BÄRNHÄCKT

leaf

«Familie Thunberg»
«Suffizienzplattform»

Informationen für die Jury

Inhalt

1 Zugänge

1.1 Aktueller Stand des Sourcecodes

- https://github.com/baernhaeckt [Organisation]
- https://github.com/baernhaeckt/Backend [Backend Source]
- https://github.com/baernhaeckt/Frontend [Frontend Source]
- https://github.com/baernhaeckt/Forecasting [Forecasting Source]
- https://github.com/baernhaeckt/Misc [Automated Tests, Helper Scripts, Doku]

1.2 CI/CD Pipeline

- https://dev.azure.com/baernhaeckt/BaernHaeckt2019/_dashboards/dashboard/2a5704d4-5069-4833-9704-ef6f9cde3cd2 [Dashboard]
- https://dev.azure.com/baernhaeckt/BaernHaeckt2019/_build?definitionId=1 [Backend Build]
- https://dev.azure.com/baernhaeckt/BaernHaeckt2019/_build?definitionId=2 [Frontend Build]
- https://dev.azure.com/baernhaeckt/BaernHaeckt2019/_build?definitionId=4 [Forecasting Build & Release]
- https://hub.docker.com/r/msallin/baernhaeckt/tags [BärnHäckt Docker Hub Repo]
- https://dev.azure.com/baernhaeckt/BaernHaeckt2019/_release?_a=releases&view=mine&definitionId=1 [Backend Release]
- https://dev.azure.com/baernhaeckt/BaernHaeckt2019/_release?_a=releases&view=mine&definitionId=2 [Frontend Release]

1.3 Production

- https://baernhaeckt.azurewebsites.net/swagger [Backend – Swagger UI]
- https://baernhaeckt.azurewebsites.net/swagger/v1/swagger.json [Backend – Swagger Defintion]
- https://baernhaeckt.z16.web.core.windows.net [Frontend]
- https://baernhaecktforecasting.azurewebsites.net/forecasting [Forecasting output]

2 Fachliche Aspekte

Dieses Kapitel gibt einen kurzen Überblick über die fachlichen Aspekte und den Umfang, der Lösung.

2.1 Ausgangslage

Im Rahmen der Challenge «Suffizienzplattform», soll eine Sozialplattform[1] erstellt werden. Unsere Plattform bzw. unser Software-System, welches die Plattform bildet, trägt den Namen «Leaf». Eine Sozialplattform lebt von ihren Nutzern. Wenn keine kritische Masse erreicht wird, dann wird die Plattform den Durchbruch nicht schaffen. Aus diesem Grund muss das *Onboarding* möglichst einfach sein. Damit die Plattform lebt, müssen die Benutzer mit ihr interagieren. Die Interaktion bzw. die Datenerfassung müssen deshalb *einfach und schnell möglich* sein. Die Sozialforschung sagt uns, dass Menschen sich an ihrem Umfeld messen und dann danach streben, in diesem Umfeld zu den besten zu gehören. Ein *Vergleich mit Freunden und den Umliegenden Benutzern* bringt diesen Ansporn. Mittels *Gamification* werden die Benutzer bei Laune gehalten und für ihre Bemühungen belohnt. Diese Bemühungen sind natürlich auch für die Freunde sichtbar! Mithilfe von *Machine Learning* erhalten die Betreiber der Plattform weitere Insights.

2.2 Umgesetzte Use Cases

- Einfaches Onboarding von Benutzern.
- Sammeln von «Suffizienz Punkte» per QR-Code.
- API für Partner, zum Generieren von «Suffizienz Punkten» zur Vergabe.
- «Suffizienz Punkte» sammeln, mittels einem Suffizienz Awareness Quiz.
- Gamification über «Suffizienz Punkte» & Awards.
- Vernetzung mit Freunden (hinzufügen, anzeigen und entfernen).
- Realtime Benachrichtigungen über Aktivitäten der Freunde.
- Ranking gegenüber anderen Personen (Schweiz, Quartier und Freunde).
- Übersicht, über die Aktivitäten bzw. gesammelten Punkte.
- Vergleich der eigenen Suffizienz Bemühungen mit dem Durchschnittsschweizer.
- Übersichtskarte, welche den Wohnort der Freunde anzeigt und dessen Punkte.
- Auf der Karte können die Punkte der Freunde direkt mit den eigenen verglichen werden
- Eigenes Profil anzeigen und editieren.
- Login und Logout.

2.3 Abgrenzungen

Bei uns gilt Qualität vor Quantität. Die implementierten Features sind vollständig umgesetzt und können bereits verwendet werden. Gerade deswegen, mussten wir uns auf das essentielle fokussieren.

- Beim Onboarding wird ein Passwort generiert und der Benutzer würde per E-Mail informiert. Aktuell wird ein Standard Passwort gesetzt und es werden keine E-Mails versendet.
- Es wurde kein echter Partner angebunden.
- Die «langweiligen» Funktionalitäten, welche jede Plattform bieten muss (editieren der persönlichen Daten, Passwort ändern und/oder zurücksetzen) sind nicht umgesetzt.
- Das Neuronale Netzwerk wurde mit generierten Daten trainiert, da noch keine Daten zur Verfügung stehen.
- Die Suffizienz Punkte sind als Konzept nicht weiter durchdacht. Uns ist bewusst, dass es sich um eine fachliche Herausforderung handelt hier sinnvolle Berechnungen anzustellen.
- Die für den Vergleich mit dem Durchschnittsschweizer verwendeten Zahlen sind fiktiv und verändern sich über die Zeit nicht.

[1] https://www.bernhackt.ch/challenges-2019-2/#bfh [24.08.19

3 Technische Aspekte

Das folgende Kapitel zeigt den technischen Aufbau des «Leaf» Software-Systems. Zuerst werden Architekturentscheide erläutert, welche signifikanten Einfluss auf die Lösung haben. Danach wird die Baustein- und Verteilungssicht genauer erklärt.

3.1 Architekturentscheide

1. Die Applikation wird als Webapplikation realisiert. Das ermöglicht eine saubere Trennung zwischen Front- und Backend und dadurch kann die Anzahl Entwickler gut skaliert werden. Des Weiteren ist eine Plattformunabhängigkeit inhärent gegeben und muss nicht erzeugt werden (wie das z.B. bei Apps der Fall ist). Die Lösung kann zudem schnell und automatisch über eine CI/CD-Pipeline in die Produktion deployt werden.
2. Aus den oben bereits genannten Gründen erfolgt eine Trennung zwischen Frontend und Backend. Das Frontend wird als SPA implementiert, so wird ein gutes Benutzererlebnis erzeugt. Das Backend bietet die Funktionalität ausschliesslich über APIs an. Mit dieser Entkopplung wird die Evolution einfacher (z.B. parallel neues UI v2 und/oder Apps). Die Schnittstellen sind Hauptsächlich als Web APIs (REST & JSON) realisiert. Wo «push» vom Server nötig ist, werden Websockets eingesetzt.
3. Das Frontend wird mit VUE.JS realisiert. VUE.JS ermöglicht eine «progressive» Entwicklung. Das Framework lässt sich am Anfang leichtgewichtig einsetzen und lässt ein rasches Prototyping zu. Wird die Lösung grösser, skaliert das Framework mit und ermöglicht weiterhin die Umsetzung von wartbaren Lösungen. Aktuell wird JavaScript eingesetzt. Wächst die Lösung, ist der Umstieg auf TypeScript möglich.
4. Das Backend wird mit ASP.NET Core mit der Programmiersprache C# realisiert. Bei ASP.NET Core handelt es sich um ein Plattformunabhängiges «cloud native» Web Framework. Das Framework lässt sich mit seinem Middleware-Konzept sehr leicht an diverse Bedürfnisse anpassen. Standardszenarien sind ohne grossen Aufwand abgedeckt. Es ist zudem auf Durchsatz optimiert.
5. Der Insights-Teil d.h. die Prognosen werden mittels eins neuronalen Netzes (LSTM, 2 Layer) implementiert. Als Programmiersprache kommt Python und folgende Frameworks/Libraries zum Einsatz: Tensorflow Keras, NumPy, Phandas, MathPlotLib und Scikit-learn.
6. Die Persistenz wird mittels einer DocumentDb umgesetzt. Da rasch eine funktionsfähige Lösung entstehen soll, ist der Entscheid zugunsten einer schemalosen DocumentDb gefallen. Aufgrund der vorhandenen Vorkenntnisse wurde der Entscheid zugunsten von MongoDb getroffen.
7. Alle Komponenten werden in der Microsoft Cloud Azure gehostet. Dabei wird überall auf dem höchst möglichen Abstraktionlayer (z.B. SaaS) gearbeitet, um initialaufwand zu vermeiden.
8. Folgende Entscheide sind aufgrund der vorherrschenden Verbreitung der jeweiligen Technologien und dem Vorwissen des Teams gefallen:
    a. API Dokumentation: Swagger & Swagger UI
    b. Authentication: Bearer Token & JWT
    c. Automatisierte Tests & Helper: PowerShell Scripts
    d. Styling: Bootstrap
    e. Frontend Connectivity: Axios für http, SignalR für Websockets

3.2 Bausteinsicht

Die Bausteine Frontend, Forecast und Backend sind Eigenentwicklungen, während der Baustein Persistenz als SaaS bezogen wird.

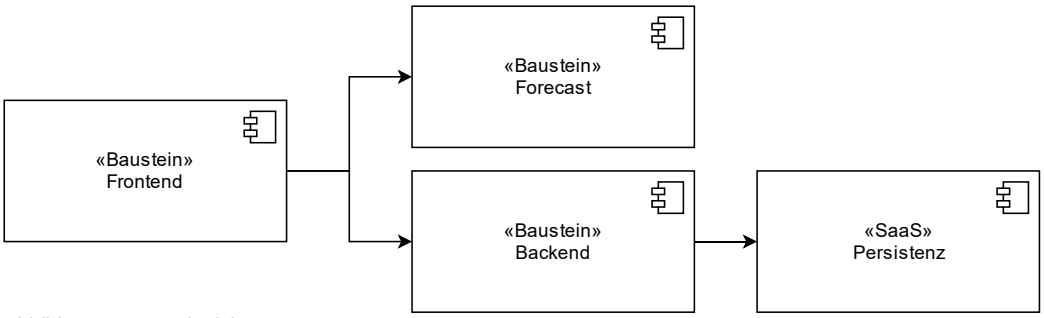

Abbildung 1 Bausteinsicht

3.2.1 Frontend

Das Frontend wurde mittels VUE.JS umgesetzt. Der Aufbau folgt dem Vue.js Standard. Es werden folgende Konzepte eingesetzt: Single-file-Components, vue-js-router, vuex. Für das Styling kommt Bootstrap und sass zum Einsatz.

3.2.2 Backend

Das Backend wird als eine einzelne Applikation umgesetzt. Es wird ASP.NET MVC eingesetzt. Die Lösung wird in Schichten gegliedert (wobei keine strikte Trennung erfolgt, es dürfen lediglich keine Referenzen von tieferen Schichten auf höhere Schichten erfolgen). Die «Web» Schicht stellt mittels Controller die REST-Endpoints bereit. Die Logik befindet sich im «Core» und wird in Services gekapselt. Der Datenbank Zugriff wird mit einer dünner Abstraktionsschicht (eine Library wird verwendet) gekapselt. Die Entitäten befinden sich im Projekt «Database».

3.2.3 Forecast

Der Forecast Teil liefert als Output eine Grafische Darstellung, der erwarteten Entwicklung, der Punktzahlen aller Nutzer auf der Plattform. Bei der Prognose handelt es sich um eine Extrapolation anhand der vorhandenen Daten.

3.2.4 Persistenz

Die CosmosDb von Microsoft Azure unterstützt die MongoDb API und das MongoDb Protokoll. Sie kann als SaaS bezogen werden und wurde deshalb ausgewählt. Sie enthält eine Datenbank und folgende Collections.
- QuizQuestions
- SufficientType
- Token
- User
- UserQuiz

3.3 Verteilungssicht

Die gesamte Lösung wird auf Microsoft Azure gehostet. Der Entscheid ist aufgrund von Vorwissen gefallen. Grundsätzlich wurde darauf geachtet, dass kein Vendor-Lock-In entsteht. Alle Komponenten lassen sich auch On-Premise oder in einer anderen Cloud betreiben.

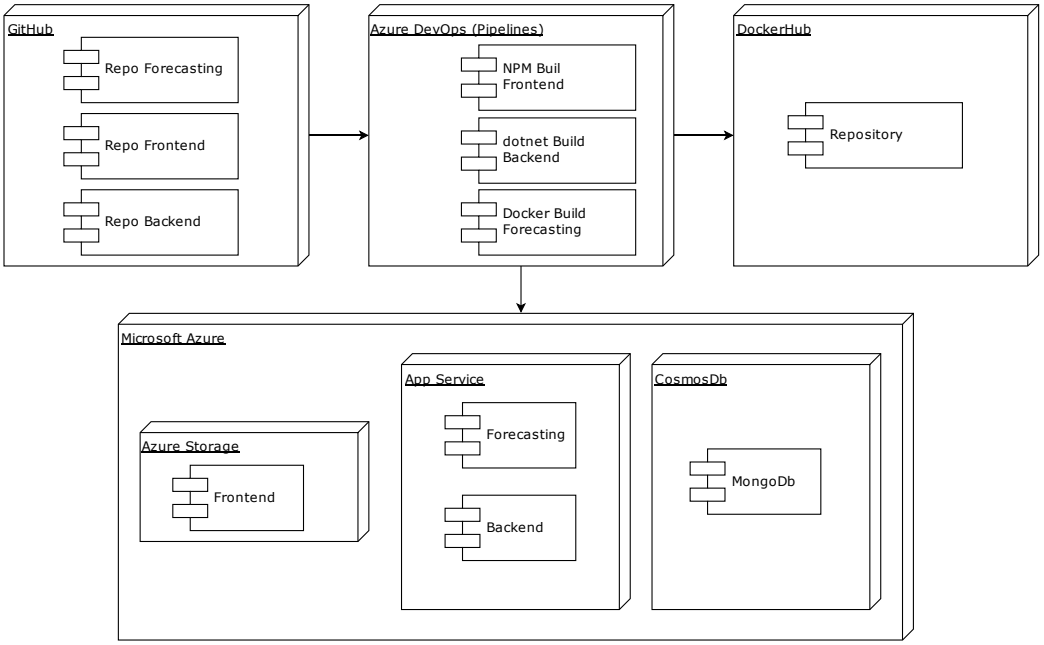

Abbildung 2 Verteilungssicht

Der Source Code wird innerhalb von mehreren Git-Repositories auf GitHub gehostet. Die CI/CD-Pipeline wurde mit Azure DevOps Pipelines implementiert.

3.3.1 Frontend

Azure Storage stellt die Möglichkeit bereit, sehr einfach und günstig eine statische Website zu hosten. Es muss keine Infrastruktur betrieben werden.

3.3.2 Backend

Azure App Service stellt eine Hosting Umgebung für verschiedene Technologien zur Verfügung. Es musst keine Infrastruktur betrieben werden. Das Backend wird nicht containerisiert, sondern direkt gehostet.

3.3.3 Forecast

Die Forecast Applikation wird als Single-Container auf Azure App Service gehostet. Somit ist auch hier keine weitere Infrastruktur nötig. Das Image für den Container wird über die CI/CD-Pipeline erstellt und steht auf einem öffentlichen DockerHub-Repository zur Verfügung.

3.3.4 MongoDb

Bei CosmosDb handelt es sich um eine vielseitige DocumentDb, welche verschiedene Protokolle und APIs unterstützt. Es handelt sich um eine SaaS-Lösung und es muss keine Infrastruktur betrieben werden.

3.4 Implementation

Nachfolgend einige Punkte, welche wir besonders hervorheben möchten.

- Alle Funktionalitäten, welche sichtbar sind, sind implementiert und voll funktionsfähig. Unser System verwendet keine Mocks oder grafische Elemente ohne Funktionalität.
- Für die kurze Entwicklungszeit ist der Funktionsumfang sehr umfangreich.
- Die gesamte Lösung ist live und für jedermann Verfügbar. Die Demos erfolgen auf Applikationen, die auf Azure laufen und nicht nur auf dem Entwicklungsrechner.
- Das System kann vollständig in der Cloud betrieben werden, es gibt jedoch trotzdem keinen Vendor-Lock-In.
- Die Lösung ist in der Entwicklung sowie auch bei der Verwendung Crossplattform tauglich.
- Einsatz von Websockets mithilfe von SignalR, zum Verteilen von Meldungen an die richtige Audienz, mittels Server-Client-Kommunikation.
- Vollständige Automatisierung von Build und Deployment (CI/CD-Pipeline), für alle drei Teilkomponenten, welche ein Deployment benötigen.
- Grundlagen für Sicherheit sind gelegt (Verwendung von HTTPS, signierten und ablaufenden JWT, sicher gespeicherte Passwörter, CORS).
- Ein automatisierter SmokeTest, welcher den Happy-Path für einen Grossteil der Use Cases abdeckt, ist vorhanden.
- Die zur Verfügung stehenden APIs sind mittels Open API dokumentiert.
- Die eingesetzten Technologien sind alle samt Enterprise fähig.
- Der geschriebene Code ist grundsätzlich, für das er im Rahmen eines Hackthons entstanden ist, in einem gut wartbaren Zustand und die Lösung kann weiterentwickelt werden.

4 Ausblick

Funktionalitäten für Endbenutzer

- Ausbau der Standard Funktionalitäten einer Online Plattform (Password zurücksetzen usw).
- Benutzer können manuell Tätigkeiten erfassen und von ihren Freunden bewerten lassen.
- Ausbauen der Awards und Suffizienz-Tätigkeiten.
- Sozialmedia Funktionalitäten implementieren (z.B. Chat, Posten/Liken/Kommentieren).
- Bessere Führung durch z.B. Hilfe, FAQ, Onboarding Wizard.
- Vorschläge zu weiteren Suffizienz-Tätigkeiten bzw. Partner.

Funktionalitäten für den Plattformbetreiber

- Auswertungen für den Plattform Betreiber, zum Verhalten der Benutzer.
- Möglichkeit zur Kommunikation mit den Endbenutzern (z.B. Microblog-Feed)

Funktionalitäten für Partner

- Partner Login mit Auswertungen (z.B. wie viele Tokens wurden generiert und verwendet).
- Direktanbindung von Partnern (z.B. BKW Energie verbrauch oder SBB Fahrten).

5 Verzeichnisse