BCS KI Pugin

Norman Steinhoff

Version vom 27. Februar 2025

Inhaltsverzeichnis

1 Entwicklung

Im Rahmen dieses Software-Projekts sollten wir eine Chrome-Extension entwickeln, die das Ticketsystem der Firma Projektron um KI-Funktionen erweitert. Dazu sollten wir einerseits ein lokales Modell (on-device) und ein externes Modell verwenden.

Wir haben Gemini Nano als lokales und Llama-3.3 als externes Modell verwendet.

1.1 Gemini Nano

Gemini Nano ist ein von Google entwickeltes Sprachmodell [1]. Der Hauptzweck von Gemini Nano ist es, KI Funktionen clientseitig ausführen zu können, ohne dafür jedesmal ein Sprachmodell herunterladen zu müssen. Dazu wurde Gemini Nano direkt in Chrome Canary integriert. [2]

Aktuell befinden sich Gemini Nano und die dazugehörige API in einer Testphase. In diesem Zustand können und haben sich [1] deren Funktionen ständig geändert. [2]

Die Gemini API ist in verschiedene Task API's aufgeteilt. Aktuell sind dies [3]:

- LanguageDetection
- Translation
- Summary
- Writer + Rewriter
- PromptApi [2]

1.2 Dokumentation von Gemini Nano

Um die oben genannten API's nutzen zu können sollte man dem Origin Trial beiteten. Dies ging über [4]. Hier erhiehlten wir die jeweils neuesten Informationen via Email. Dies war notwendig, da die eigentliche Dokumentation [5, 6, 7, 8, 9] seit längerer Zeit nicht mehr aktuell ist (Stand 04.02.2025). Inzwischen gibt es zwar über [2] eine neue öffentlich zugängliche Dokumentation, allerdings ist diese ebenfalls nicht vollständig.

1.3 Crome Canary

Chrome Canary ist die experimentelle Version von Googles Chrome Browser und bietet die Möglichkeit, die neuesten Funktionen zu testen. Canary kann als eine Art Labor betrachtet werden, in dem neue Ideen und Features ausprobiert werden, bevor sie in die endgültige Version von Chrome aufgenommen werden. Diese Vorabversion wird täglich aktualisiert, wodurch die neuesten Entwicklungen stets verfolgt werden können. Da es sich jedoch um eine experimentelle Version handelt, kann es vorkommen, dass Canary instabil ist oder Fehler aufweist [10].

Zum Zeitpunkt der Entwicklung unserer Extension war Chrome Canary die einzige Möglichkeit Gemini Nano zu benutzen.

1.4 Notwendige Voraussetzungen für Gemini Nano

Gemini Nano ist unter Windows (10, 11) und unter MacOs (>=13) verfügbar, sofern man mindestens 22 GB freien Speicher hat, über eine GPU verfügt und >= 6 GB VideoRAM hat.

Um Gemini Nano in Chrome Canary nutzen zu können müssen dann mehrere Flags gesetzt werden. Dazu gibt man in der Adresszeile `chrome://flags` ein. Dort müssen dann folgende Flags gesetzt werden:

- Enables optimization guide on device –> *Enabled BypassPerfRequirement*

[1] Wie wir mehrfach während der Entwicklung unseres Plugins feststellen mussten

[2] Unser Plugin stützt sich letzlich auf die PromptApi, auch wenn wir zwischendurch die anderen API's ausprobiert haben.

- Text Safety Classifier [3] –> *Disabled*
- Language detection web platform API –> *Enabled*
- Experimental translation API –> *Enabled without language pack limit*
- Summarization API for Gemini Nano –> *Enabled*
- Writer API for Gemini Nano –> *Enabled*
- Rewriter API for Gemini Nano –> *Enabled*
- Prompt API for Gemini Nano –> *Enabled*

Desweiteren sollte man danach in der Adresszeile `chrome://components/` eingeben und dann *Optimization Guide On Device Model* suchen und aktualisieren.

Um zu testen, ob alles korrekt eingestellt ist und ob man die notwendigen Voraussetzungen erfüllt kann man nun in der Console des Chrome Browsers [4] folgendes eintippen: `await self.ai.languageModel.availability()` falls man nun `after-download` oder `available` als Antwort erhält, kann man Gemini Nano prinzipiell verwenden. Sollte man jedoch `no` als Antwort bekommen, so wird der verwendete Computer nie in der Lage sein Gemini Nano auszuführen.

1.5 Chrome Extensions

Chrome-Extensions [11] sind kleine Softwareprogramme, die die Funktionalität des Google Chrome-Browsers erweitern und an die individuellen Bedürfnisse des Nutzers anpassen. Sie stellen eine Bereicherung des Browser-Erlebnisses dar, indem sie zusätzliche Funktionen implementieren, bestehende modifizieren oder repetitive Aufgaben automatisieren.

Chrome-Extensions basieren auf Webtechnologien wie HTML, CSS und JavaScript. Somit sind sie prinzipiell in jedem Texteditor erstellbar.

1.6 TypeScript

Typescript (TS) ist eine von Microsoft entwickelte Erweiterung von JavaScript (JS). [12]

TS wird zu JS convertiert, wodurch der geschriebene Code in nahezu jedem Browser läuft. [12]

TS erweitert JS durch ein Typsystem in dem unter anderem Interfaces und Klassen beschrieben werden können. Dadurch reduzieren sich die möglichen Fehlerquellen enorm, da der TS-Compiler einfache Typ-Überprüfungen vollziehen kann. Desweiteren ist dadurch auch eine bessere Integration mit IDE's wie zum Beispiel VSCode gegeben. [12]

[3] Wenn der Text Safety Classifier nicht ausgeschaltet ist, weigert sich Gemini Nano Texte zu verarbeiten, die nicht in Englsich sind.

[4] auf einem Mac: `Darstellung -> Entwickler -> Java-Script-Konsole`

TS ist dabei so entworfen worden, dass jeder gültige JS-Code auch gültiger TS-Code ist. Es ist also möglich ein bestehendes JS-Projekt ohne weiteres auf TS umzustellen. Anschließend kann man, wenn man möchte, den vorhandenen Code zu TS convertieren. [12] [13] Dabei hilft ein Feature von TS. Die so genannte tsconfig.json Datei enthält Einstellungen für den TS-Compiler um beispielsweise erweitertes ErrorChecking zu ermöglichen. [14]

1.7 Wie haben wir Typescript genutzt?

Aufgrund der vielen Vorteile von TS haben wir unsere Chrome Extension direkt und vollständig (100 %) in TS geschrieben.

Außerdem haben wir die tsconfig.json Datei so eingestellt, dass der TS-Compiler maximal streng vorgeht. Dies bedeutet, dass wir so viele Warnungen und Vorschläge vom Compiler bekommen möchten, wie irgend möglich. Außerdem wurde eingestellt, dass jede Warnung als Fehler zu interpretieren ist und zum Abbruch führt. Für weitere Informationen zur tsconfig.json kann man auf der offiziellen Webseite [14] nachsehen .

1.8 Architektur

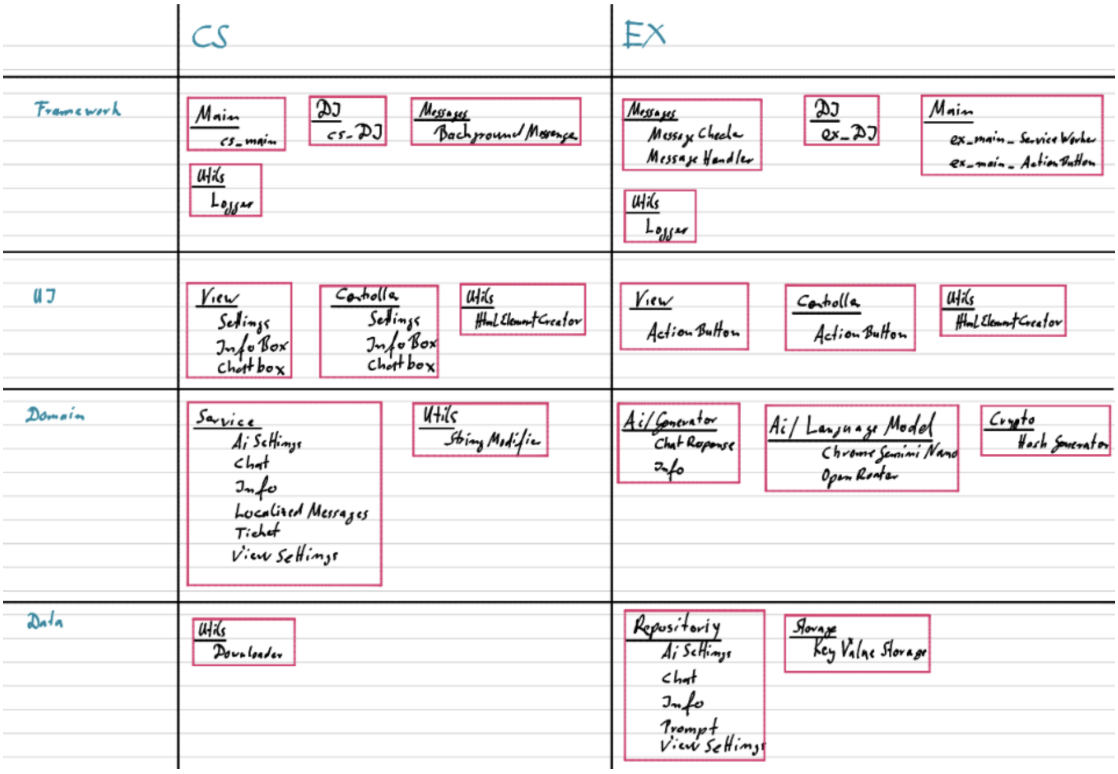

Abbildung 1: Architektur

Wir haben unsere Extension nach einem Schichten-Modell aufgebaut, welches eine

Daten-, Domain-, UI- und eine Framework-Schicht enthält. Durch die Vorgabe, dass wir eine Chrome-Extension erstellen sollten, gab es zusätzlich noch eine Unterteilung in Content-Script (CS) und Extension-Script (EX) [5].

1.8.1 Content-Script (CS)

Das CS in einer Chrome-Extension hat zugriff auf die gerade angezeigte Seite. Dies bedeutet, es ist in der Lage die aktuellen Daten der Seite zu lesen und das DOM der Seite zu manipulieren. Eine der wenigen Chrome-APIs auf die man im CS zugriff hat ist `hrome.runtime.sendMessage`. Hiermit können Nachrichten an das EX gesendet und auf deren Antwort gewartet werden.

1.8.2 Extension-Script (EX)

Das EX hat follen Zugriff auf die Chrome-APIs, kann aber nicht auf die aktuelle Seite zugreifen. Es kann Listener registrieren, die auf verschiedene Events reagieren. Zum Beispiel gibt es die Events `onInstall` und `onUpdate`, aber auch das Event `onMessage` [6]. Das letztgenannte Event ermöglicht es die vom CS gesendeten Nachrichten zu verarbeiten. Nach dem erhalt eines Events hat das EX genau 30 Sekunden Rechenzeit [7], bevor es unterbrochen wird.

1.8.3 Data-Layer

Zu unterst befindet sich bei uns die Data-Layer. Sie ist für das Speichern und Laden der Daten zuständig. Außerdem haben wir hier eigene Datentypen definiert und für jeden zu speichernden Datentyp haben wir ein Repository erstellt. Die Repositories dienen als Abstraktion um zu verbergen, wie und wo die Daten gespeichert werden. Mögliche denkbare Optionen wären lokal vs remote oder Key-Value-Storage vs. Database vs File.

Während unserer Evaluation der Speicher-Möglichkeiten haben wir uns folgendes überlegt:

- Wir haben nur einen User –> Es ist sehr unwahrscheinlich, dass konkurrierende Zugriffe gemanaged werden müssen.
- Selbst wenn Daten überschrieben werden, ist dies egal, da sie von der KI generiert wurden. Und man diesen Vorgang im Zweifel nochmal starten kann.
- Mit `chrome.storage.local` gibt es eine einfach zu nutzende API.
- Wir haben keine komplexen Anfragen an den Speicher. Ein Element wird entweder ganz oder gar nicht gebraucht [8].

[5] Diese Unterteilung haben wir nicht freiwillig getroffen, sondern sie ist eine Notwendigkeit in Chrome-Extensions.
[6] und weitere Events
[7] Dies ist eine harte Randbedingung von Chrome
[8] Kein `select, from, where, ...`

Gemäß dem KISS-Prinzip [9] haben wir uns daher gegen die Nutzung einer Datenbank und für einen Key-Value-Storage entschieden.

1.8.4 Domain-Layer

Die zweite Schicht ist die Domain Layer. Hier werden die eigentlichen Aufgaben der Extension ausgeführt.

Die Domain-Layer bietet bei uns verschieden Service-Klassen an, über die die UI-Layer aufträge erteilen oder Daten erhalten kann. Sie befinden sich im CS. Die Service Klassen selbst verschicken und empfangen hauptsächlich Nachrichten an das EX.

Im EX wiederum befinden sich zwei Generator-Klassen. Diese dienen als Abstraktion über das verwendete Sprachmodell. Die Aufgabe des Infogenerators besteht darin die statischen Informationen wie Zusammenfassung, Todos oder Termine zu generieren, während der ChatResponse-Generator sich um die antworten des Chatbots kümmert.

Das zentrale Interface unser Extension ist das LanguageModel. Es stellt folgende Methoden bereit:

- modelName(): Promise<string>
- setSystemPrompt(systemPrompt: string): Promise<void>
- releaseResources(): Promise<void>;
- generate(prompt: string;): Promise<string | undefined>;
- maxTokens(): Promise<number>;
- countTokens(prompt: string): Promise<number>;

Die Methode `setSystemPrompt` bereitet das Sprachmodell auf seine Aufgabe vor. Die Methode `generate` generiert die eigentliche Antwort des Sprachmodells.

Die Methoden `maxTokens` und `countTokens` sind wichtig um die Länge der Eingabe zu überprüfen, da die meisten Sprachmodelle nur eine begrenzte Menge [10] an Daten verarbeiten können.

Im Fall von Gemini Nano ist die Methode `releaseResources` wichtig, da Gemini Nano immer nur von einem Benutzer verwendet werden kann.

Wir haben in unserer Extension zwei Implementierungen des LanguageModel-Interface. Diese sind ChromeGeminiNano und OpenRouter. OpenRouter ist unsere Wahl um das Remote Modell zu implementieren. Hier werden Http Aufrufe an den Server von Open Router getätigt. Wir übergeben dabei mehere Parameter wie zum Beispiel das gewünschte Sprachmodell und den Prompt und warten dann auf die Antwort des Servers.

Generell ist unsere Extension so aufgebaut, dass überall nur das Interface LanguageModell verwendet wird. Das heißt außer zum Zeitpunkt der Auswahl des konkreten LanguageModell weiß unsere Extension nie, welches Sprachmodell sie verwendet [11]. Dies

[9] Keep It Simple Stupid

[10] Diese Menge wird in Tokens gemessen. Wir überlassen diese Messung jeweils dem verwendeten Sprachmodell

[11] ... und damit auch nicht, ob die Verarbeitung lokal oder remote erfolgt

vereinheitlicht und vereinfacht die Verarbeitung der Prompts.

1.8.5 UI-Layer

In der UI-Layer haben wir eine strenge Trennung zwischen der Anzeige und der Auswahl der Daten getroffen. Unsere View-Klassen zeigen lediglich Daten an, die an sie übergeben wurden und detektieren Events wie zum Beispiel Clicks. Für die Auswahl der anzuzeigenden Daten, sowie die Reaktion auf die detektierten Events sind unsere ViewController-Klassen zuständig. Die ViewController halten außerdem Referenzen auf die Service-Klassen der Domain-Layer. Sie sind aber lediglich für die Auswahl der Service-Methoden zuständig und geben den Service-Klassen Aufträge.

Unser UI besteht im Wesentlichen aus drei Teilen:

- InfoBoxen –> zeigen statische Informationen an
- Chatbot –> beantwortet Fragen des Nutzers
- Settings –> verschiedene Einstellmöglichkeiten

Die InfoBoxen und die Settings orientieren sich am Design von Projektron BCS. Beim Chatbot haben wir ein eigenes Design gewählt.

1.8.6 Framework-Layer

Als oberste Schicht unser Architektur haben wir eien Framework-Layer eingefügt. In dieser Schicht befinden sich die `main`-Funktionen des CS und des EX, sowie jeweils eine Klasse die für Dependency Injection (DI) zuständig ist [12].

Die DI-Klasse hat für jeden Datentyp mindestens eine Funktion, welche ein Objekt dieses Typs erzeugt. Dies ist die einzige Stell im Code, an der wir Objekte erzeugen. Zur Erzeugung von Objekten ist zu sagen, das wir an keiner Stelle Vererbung benutzt haben und stattdessen vollständig auf Komposition gesetzt haben. Dies bedeutet [13], dass keine Klasse eine IST-EIN Beziehung sonder stattdessen (viele) HAT-EIN Bezihung(en) aufweist. In den DI klassen werden diese Objekte daher zusammengesetzt.

Die Kommunikation zwischen CS und EX haben wir auch in dieser Schicht angesiedelt. Dazu haben wir einen Messenger (zum Senden) und MessageHandler (zum Empfangen und Behandeln) von Nachrichten erstellt.

Literatur

[1] Google. Artificial intelligence | chrome for developers. `https://developer.chrome.com/docs/ai`. Accessed: 2025-02-04.

[12] Soweit das in Typescript möglich ist. Eigentlich würde man für so etwas Reflection benutzen, was es in Typescript aber nicht zur Laufzeit gibt.

[13] Mit Ausnahme der Interfaces natürlich

[2] Google. Built-in ai | ai on chrome | chrome for developers. https://developer.chrome.com/docs/ai/built-in. Accessed: 2025-02-04.

[3] Google. Built-in ai apis | ai on chrome | chrome for developers. https://developer.chrome.com/docs/ai/built-in-apis. Accessed: 2025-02-04.

[4] Google. Join the prompt api for chrome extensions origin trial | blog | chrome for developers. https://developer.chrome.com/blog/prompt-api-origin-trial. Accessed: 2025-02-04.

[5] Google. The language detection api - update 4 - google docs. https://docs.google.com/document/d/1lY40hdaWizzImXaI2iCGto9sOY6s25BcDJDYQvxpvk4/edit?tab=t.0. Accessed: 2025-01-01.

[6] Google. The translation api - update 6 - google docs. https://docs.google.com/document/d/1bzpeKk4k26KfjtR-_d9OuXLMpJdRMiLZAOVNMuFIejk/edit?tab=t.0. Accessed: 2025-01-01.

[7] Google. The summarization api - update 3 - google docs. https://docs.google.com/document/d/1Bvd6cU9VIEb7kHTAOCtmmHNAYlIZdeNmV7Oy-2CtimA/edit?tab=t.0. Accessed: 2025-01-01.

[8] Google. The writer and rewriter apis - update 5 - google docs. https://docs.google.com/document/d/1WZlAvfrIWDwzQXdqIcCOTcrWLGGgmoesN1VGFbKU_D4/edit?tab=t.0. Accessed: 2025-01-01.

[9] Google. Welcome and heads-up about the prompt api - update 1 - google docs. https://docs.google.com/document/d/1VG8HIyz361zGduWgNG7R_R8Xkv0OOJ8b5C9QKeCjU0c/edit?pli=1&tab=t.0#heading=h.drihdh1gpv8p. Accessed: 2025-01-01.

[10] Google. Chrome canary-funktionen für entwickler– google chrome. https://www.google.com/chrome/canary/. Accessed: 2025-02-04.

[11] Google. Chrome extensions | chrome for developers. https://developer.chrome.com/docs/extensions. Accessed: 2025-01-01.

[12] Microsoft. Typescript: Javascript with syntax for types. https://www.typescriptlang.org. Accessed: 2025-02-04.

[13] Microsoft. Typescript: Documentation - typescript for the new programmer. https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html. Accessed: 2025-02-04.

[14] Microsoft. Typescript: Tsconfig reference - docs on every tsconfig option. https://www.typescriptlang.org/tsconfig/. Accessed: 2025-02-04.