|  |
| --- |
| 3050 Webprogrammierung und interaktive DatenvisualisierungAufgabe und Bewertung Projektarbeit |

Aufgabe

Programmiere als **Projektarbeit im Modul 3050** eine **geeignete Applikation** und beantworte die **Reflexionsfragen**. **Jede:r** Student:in **bearbeitet** die **Aufgabe selbstständig**. Eine Verwendung von KI-Tools oder anderen Werkzeugen zur automatisierten Text- oder Code-Generierung ist nicht gestattet.

Das Ziel dieser Projektarbeit ist eine Daten-basierte Webapp zur Erkundung und Visualisierung eines ausgewählten Datensatzes der **opendata.swiss** Plattform zu erstellen. Den Benutzenden sollte sie mit Hilfe von UI Elementen, tabellarischen und visuellen Inhalten eine Antwort auf eine konkrete Fragestellung in Bezug auf die Daten geben.

Die Projektarbeit besteht aus drei Teilschritten:

1. Die **Wetterdaten** der **opendata.swiss** Plattform sind die Datengrundlage für deine Applikation. Zunächst machst du dich mit dem Datensatz vertraut und überlegst, welche Fragestellung du damit beantworten oder welches Nutzenden-Bedürfnis du damit abdecken möchtest.

Links:

1. Meteodaten-täglich: <https://opendata.swiss/de/dataset/taglich-aktualisierte-meteodaten-seit-19921>
2. Meteodaten-stündlich: <https://opendata.swiss/de/dataset/stundlich-aktualisierte-meteodaten-seit-1992>

Die Wetterdaten für 2023 sind bereits aufbereitet (um Standort-Metadaten ergänzt worden) und als Dateien in **Moodle** vorhanden:

* 1. JSON Dateien: meteodaten\_2023\_daily.json, meteodaten\_2023\_hourly.json
  2. CSV Dateien: meteodaten\_2023\_daily.csv, meteodaten\_2023\_hourly.csv

Wir empfehlen die *täglichen* Daten zu verwenden und zu den *stündlichen* Daten nur dann zu greifen, wenn alles andere in der App läuft und du eine Extrameile mit den Visualisierungen gehen willst. Die Entscheidung, welches Format (CSV o. JSON) du nimmst, ist dir überlassen.

1. Anhand der Minimalanforderungen (s.u.) planst, entwickelst und veröffentlichst du deine WebApp. Die konkrete Umsetzung und Gestaltung kannst du frei bestimmen, solange deine App dir erlaubt nachstehende Reflexionsfragen anhand deiner Umsetzungen zu beantworten.
2. Abschliessend beantwortest du die Reflexionsfragen zu deinem Projekt.

## Minimalanforderungen

Die folgenden Kriterien muss deine WebApp mindestens erfüllen.

* Deine Applikation muss über ein Frontend sowie ein Backend verfügen, welche miteinander kommunizieren können.
* Das Frontend (GUI) muss mindestens drei sinnvolle Interaktionen für die Nutzenden und die zu erfüllenden Aufgaben anbieten.
* Das Frontend beinhaltet mindestens eine Visualisierung der Daten in geeigneter Form, welche von dem Backend aufbereitet und abgerufen werden (z.B. durch Nutzer-Interaktion)
* Das Backend wird in Python mittels FastAPI implementiert.
* Die App wird auf einem Linux-Server bereitgestellt. (Server-Zugang wird zur Verfügung gestellt)

Reflexionsfragen – Teilaufgaben

In Bezug zu deiner implementierten WebApp zu beantworten:

1. Beschreibe das Ziel und die Funktionalität deiner WebApp (verwende ein Flussdiagramm und definiere die notwendigen UI-Elemente für die vorhandene Funktionalität).
2. Beschreibe mindestens eine mögliche **in**korrekte Interaktion mit der UI und wie das Feedback dazu ausfällt.
3. Beschreibe nach welchen Überlegungen du die WebApp in (*React-)Komponenten* unterteilt hast und gib dabei konkrete Beispiele anhand deiner Komponenten.
4. Dokumentiere die verwendeten Node.js (Frontend) sowie Python (Backend) Module / Bibliotheken und für welchen Zweck du sie in deiner WebApp eingesetzt hast.
5. Beschreibe, wie du die Daten handhabst. Welche Komponenten verwenden (Präsentation) oder manipulieren (Abrufen, Filtern, usw.) die verwendeten Daten?
6. Beschreibe die umgesetzte Visualisierung in Bezug auf die verwendeten visuellen Elemente (d.h. welche visuellen Elemente wurden warum für welche Datendimension eingesetzt?) sowie die Hauptbotschaft der Visualisierung.
7. Welche spezifischen Herausforderungen sind bei der Implementierung der API-Endpunkte aufgetreten und welche Strategien hast du angewendet, um diese zu überwinden? Nenne mindestens zwei Herausforderungen und Lösungen.c
8. Welche Sicherheitsmassnahmen hast du implementiert, und was waren die Gründe für die Auswahl dieser Massnahmen? Falls keine Sicherheitsmassnahmen implementiert wurden, erläutere die Gründe dafür.

Die **Resultate der Teilaufgaben** füllst du unten **ins Bewertungsformular** ein und reichst dieses Dokument mit Aufgabe und Bewertung **als PDF** (Dateiname: Nachname\_Projektarbeit.pdf) mit **maximal vier (4) A4 Seiten bis spätestens Freitag, 17.01.2024, 23:59h auf Moodle ein.**

Hinweis: Erstelle das Flussdiagramm (und allenfalls auch die Darstellung der dafür notwendigen GUI-Elemente) in einer Software deiner Wahl oder mache Handskizzen und füge lesbare Screenshots/Fotos in das Bewertungsformular ein.

## Bewertung

Verwende die Bewertungskriterien (blau, Anzahl Punkte in Klammer) und **erläutere deine WebApp** in Bezug dazu (dein Text in den grau hinterlegten Zellen/Flächen). Verwende dazu die detaillierteren Angaben aus der Aufgabenstellung

|  |  |
| --- | --- |
| Name Student:in | Jonas Heinz |
| WebApp URL (6) | *https://wettervergleich.vercel.app/* |
| GitHub URL (6) | *https://github.com/JonasHeinz/wettervergleich* |
| Reflexionsfrage 1 (3) | Das Ziel war es eine Webapp zu entwickeln mit den Meteodaten eines Zeitraumes mit den Meteodaten des gleichen Zeitraumes eines anderen Jahres vergleichen kann. Z.B. Sollte es möglich sein die Temperatur im Monat April 2020 mit der im April 2000 vergleichen zu sollen. Mithilfe solcher Vergleiche sollte es möglich sein z.B. Klima Veränderungen über die Jahre festzustellen. Ausserdem sollte die App auch einfach zum Spass oder aus Interesse verwendet werden können, um z.B. rauszufinden, ob es in diesem Jahr im Vergleich zu einem anderen viel geregnet hat. (Flussdiagramm und UI-Elemente sind auf Seite 7) |
| Reflexionsfrage 2 (3) | Für mich war es wichtig, ungültige Interaktionen mit der Benutzeroberfläche von vornherein zu verhindern. Deshalb habe ich darauf geachtet, dass die Daten für die möglichen Zeiträume vollständig und konsistent sind. Ich habe mich entschieden, die Daten nach der Wetterstation Stampfenbachstrasse zu filtern und nur die Parameter Temperatur, Regendauer und Globalstrahlung zu verwenden, da andere Daten erst zu späteren Zeitpunkten erhoben wurden.  Ich habe ausserdem für den Date Picker Regeln festgelegt welche Werte auswählbarsein dürfen. Z.b. wurden Regendauerdaten erst 1999 erhoben, darum sollten für die Regendauer ältere Daten nicht auswählbar sein. Auch sollte es nicht möglichsein für das Intervall «Jahr» als Startdatum den heutigen Tag auszuwählen, sondern eines im Jahr 2023, da sonst der Zeitraum in der Zukunft liegen würde.  Sollte es dennoch gelingen, einen ungültigen Request zu versenden, wird dies durch eine Catch-Funktion abgefangen. Diese gibt eine entsprechende Fehlermeldung in der Konsole aus. Darüber hinaus wird das Diagramm erst dann dargestellt, wenn auch tatsächlich Daten für die Visualisierung vorliegen. |
| Reflexionsfrage 3 (3) | Ich habe grundsätzlich meine Komponenten nach Funktionalitäten und logischen Bereichen aufgeteilt. Dadurch, dass es sich nur um eine kleinere Single Page Applikation handelt, ist meine Einteilung in verschiedene Komponenten eher übertrieben. Ich wollte jedoch, dass die App einfach erweiterbar ist. Z.b habe ich für den Header eine einzelne Komponente gemacht, obwohl dieser nur die Überschrift enthält, damit dieser z.B. mit einer Navigationsleiste erweitert werden kann. Ausserdem habe ich die komplette Interaktivität in einen eigenen Komponent gemacht. Dies habe ich vor allem für einen strukturierten übersichtlichen Code gemacht aber auch für eine eventuelle Wiederverwendung des Formulars. Gleiches gilt auch für die Statistik. (Komponenten sind auf Seite 7) |
| Reflexionsfrage 4 (3) | **Frontend**  React: Grundlage zum Erstellen des Frontend (Komponenten, State Management)  Material-UI: Bereitstellung von vorgefertigten UI-Komponenten  MUI X: Bereitstellung von Experimentelle MUI-Komponenten in meinen Fall der DatePicker  Daysjs: Verarbeitung des Datums des DatePickers  React-Vega: Darstellung der Visualisierung  Axios: Erstellen von http-Anfragen zum Backend  **Backend**  FastAPI: Schnelles Webframework als Grundlage für das Backend  Pandas: Zum Datenmanipulieren und für die Erstellung des DataFrames für das Vega-Diagramm  Altair: Zum erstellen des spec JSON für die Datenvisualisierung  Uvicorn: Server zur Ausführung der FastAPI-Anwendung |
| Reflexionsfrage 5 (3) | Meine Applikation enthält einen Axios API-Aufruf, der mithilfe eines useEffect-Hooks auf Änderungen der State-Variablen reagiert, welche durch das Eingabeformular verändert, werden können. Dem HTTP-Request werden die benötigten Parameter übergeben, und als Antwort liefert das Backend ein JSON-Objekt zurück. Dieses enthält die Vega-Spec, den Durchschnitt der Wetterwerte für beide Jahre sowie die entsprechende Einheit. Ich habe mich bewusst dafür entschieden, diese Werte in einem einzigen API-Endpunkt zu bündeln, da sie ohnehin immer gleichzeitig aktualisiert werden müssen. Die aus dem Backend erhaltenen Daten werden in den Komponenten VegaViewer und Statistiken dargestellt.  Im Backend werden zunächst die Zeiträume definiert, für die die Daten zurückgegeben werden sollen. Anschließend wird für beide Jahre eine Filterfunktion aufgerufen, die die Daten basierend auf den aus dem Frontend übergebenen Parametern filtert. Die Daten stammen aus einer CSV-Datei, die ich mithilfe eines Python-Skripts aus den täglichen Wetterdaten seit 1992 zusammengestellt habe.  Mit den gefilterten Daten werden zwei Altair-Charts erstellt und zusammengefügt. Zusätzlich werden die Durchschnittswerte berechnet. Schließlich werden alle Ergebnisse als JSON-Response zurückgegeben. |
| Reflexionsfrage 6 (3) | Die Visualisierung hat zum Ziel, die Veränderung der ausgewählten Wetterparameter über einen Zeitraum darzustellen und sowohl kurzfristige Schwankungen als auch langfristige Trends erkennbar zu machen. Da die Grafik vor allem der Datenerkundung dient und dynamisch ist, habe ich einen generischen Titel gewählt, der den Fokus auf den allgemeinen Zweck der Analyse legt.  Für die Darstellung der Wetterdaten habe ich ein Liniendiagramm gewählt, da dieses am besten dazu geeignet ist, Trends im Zeitverlauf zu visualisieren und eine direkte Vergleichbarkeit der Daten aus verschiedenen Jahren zu ermöglichen.  Die Zeitdimension wurde auf der X-Achse abgebildet. Um den Vergleich von Wetterdaten aus verschiedenen Jahren zu ermöglichen, habe ich mich darauf beschränkt, nur den Monat und den Tag anzuzeigen.  Die Wetterwerte sind auf der Y-Achse dargestellt. Da diese Achse dynamisch durch die jeweiligen Parameter bestimmt wird, habe ich die Einheit als dynamische Achsenbeschriftung integriert, um die Lesbarkeit der Visualisierung zu verbessern.  Um die Linien der beiden Jahre voneinander zu unterscheiden, habe ich eine Farbskala (Viridis) verwendet, die sowohl gut unterscheidbar ist als auch zu meinem modernen Dashboard-Design passt. Die Farben der Linien werden zudem in der Legende erklärt.  Obwohl das Liniendiagramm gut für die visuelle Darstellung von Trends und Unterschieden geeignet ist, lässt es keine quantitativen Aussagen zu. Deshalb habe ich zusätzlich eine Tabelle integriert, die die durchschnittlichen Wetterdaten der beiden Jahre zeigt. Besonders wichtig ist die Differenz der beiden Jahre, die hervorgehoben wird: Sie wird rot eingefärbt, wenn der Wert im Vergleich zum anderen Jahr niedriger ist, und grün, wenn der Wert höher liegt. |
| Reflexionsfrage 7 (3) | Die grösste Herausforderung war es mit dem Datetime Datentyp zu jonglieren. Um die Daten zu Filtern musste ich nämlich zunächst das Start- und das Enddatum beider Jahre errechnen. Nachdem die Daten beider Jahre aber gefiltert waren konnte ich diese nicht in der Visualisierung anzeigen, weil sie sich an verschiedenen Punkten auf dem Zeitstrahl befanden. Der Trick war es nun beide Datensätze auf das gleiche Jahr zu setzen.  Dabei bin ich wiederum auf ein anderes Problem gestossen. Ich konnte nicht einfach mit einem Replace das Jahr in jeder Zeile des Datensatzes ändern, da ich sonst Problemen mit den Schaltjahren bekommen würde. Dies habe ich gemerkt, als ich plötzlich alle vier Jahre ein Error bekommen habe. Statt einen Replace habe ich beim Vergleichsjahr also ein DateOffset gemacht, welchem ich die Differenz der beiden Jahre mitgegeben habe.  Ausserdem war das Individualisieren der Charts schwierig da die Vega-Dokumentation nicht viel zu diesem Thema enthalten hat. Ein Problem war z.B. das ich viele Anpassungen wie z.B. die Titel Textgrösse anpassen vor dem zusammenführen der Charts gemacht habe. Ich habe erst später gemerkt das ich diese globale Anpassung, nachdem die Charts zusammengeführt wurden machen muss. |
| Reflexionsfrage 8 (3) | Da meine eine Applikation nur dazu dient Wetterdaten darzustellen und keine sensiblen Benutzerdaten enthält habe ich mich dazu entschieden nur minimale Sicherheitsmassnahmen zu implementieren.  Da die App keine Textinputfelder hat welche irgendwelche Injections zulassen, sondern nur Selects welche nur ausgewählte Werte zulassen, habe ich die serverseitige Datenvalidierung unterlassen.  Und um sicherzugehen das die API auch nur über mein Frontet erreicht werden kann habe ich habe ich meinen CORS-Header konfiguriert und habe als allowed-Origin einen Array angegeben, welcher alle meine vertrauenswürdigen Quellen beinhaltet. |

Erreichbare Gesamtpunktzahl: **36**

Punkte

