

**Brainwavebandits**

**Welcher Wein zu meinem Menu?**

**Technische Informationen für die Jury**

# Technische Informationen für die Jury: BÄRNHÄCKT 2024

## Aktueller Stand des Sourcecodes

### Link zu GitHub Repository

[GitHub Repository Link](https://github.com/DresBaumann/Brainwave-Bandits)

## Ausgangslage

### Worauf habt ihr euch fokussiert?

Unser Hauptfokus lag auf der Entwicklung einer benutzerfreundlichen und leistungsstarken Webanwendung, die es Nutzern ermöglicht, schnell und einfach herauszufinden, welcher Wein zu ihrem Essen passt. Insbesondere haben wir uns auf folgende Aspekte konzentriert:

* **Innovation**: Implementation einer zeitgemässen Lösung basierend auf Machine Learning
* **Benutzerfreundlichkeit (UI/UX)**: Gestaltung einer ansprechenden und intuitiven Benutzeroberfläche, die es den Nutzern erleichtert, Weininformationen und Empfehlungen zu erhalten. Dazu gehört unter anderem die Möglichkeit, mehrere Weine gleichzeitig per Spracherkennung zum digitalen Weinkeller hinzufügen zu können.
* **Performance und Skalierbarkeit**: Umsetzung als Single Page Application, welche in Zukunft zu einer cloud-basierten Enterprise Applikation ausgebaut werden könnte.
* **Effizienz**: Optimierung der Such- und Empfehlungsfunktionen, um den Nutzern eine schnelle und präzise Antwort auf ihre Fragen zur Wein- und Speiseempfehlung zu bieten.

### Welche technischen Grundsatzentscheide habt ihr gefällt?

* **Audio-Transkription**: Wir wollten mindestens ein innovatives und nutzerstiftendes Feature implementieren.
* **Technologie-Stack**: Wahl von C# .NET für das Backend und Angular für das Frontend, um eine moderne, reaktive und benutzerfreundliche Anwendung zu ermöglichen, welche dem Zeitgeist entspricht.
* **Datenbank**: Einsatz von Azure SQL Server für eine robuste, skalierbare und effiziente Verwaltung der Weininformationen und Rezepte. Dies hat zudem die Entwicklung erleichtert, da so alle Team-Mitglieder standortunabhängig auf die Datenbank zugreifen können.
* **Architektur**: Das Backend ist mit Clean-Architecture und Patterns wie CQRS (MediatR) aufgebaut.

## Technischer Aufbau

### Welche Komponenten und Frameworks habt ihr verwendet?

* **Frontend**: Angular, TypeScript, HTML, CSS, Bootstrap
* **Backend**: C# .NET 8
* **Datenbank**: Azure SQL Datenbank
* **CI/CD**: GitHub Actions für automatisierte Tests und Deployment

### Wozu und wie werden diese eingesetzt?

* **Angular**: Für die Entwicklung der interaktiven und dynamischen Benutzeroberfläche. Angular bietet eine robuste Framework-Architektur für Single-Page-Applications (SPAs).
* **TypeScript**: Für die clientseitige Logik und Interaktivität innerhalb der Angular-Anwendung.
* **HTML & CSS**: Zur Strukturierung und Gestaltung der Benutzeroberfläche, wobei HTML für die Struktur und CSS für das Styling verwendet wird.
* **Bootstrap**: Für die schnelle und konsistente Gestaltung von Benutzeroberflächenkomponenten, um eine ansprechende und responsive Benutzererfahrung zu gewährleisten.
* **C# .NET 8**: Für das serverseitige Backend, einschließlich der API-Entwicklung und der Verarbeitung von Datenanfragen und -antworten.

## Implementation

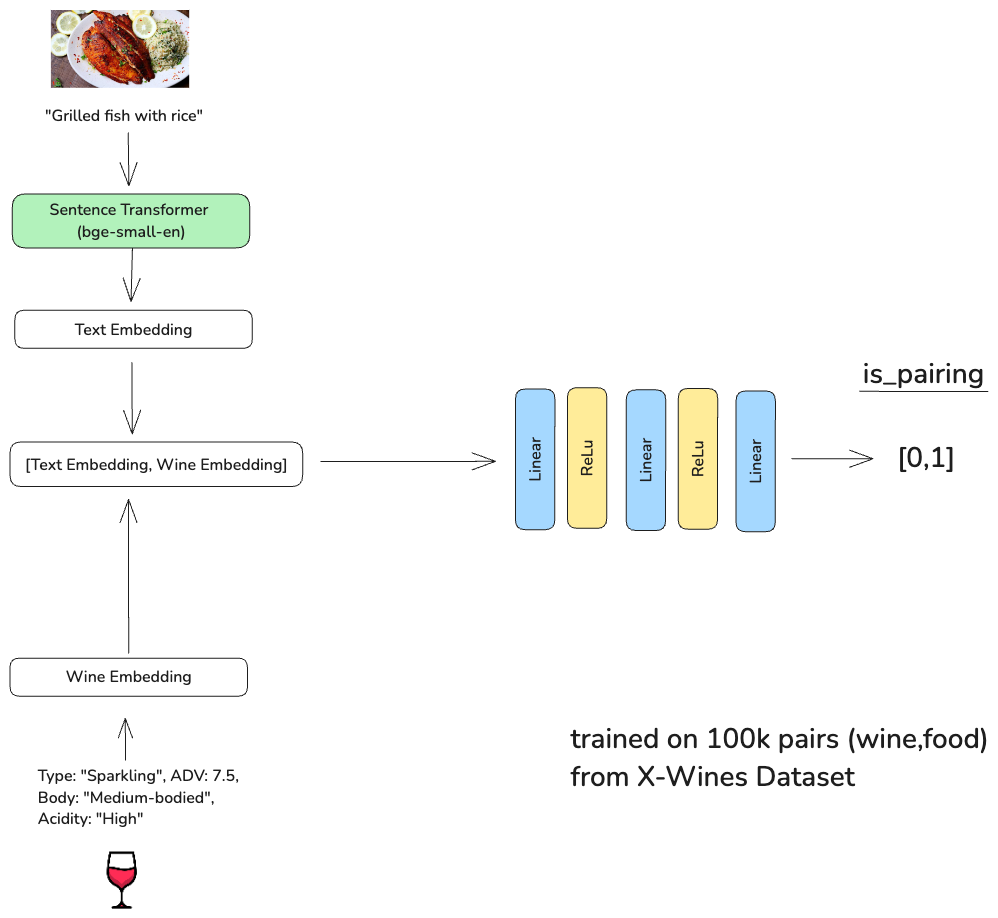
### Machine Learning Model

#### Modellbeschreibung

Das Machine Learning Modell wurde entwickelt, um vorherzusagen, ob eine bestimmte Wein- und Speisekombination gut zusammenpasst. Das Modell ist ein neuronales Netzwerk, das sowohl Lebensmittel- als auch Weineinbettungen umfasst. Die Architektur im Detail:

* **Text Embedding**: Wandelt die Beschreibung der Speisen (z.B. „Gegrillter Fisch mit Reis“) in Vektoren um, die mithilfe eines vortrainierten Sprachmodells (bge-small-en-v1.5) erstellt werden. Dies ermöglicht dem Modell, den Kontext der Speisen zu verstehen.
* **Wine Embedding**: Wandelt Weinmerkmale (wie Typ, ABV, Körper, Säure) in Vekotren um, die durch eine benutzerdefinierte Projektionsebene erzeugt werden. Dies ermöglicht es dem Modell, die Eigenschaften des Weins mit dem Embedding der Speisen zu vergleichen
* **Concatenation**: Kombiniert die Lebensmittel- und Weineinbettungen zu einem einzigen Vektor, der die Paarung der beiden repräsentiert.
* **Classification Layers**: Verwendet vollständig verbundene Schichten, um die kombinierten Einbettungen zu analysieren und auszugeben, ob der Wein zur Speise passt (0 oder 1).

Das Modell wurde mit 100.000 Paaren basierend auf der X-wine-Datenbank trainiert ([X-wine Dataset](https://www.mdpi.com/2504-2289/7/1/20)).





### Audio Transcription and Wine Name Extraction (audiowine.py)

#### Funktionalität

Dieses Feature ermöglicht es Nutzern, Weine durch eine Audioaufnahme verbal zu beschreiben. Das System transkribiert das Audio, extrahiert die genannten Wein-Namen und gleicht diese Namen mit einer Weindatenbank ab. Dies ist besonders nützlich für Szenarien, in denen ein Nutzer Weine vor sich hat und diese in der App einpflegen möchte.

* **Transcribe Audio**: Wandelt gesprochene Worte aus einer Audiodatei in Text um, indem ein Speech-to-Text-Modell verwendet wird. Es nutzt Whisper von OpenAI.
* **Extract Wine Names**: Identifiziert und extrahiert automatisch Wein-Namen aus dem transkribierten Text. Dieser Schritt ist entscheidend für das Erkennen der spezifischen Weine, die erwähnt werden. Es verwendet GPT-4 von OpenAI.
* **Match Wine Names**: Vergleicht die extrahierten Wein-Namen mit einer Datenbank bekannter Weine, um die genauesten Übereinstimmungen zu finden. Dies hilft, die genauen Weine zu identifizieren, auch wenn die Namen mit leichten Variationen erwähnt werden.

### Was ist aus technischer Sicht besonders cool an eurer Lösung?

Besonders hervorzuheben ist die Kombination aus dem leistungsfähigen Machine Learning Modell und der Audio-Transkriptionsfunktion, die eine nahtlose und präzise Identifikation und Empfehlung von Wein- und Speisekombinationen ermöglicht. Die Echtzeit-Datenverarbeitung durch WebSocket-Technologie und die Integration fortschrittlicher Modelle wie GPT-4 tragen zur Benutzerfreundlichkeit und Effizienz der Anwendung bei.

## Abgrenzung / Offene Punkte

### Welche Abgrenzungen habt ihr bewusst vorgenommen und damit nicht implementiert? Weshalb?

* **Offline-Funktionalität**: Eine Offline-Funktionalität wurde bewusst nicht implementiert, um die Komplexität der Anwendung zu reduzieren und sich auf die Kernfeatures und eine hochwertige Benutzererfahrung zu konzentrieren.
* **Mehrsprachigkeit**: Die Unterstützung mehrerer Sprachen wurde in dieser Phase nicht realisiert, um die Entwicklungszeit zu verkürzen und eine ausgezeichnete Benutzererfahrung in der Ausgangssprache sicherzustellen.