Wordle Time

Kay Fischbach

Julian Türner

Gliederung

Aufbau

- GitHub Issues, Zeitliche Planung
- NX Monorepo
- Architektur

Frontend

- Besonderheiten von Qwik
- Spekulatives Module Fetching

Backend

- Webserver
- OpenApi/Swagger
- CORS

Testing

- Cypress
- JUnit

Planung des Projekts

Github Projekt & Github Issues

Erstellung von Funktionalen und nicht funktionalen Anforderungen

Anforderungen in User Stories mit Abnahme Kriterien gegossen

User Stories als Issues in GitHub angelegt und Veranwortlichkeiten, Priorität, Start, Label sowie Meilenstein zugewiesen

Jedes Issue wurde vom Team zusammen abgenommen und mit einem Test verifizert

Meilensteine

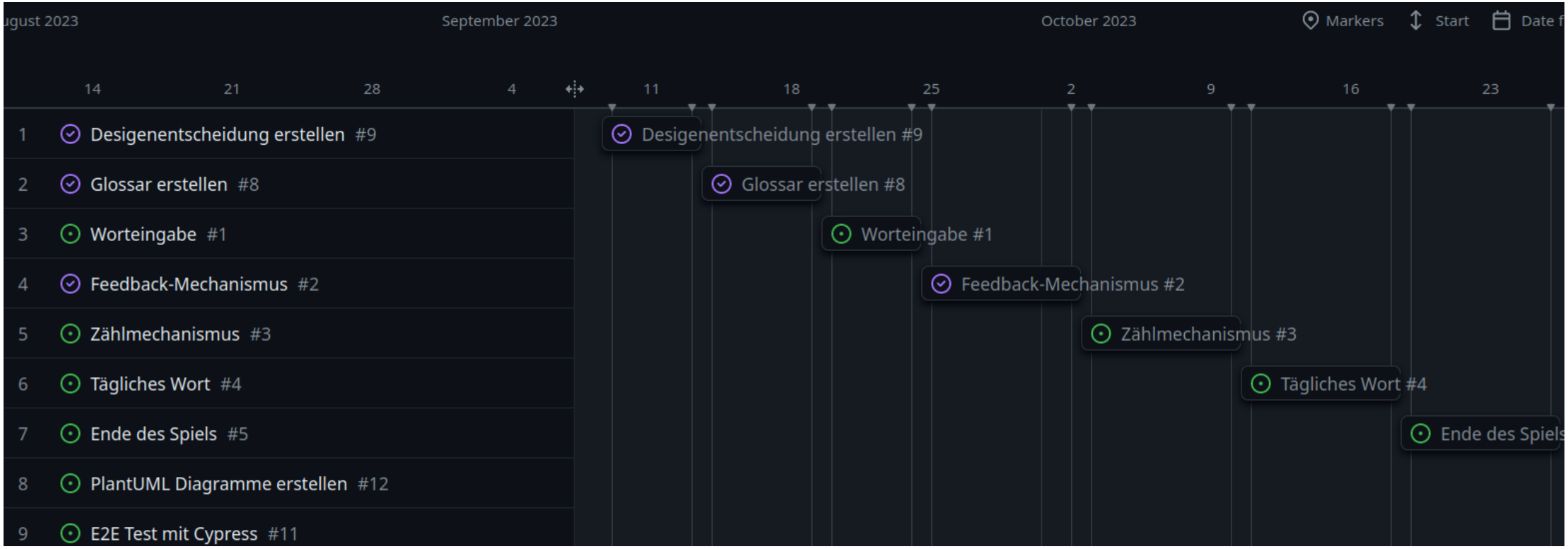
Entwicklung Wordle-Time -7 Issues

Dokumentation - 5 Issues

Einblick in die Projektplanung

Title	Status	Priority	Milestone	Start	Labels
Feedback-Mechanismus #2	Done	High	Entwicklung Wordle-Time	Oct 2, 2023	enhancement
Worteingabe #1	Todo	High	Entwicklung Wordle-Time	Sep 24, 2023	enhancement
Zählmechanismus #3	Todo	High	Entwicklung Wordle-Time	Oct 10, 2023	enhancement
Ende des Spiels #5	Todo	High	Entwicklung Wordle-Time	Oct 26, 2023	enhancement
Glossar erstellen #8	Done	High	Dokumentation	Sep 19, 2023	documentation
Desigenentscheidung erstellen #9	Done	High	Dokumentation	Sep 13, 2023	documentation
Markdown zu PDF konvertieren #10	Todo	High	Dokumentation	Jan 26, 2024	documentation
E2E Test mit Cypress #11	Todo	High	Dokumentation	Jan 21, 2024	documentation
PlantUML Diagramme erstellen #12	Todo	High	Dokumentation	Dec 31, 2023	documentation

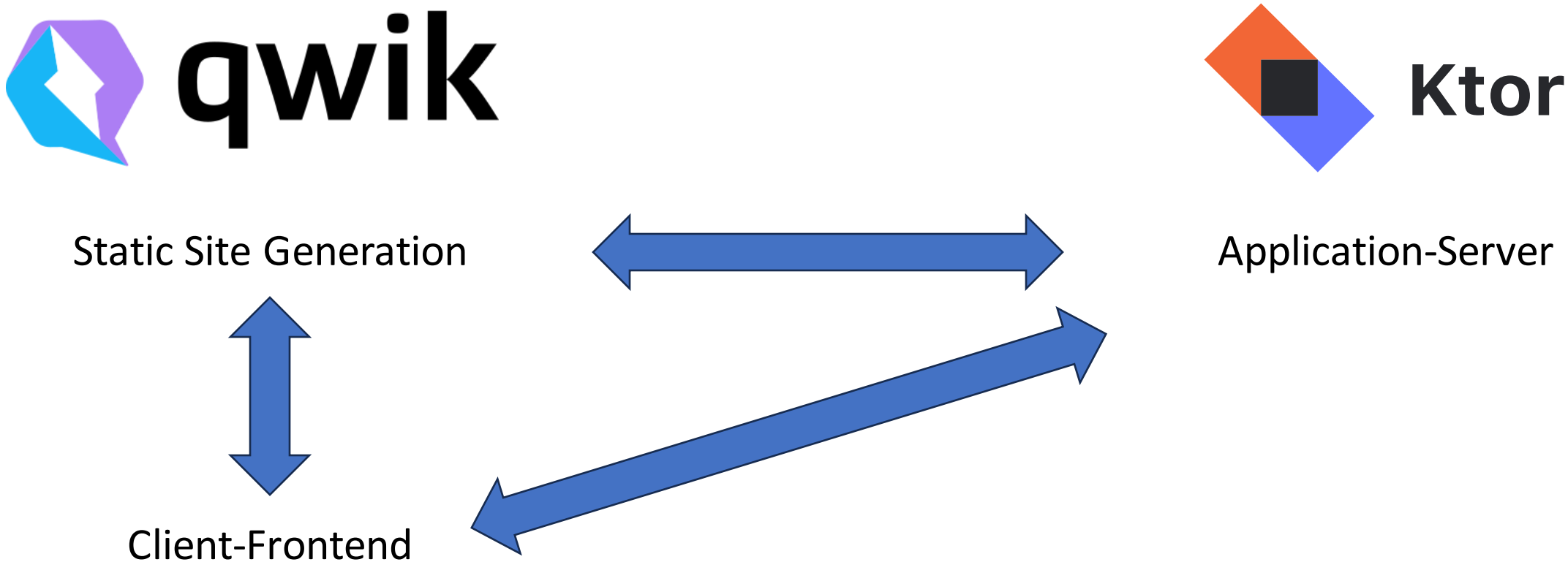
Zeitlicher Überblick der Issues



NX Monorepo

- Gemeinsames Repository indem alle Sachen gespeichert sind
- Tasks abstrahieren Projekteigenschaften
- Generiert eindeutige, nach einem Schema definierte Komponenten
- Kann Tasks in verschieden Konfigurationen ausführen

Architektur



Was ist Qwik?

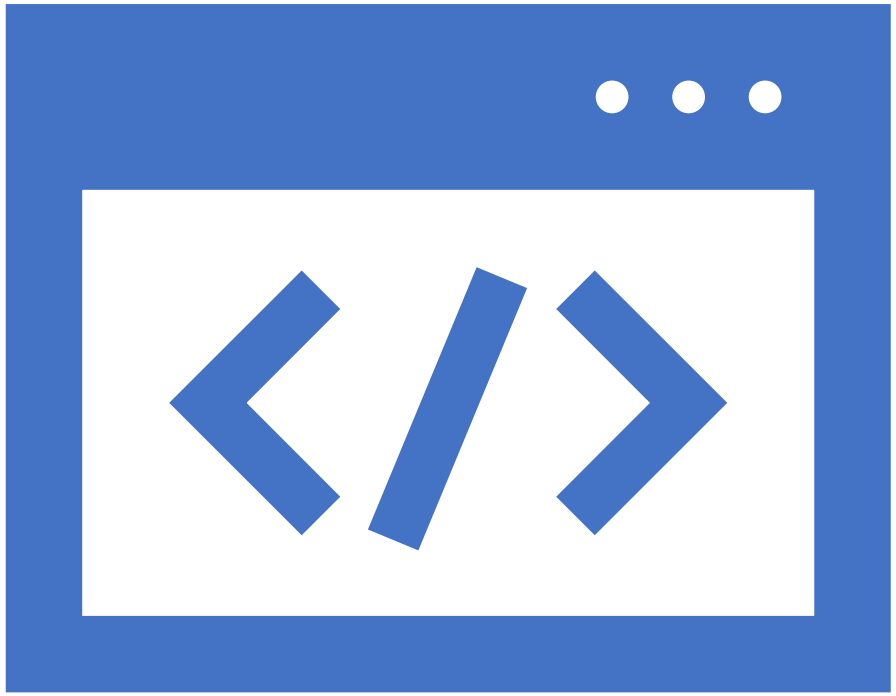
- Web-Framework für interaktive Anwendungen mit Fokus auf Komponentenbäume
- Hauptziel liegt in Instant-On Anwendungen
- Strategie:
  - Verzögerung der JavaScript-Ausführung und des Downloads so lange wie möglich
  - Auf dem Server erfolgt die Serialisierung des Ausführungszustandes
  - Fortsetzung folgt auf dem Client

Qwiks technische Ansätze

- Minimaler JavaScript-Code bei Anwendungsstart -> ca. 1KB JavaScript für Interaktivität
- Fortsetzung der Ausführung, wo der Server aufgehört hat
- Serialisierung von Listernen, internen Datenstrukturen und Anwendungsstatus

Problemstellung:

Viel JavaScript führt zu Problemen bei Bandbreite und Startzeit



Qwiks Vision und Lösungsansatz

Herausforderung: Weniger JavaScript versenden

Integration der Lazy-Loading-Philosophie auf tiefgreifende Weise.

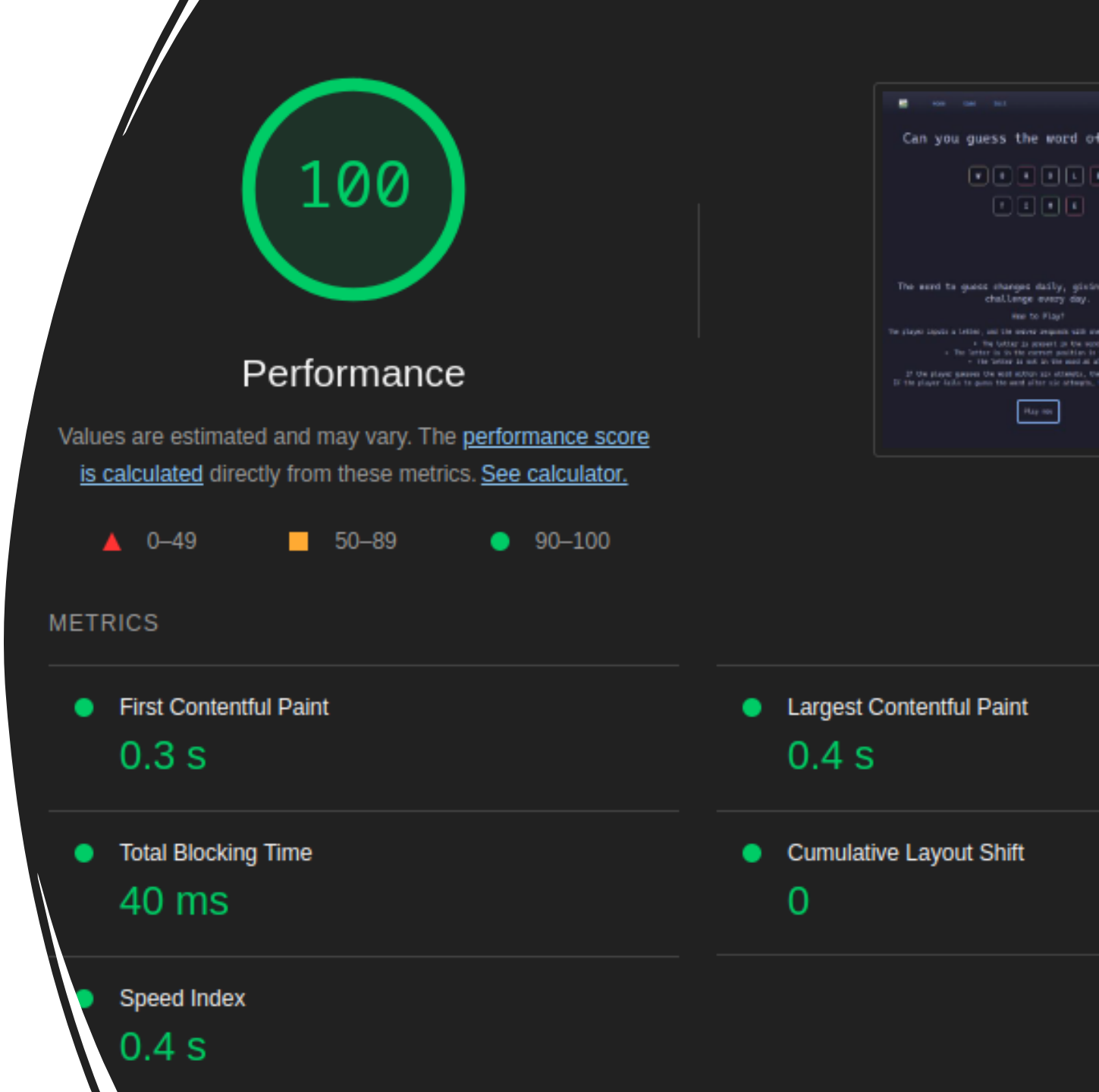
Kleine Bundle-Größen als primäres Designziel

Spekulatives Module Fetching

- Seiten schnell laden und interaktiv werden, ohne sofort JavaScript zu verwenden
- Spekulatives Vorausladen von Modulen in einem Hintergrundthread
- Vorabfüllen des Browser-Caches mit einem Service Worker
- Parallelisieren von Netzwerkanfragen, um Leistung zu optimieren -
- Vermeiden von doppelten Anfragen durch Cache-Verwaltung
- HTTP-Cache vs. Service Worker Cache wird auf Unterschiedlichen Caching-Ebenen beachtet

Ergebnis

- Wordle Time erzielt sehr gute Ergebnisse bei Google Lighthouse



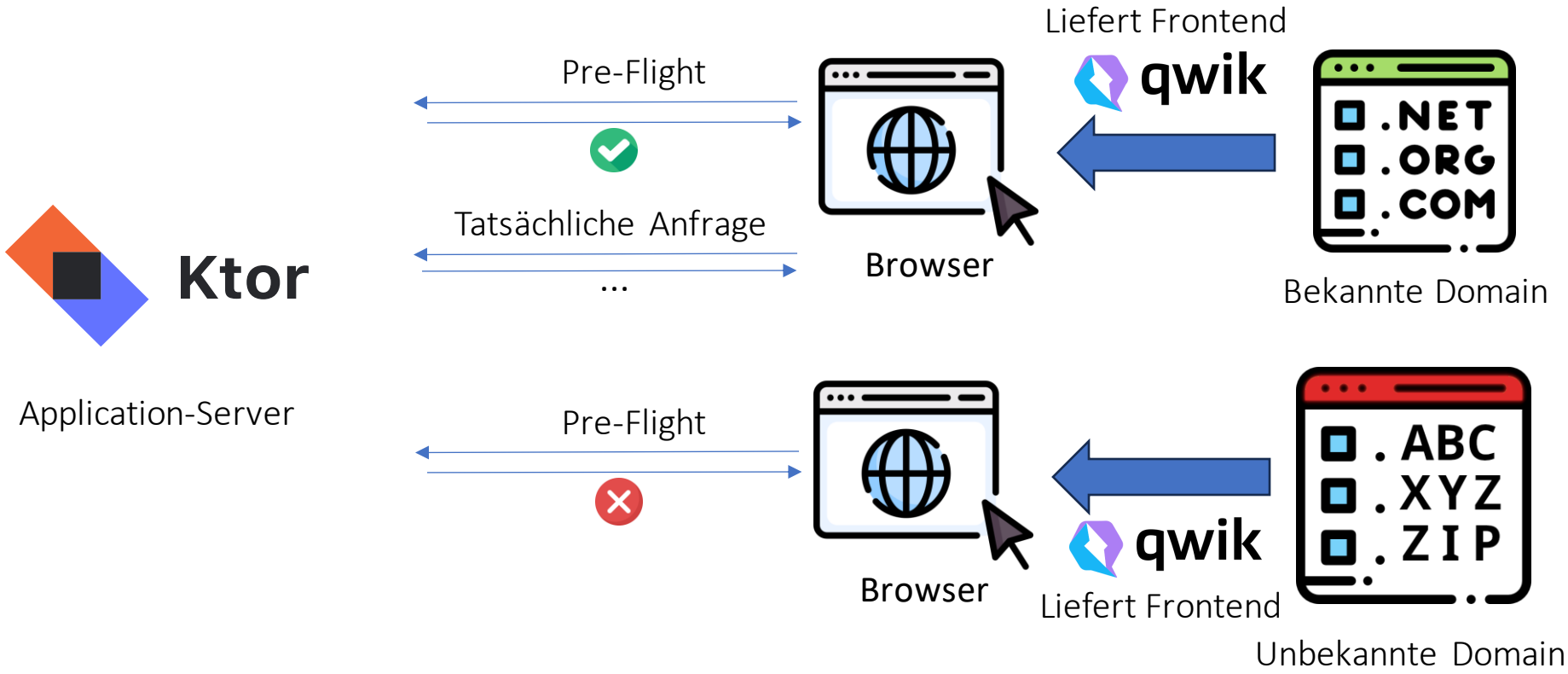
Ktor Application Server

Ktor

- Backend in Kotlin programmiert
- Ktor Server für asynchrone Anfragen-Beantwortung
- Lose-gekoppelter Aufbau der Server-Anwendung mittels Dependency-Injection
- Verifizierung der Frontend Authentizität mittels CORS

```
private fun Route.apiGuessCurrentGameID() {
  get<API.Guess.CurrentGameID> {
    val wordState: WordState by closestDI().instance()
    call.respond(wordState.currentWordContainer().stripWord())
  }
}
```

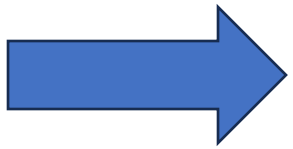
CORS – Schutz


Liefert Frontend
qwik
Pre-Flight
.NET
.ORG
.COM
Tatsächliche Anfrage
...
Ktor
Browser
Bekannte Domain
Application-Server
Pre-Flight
.ABC
.XYZ
.ZIP
qwik
Browser
Liefert Frontend
Unbekannte Domain

OpenAPI / Swagger

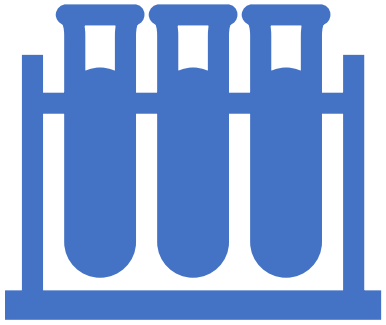
- Koordination des Routings zwischen Frontend- und Backend-Entwicklern
- Genaue Dokumentation aller Routen Anfragen und Antworten
- Spezifikation von obligatorischen und optionalen Parametern

```yaml
paths:
  /api/guess/word:
    get:
      summary: 'Guess the current wordl of the day'
      tags:
        - guess
      parameters:
        - name: 'gameID'
          in: 'cookie'
          required: false
          schema:
            type: 'string'
        - name: 'word'
          in: 'query'
          required: true
          schema:
            type: 'string'
```



Testing

Cypress -> End to End



Junit -> Unit Tests

Cypress

- Testumgebung für Webanwendungen
- Läuft im gleichen Laufzyklus wie die Anwendung
- Node-Serverprozess im Hintergrund mit Echtzeitreaktionen
- Native Zugriffsmöglichkeiten der Objekte
- Vollständige Kontrolle über Anwendung und Netzwerkverkehr
- Ermöglicht künstliche Erstellung von Testzuständen