt die Einarbeitungszeit für Entwickler.

Grund: Konsistenz

Swagger stellt sicher, dass die API-Dokumentation konsistent und gut strukturiert ist, da sie automatisch aus dem Quellcode generiert wird. Dies trägt dazu bei, Verwirrung und Missverständnisse zu vermeiden.

Entscheidung: JSON

Die Anwendung verwendet JSON (JavaScript Object Notation) als das bevorzugte Datenformat für die Kommunikation zwischen Client und Server sowie für die Speicherung und den Austausch von Daten. JSON ist ein leichtgewichtiges und einfach zu lesendes Datenformat, das auf Schlüssel-Wert-Paaren und Hierarchien von Objekten und Arrays basiert.

Grund: Lesbarkeit und Verständlichkeit

JSON ist leicht lesbar und verständlich, sowohl für Menschen als auch für Maschinen. Dies erleichtert die Fehlersuche und das Debugging während der Entwicklung.

Grund: Plattformunabhängigkeit

JSON ist plattformunabhängig und kann von verschiedenen Programmiersprachen und Plattformen verarbeitet werden. Dies ermöglicht die Interoperabilität zwischen verschiedenen Teilen der Anwendung (Backend und Frontend).

Grund: Breite Unterstützung

JSON wird von den meisten Programmiersprachen und Entwicklungstools unterstützt. Es stehen zahlreiche Bibliotheken und Parser zur Verfügung, um die Verarbeitung von JSON-Daten zu erleichtern.

Entscheidung: LocalStorage

Die Anwendung verwendet die LocalStorage-API, um Daten lokal im Webbrowser des Benutzers zu speichern und abzurufen. LocalStorage ist eine Web-Storage-Technologie, die es ermöglicht, Daten in Form von Schlüssel-Wert-Paaren clientseitig zu speichern, sodass sie zwischen den Sitzungen erhalten bleiben.

Grund: Persistenz

Mit LocalStorage können Daten dauerhaft im Browser des Benutzers gespeichert werden. Dies ermöglicht es, Informationen zwischen verschiedenen Sitzungen und Seiten beizubehalten, ohne sie auf dem Server speichern zu müssen.

Grund: Einfache Implementierung

Die Verwendung von LocalStorage ist einfach und erfordert nur JavaScript-Kenntnisse. Es gibt keine Notwendigkeit für komplexes Server-Setup oder Datenbank-Management.

Grund: Clientseitige Daten

LocalStorage ist clientseitig, was bedeutet, dass die Daten nicht über das Netzwerk übertragen werden müssen. Dies trägt zur Reduzierung des Datenverkehrs bei und kann die Ladezeiten der Anwendung verbessern.

Entscheidung: Cookie

In diesem Projekt werden Cookies verwendet, um Benutzerinformationen auf dem Client-Gerät zu speichern. Cookies sind kleine Textdateien, die vom Webserver auf dem Client-Gerät des Benutzers gespeichert werden und Informationen über die Benutzersitzung oder Präferenzen enthalten können.

Grund: Sitzungsverwaltung

Cookies können verwendet werden, um Sitzungsinformationen zu speichern, sodass Benutzerdaten während einer Sitzung auf einer Website aufrechterhalten werden können.