HOCHSCHULE HEILBRONN

Projektdokumentation

Fach:	Ausgewählte Kapitel des Mobile Computing
Semester:	SoSe 20
Challenge:	Data
Teammitglieder:	Gebhardt, Patrick (198948)
	Hinderberger, Lucas (198954)
	John, Timo (198320)
	Volkmann, Timo (199267)

Inhaltsverzeichnis

Installation und Deployment

Frontend

- node in Version 10.15.3 oder höher installieren
- “(cd frontend && npm i)” ausführen
- “(cd frontend && npm run start)” ausführen um einen Dev-Server auf http://localhost:4200 zu starten

Backend

- MariaDB aufsetzen
- node in Version 10.15.3 oder höher installieren
- Datenbank in .env oder Environment Variablen definieren. Siehe .env als Referenz.
- API-Keys für meteostat.net und Google Place API in .env oder Environment Variablen definieren
- “setup.sql” auf Datenbank für Beispieldaten ausführen
- “(cd backend && npm i)” ausführen
- “(cd backend && npm run start)” ausführen um einen Dev-Server auf http://localhost:3000 zu starten

Build Pipeline

Der gesamte Prozess des Kompilierens und Bereitstellen neuen Codes wurde von sowohl für Front- als auch Backend automatisiert. Dazu nutzen wir den CI-/CD Server Teamcity, dessen kostenfreie Nutzung lediglich durch die Anzahl an Build- Agents und Konfigurationen begrenzt ist. Weiterhin hat ein Teammitglied bereits Erfahrung mit dieser Software im professionellen Umfeld gesammelt.
Teamcity wurde mithilfe des Teamcity Plugins für GitLab (dort liegt unser Git-Repository) eingebunden, wodurch man unter anderem die Ergebnisse jedes Builds in Gitlab sieht und bei Fehlgeschlagenen Builds beispielsweise Merge-Requests auf den Git-Branch “develop” geblockt werden.
Weiterhin wird der Teamcity-Server bei jedem Hochladen von Änderungen benachrichtigt und kann dann mithilfe von Trigger-Regeln entscheiden, welche Build-Konfiguration für den Build benutzt werden soll. Wie der Name schon sagt konfiguriert diese Konfigurationsdatei die Schritte, die bei einem neuen Build absolviert werden müssen.
Wir nutzen zwei Build-Configs, eine für das Backend und eine für das Frontend.
Welche davon ausgeführt werden soll wird anhand des Ordners entschieden, in dem Daten verändert wurden. So liegen alle Dateien für das Frontend in einem Ordner namens `frontend`, und alle Dateien in einem Ordner namens `backend`. Werden nun Dateien für das Backend verändert führen wir nur die für das Backend nötigen Schritte aus.

Die CI-/CD-Pipelines wurde überwiegend mithilfe der Kotlin-like DSL (Domain specific language) konfiguriert, welche von Teamcity angeboten wird. Das Erlaubt größere Kontrolle

und das Verwenden gleicher Codeblöcke über verschiedene Konfigurationen. Diese DSL-Skripte finden Sie im Anhang.

Die Build-Pipelines besteht aus mehreren Schritten. Schlägt ein Schritt fehl, wenn beispielsweise aufgrund eines compile-time errors das kompilieren von Code fehlschlägt, bricht der build ab und wird als fehlgeschlagen gekennzeichnet.

Am Anfang jedes Builds erhöhen wir den build-counter, diesen verwenden wir später noch.

Backend

Da das Backend mit Node.js und dem Express-Framework umgesetzt ist, muss zunächst NodeJS auf dem Teamcity-Agenten installiert werden. Dazu nutzen wir den Node-Version-Manager (NVM), der es ermöglicht verschiedene Versionen von Node zu installieren und zu verwalten. Wir arbeiten mit Version v10.15.
Im zweiten Schritt installieren wir zunächst alle Node-Packages, mit deren Hilfe das Back-End umgesetzt wurde, diese werden in der Datei “Package.json” deklariert. Als nächstes wird das Projekt aus dem Source-Code gebaut, bei diesem Schritt fallen alle Compiler-Errors auf.
Statt die gebauten Dateien direkt auf den Server zu deployen haben wir uns dafür entschieden, Docker zu verwenden. Dazu verwenden wir seitens des Teamcity-Agent das offizielle Docker Plugin, das die Nutzung der wichtigsten Docker-funktionen erheblich erleichtert.
Docker ist eine Software zur Containervirtualisierung, die es nach dem Prinzip von Microservices erlaubt Anwendungen zu kapseln. Dazu muss ein sogenanntes Docker-Image erstellt werden, welches alle Abhängigkeiten beinhaltet, die zur Ausführung des Programms nötig sind. Dadurch kann man mehrere Container (Entsteht wenn man das Image ausführt) desselben Images nutzen, ohne dass es zu Konflikten kommt, was einfache Skalierbarkeit erlaubt.
Verschiedene Versionen des Images unterscheidet man mithilfe sogenannter Tags, diese kann man spezifizieren, wenn man ein Image herunterlädt.
Um im Falle eines logischen Fehlers auf vorherige Versionen zugreifen zu können, taggen wir das Image zunächst mit dem build-counter, anschließend als `latest`, das wird später noch notwendig.
Als letzter Schritt auf dem Teamcity laden wir das nun gebaute Image zu docker-hub hoch, eine sogenannte `docker-registry`. Diese speichert das Image in allen hochgeladenen Versionen (spezifiziert durch Tags), und ermöglicht den Zugriff auf diese Images via dem Befehl “docker-pull”. Unser Image ist unter `lhinderb/travoptijs:latest` zu finden. Auf unserem Server, läuft ebenfalls Docker, das mithilfe von Docker-Compose neben einer MariaDB-Instanz auch dieses Image hostet. Docker-Compose ermöglicht das gleichzeitige Hochfahren mehrerer Services mit einem Befehl in abhängigkeit voneinander, das Docker-Compose File finden Sie ebenfalls im Anhang.

Ein Service namens Watchtower ermöglicht das Überprüfen von Containern auf das Vorhandensein eines neuen Images. Hierdurch wird ein neues Image automatisch heruntergeladen und der Container neu gestartet.

Frontend

Wir verwenden den Nginx als Webserver und Reverse Proxy, dessen Konfiguration Sie ebenfalls im Anhang finden. Zur Realisierung von HTTPS haben wir den Certbot von Let's Encrypt genutzt, der das einfache Erstellen von Zertifikaten ermöglicht.

Wie bei der Backend-Konfiguration installieren wir auch hier node und die Abhängigkeiten des Projekts, bevor wir es kompilieren. Anschließend laden wir den Sourcecode mithilfe von SSH auf den in der Nginx-Konfiguration angegebenen Ordner auf unserem Server.
Hierzu hat der Teamcity-Agent einen Nutzer auf unserem Server bekommen, der außer auf diesen Ordner keinerlei Rechte hat. Mithilfe eines SSH-Keys kann sich der Teamcity-Agent ohne Username und Passwort für diesen Nutzer anmelden und die neuen Dateien hochladen.

Probleme machte vor allen Dingen die richtige Konfiguration des Teamcity-Agents, der ein inoffizielles Node-Plugin verwendet. Dieses erleichtert zwar die Verwendung von Node, machte allerdings Probleme in Verbindung mit der Verwendung des Docker-Plugins. Darum muss man nun beim Start des Containers die Flag “DOCKER_IN_DOCKER=start” angeben.

Technische Entscheidungen

Umsetzung als Web-App

Wir haben uns für die Entwicklung einer Web-App entschieden. Gegenüber einer nativen App besteht hier der Vorteil, dass diese einfach über den Browser aufzurufen ist und nicht erst vor der erstmaligen Nutzung installiert werden muss, was in vielen Fällen ein großes Hindernis für neue Anwendungen darstellt. Wir gehen davon aus, dass der durchschnittliche Nutzer die App nicht regelmäßig verwendet, sondern eher gelegentlich, wenn das Interesse besteht in den Urlaub zu gehen. Hier ist es für den User ein klarer Vorteil, wenn die App nicht zuvor installiert sein muss.
Durch moderne Frameworks lässt sich jedoch trotzdem eine Oberfläche erstellen, welche in dem jeweiligen Betriebssystem nativ wirkt und sich den Erwartungen entsprechend bedienen lässt.

Frontend

Angular

Angular ist ein Typescript basiertes Front-End Framework. Mit Angular können auch sehr komplizierte Anwendungen in den Browser verlegt werden. Das Framework hat sich in den letzten 4 Jahren zu einem der bekannten Fronten-End Frameworks am Markt entwickelt. In Verbindung mit Angular Material konnten wir vergleichsweise schnell ein Frontend bauen, welches sich für den Nutzer wie eine native Android App anfühlt. Durch die Modularität von Angular lassen sich viele Funktionen strukturieren. So beispielsweise der Einsatz eines Data-Services, der die Kommunikation mit dem Backend für den Rest der Anwendung abstrahiert.

Backend

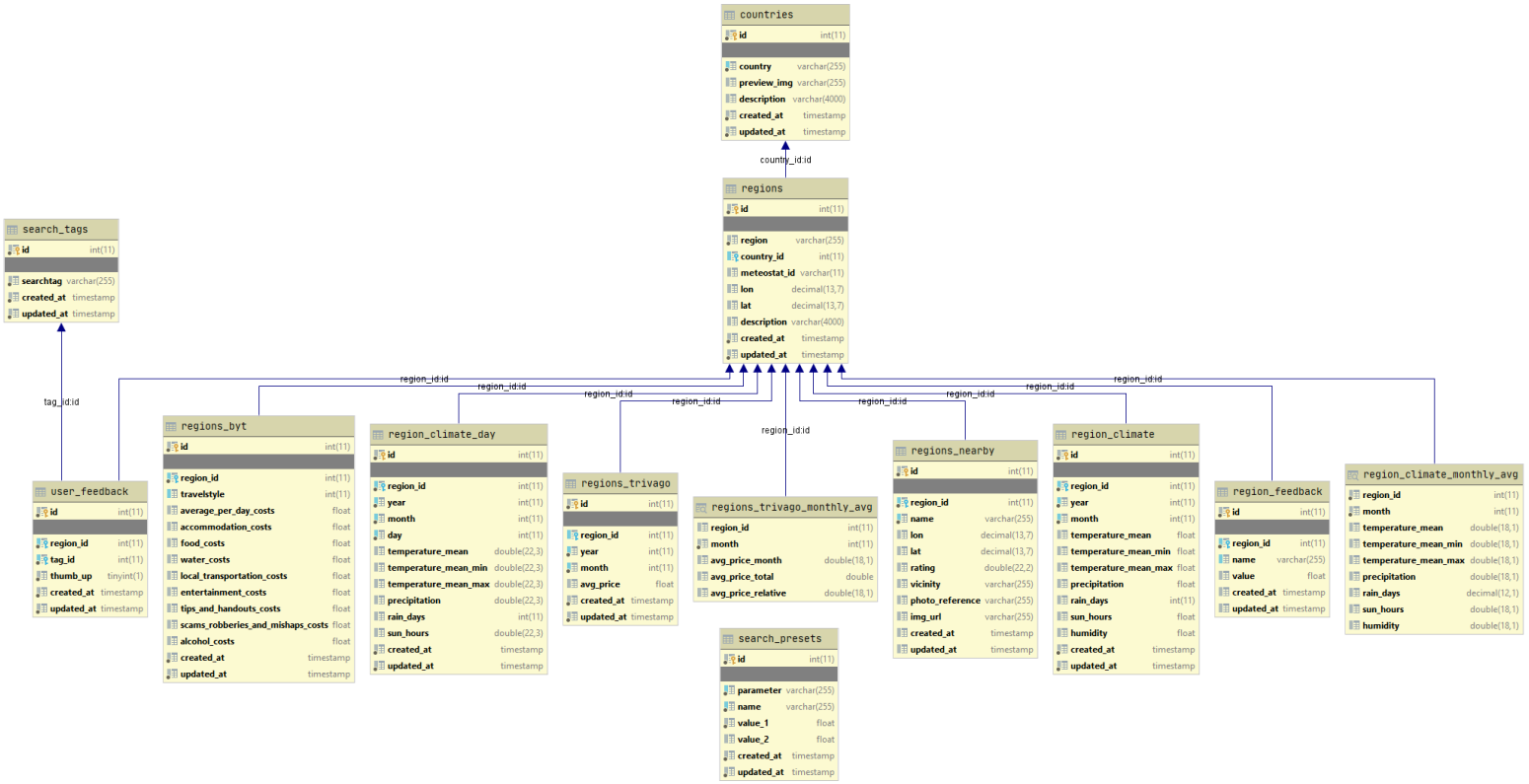
express.js

Express ist ein Node.js Framework zur schnellen Entwicklung von RESTful API Schnittstellen. Da wir bereits Erfahrungen damit hatten haben wir uns für dieses Framework entschieden. Desweiteren hat Express weitere Vorteile für die Entwicklung eines Prototypen:

- Wenig Aufwand beim Aufsetzen eines grundlegenden Backends.
- Ein Router bietet die Möglichkeit schnell und einfach strukturierte APIs zu erstellen.
- Das Framework ist gut dokumentiert und einfach zu lernen.
- Durch die weite Verbreitung von node.js können viele Komponenten als Package über den npm bezogen werden.

MariaDB

MariaDB ist eine der beliebtesten und weit verbreiteten SQL-Datenbanken. Durch Erfahrung im Umgang mit diesem Datenbanksystem durch das “Labor für Software-Projekte” konnten wir zügig die nötige Datenstruktur für die API-Schnittstelle bereitstellen.



Alternativen

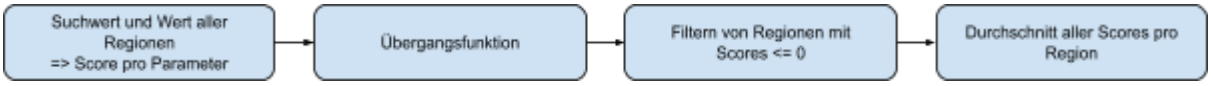
Ursprünglich war für das Backend Google Firebase geplant, jedoch wurde dies aufgrund fehlender Erfahrung mit der Plattform und der dadurch benötigten Einarbeitung aus Zeitgründen verworfen.
Desweiteren war als Datenbank auch MongoDB und weitere SQL Alternativen im Gespräch. Jedoch wurde aufgrund der allgemein vorhandenen Erfahrung bezüglich MariaDB diese als Datenbank gewählt.

API-Dokumentation

Die API-Dokumentation ist unter https://TravOpti.de/api/v1/doc/ erreichbar.

Allgemein

Scoring-Algorithmus



Da bei TravOpti im Kern alles um die Suche dreht, ist es besonders wichtig, sinnvolle Ergebnisse präsentieren zu lassen. Da wir Regionen jedoch nicht nur nach einem Parameter bewerten, sondern viele, inhaltlich komplett verschiedene Parameter berücksichtigen wollen, mussten wir ein System entwickeln, dass folgenden Ansprüchen gerecht wird:

- auf alle Arten von Parametern anwendbar
- fein justierbar / flexibel konfigurierbar
- sinnvolles Ranking auch mit mehreren Parametern
- einfache Handhabung aus User-Sicht

1. Scoring
Um die Parameter miteinander verrechnen und am Ende die Ergebnisse sortieren zu können, haben wir eine Scoring-Skala von 0 bis 10 eingeführt. Dabei wird der Abstand des gesuchten Wert mit dem Wert der Region für den gesuchten Zeitraum berechnet und mithilfe eines definierbaren Übergangsbereichs in den Wertebereich zwischen 0 und 10 transformiert.
Als Benutzer kann man einen Wert pro Parameter (z.B. Temperatur) an die Suche übergeben (Sweetspot). Alternativ dazu einen Wertebereich (z.B. von 12-14°C). Hat eine Region diesen Wert, wird die Region beim entsprechenden Parameter mit dem maximalen Score bewertet.

2. Übergangsbereiche
Wir definieren nun für jeden Parameter entsprechende Übergangsbereiche, in denen der Score zunächst linear abfällt. Per Definition sagt der Score 0 aus, dass sich der entsprechende Parameter außerhalb des Suchbereichs befindet und wird aus den Ergebnissen ausgeschlossen. Diese Übergangsbereiche wurden von uns selbst festgelegt und sollen sich nach den menschlichen Empfindungen richten.
Da das menschliche Empfinden oft nicht linear ist, kann man dem Algorithmus für jeden Parametertyp eine individuelle Übergangsfunktion übergeben:

- EaseIn *(nach Beispiel der CSS-Implementierung)*
- EaseOut *(nach Beispiel der CSS-Implementierung)*
- EaseInOut *(nach Beispiel der CSS-Implementierung)*
- Sigmoid-Funktion

3. Gesamtscore und Ergebnisse

Der Algorithmus soll alle Aspekte möglichst gleich gewichten und keine Annahmen bezüglich den Vorlieben potenzieller User treffen, daher werden grundsätzlich alle Parameter gleich gewichtet. Logischerweise fallen somit einzelne Parameter bei komplexeren Suchen deutlich weniger ins Gewicht als bei Suchen mit wenigen Parametern, was aber voraussichtlich der Diversität der Suchergebnisse zugute kommt.

Da es Parameter gibt, die sich überlagern bzw. eine enge Korrelation aufweisen, werden diese bei Auswahl miteinander verrechnet, bevor sie in den Durchschnitt der Scores einfließen. Damit werden Verfälschungen und ungewollte Gewichtungen vermieden.

Damit der Suchende einzelne Parameter seiner Suche selbst in den Fokus stellen kann, bieten wir Ihm für alle wichtigen Parameter eine Sortierfunktion an. Somit ist es möglich innerhalb der Suchergebnisse nochmals zu priorisieren und dem Benutzer insgesamt größtmögliche Flexibilität bei einfachster Bedienung zu bieten.

Datenquellen

Um unsere App mit den gewünschten Features auszustatten, sind verschiedene Arten von Daten notwendig:

1. Wetter/Klimadaten (monatlich)
2. Preise für Unterkünfte (monatlich)
    a. Preise für Kategorie Budget
    b. monatliche Preistendenzen
3. tägliche Lebenshaltungskosten
4. Attribute zu einzelnen Regionen (kategorisierbar, bewertbar)

Prototyp
Für den Prototyp haben wir uns zunächst einen beschränkten Datensatz an Regionen erstellt, basierend auf dem Hotelpreisindex von Trivago. Damit hatten wir eine ausreichend diverse Datengrundlage um ein Proof-of-Concept erstellen zu können. Zudem konnten wir aus diesen Daten schon die Preistendenzen pro Region und Monat extrahieren.

Für jede dieser Regionen haben wir uns nun monatliche Wetterdaten über meteostat.net besorgt. Die Daten sind kostenlos zugänglich und schon sehr nah an der Form, wie wir sie benötigen. Nachteil ist, dass die Daten nicht kommerziell genutzt werden dürfen. Für ein erstes PoC ist dies jedoch noch kein Problem. Zudem sind alle Quellen für die öffentlich und frei zugänglichen Daten auf der Webseite angegeben.

Daten für Lebenshaltungskosten zu finden stellte sich am schwierigsten heraus, da diese Daten sehr subjektiv sind und mit entsprechendem Lebensstil in der gleichen Region sehr unterschiedlich ausfallen können. Einen sehr guten Ansatz zeigte hier budgetyourtrip.com. Die bereitgestellten Daten dieser Seite sind Erfahrungswerte von Nutzern der Seite, die beim anlegen der Daten ihren Reisestil (Budget, Mid-Range, Luxury) mit angeben und sich so ein sehr realitätsnahes Bild abzeichnen lässt. Zudem sind die Preise in sehr viele

Kategorien aufgeschlüsselt, wie “öffentliche Verkehrsmittel”, “Kosten für Essen”, “Wasser”, “Unterkunft” und mehr.
Dies ermöglichte uns, für den Prototyp realistische Preise anzugeben, ohne weitere Schnittstellen nutzen zu müssen oder selbst Daten zu sammeln.

Bleiben noch die Attribute zu den Regionen. Geplant ist hier auf längere Sicht, Meinungen von Nutzern zu sammeln und die Regionen damit kategorisierbar zu machen. Legt ein Nutzer beispielsweise Wert auf schöne Strände bzw. Strandurlaub, kann er bei der Suche diese Information angeben. Zudem erhält jede Region Schlagworte (“Tags” in unserer Terminologie), die andere Nutzer mit Daumen hoch oder runter bewerten können. besitzt eine Region mehr positive als negative Bewertungen für einen Tag, wird sie in der entsprechenden Suche als potentielles Ergebnis gelistet. Der Score steigt dann weiter mit dem Verhältnis von positiven zu negativen Bewertungen. Somit lassen sich die Regionen dann beliebig kategorisieren und gezielt nach Tags auffindbar machen. Für den Prototyp haben wir die Regionen nach bestem Gewissen mit einem vordefinierten Satz an Tags ausgestattet. Die Daten sind daher bisher nicht repräsentativ.

Zukünftige Datenbeschaffung
Bisher ging es nur darum, ein Proof-of-Concept bzw. einen funktionalen Prototyp mit Daten zu versorgen. Für den Live-Einsatz sind die Daten jedoch so nur eingeschränkt nutzbar.

Wie bereits im Prototyp-Abschnitt erläutert werden statt unseren Schätzungen für einen festen Tag-Satz in Zukunft echte Daten von echten Reisenden für die Verschlagwortung verwendet. Dafür bieten wir dann ein leicht zugängliches Userinterface an, welches die Nutzer dazu animiert sein Feedback schnell und anonym abzugeben.

Für das Klima ist geplant kommerzielle Wetter- und Klimadaten einzukaufen. Zum einen können wir dann auch Wettervorhersagen für einen gewünschten Zeitraum mit anbieten und zum anderen müssen wir uns dann nicht um die meteorologische Expertise und den Aufwand für die Akkumulierung der weltweiten Daten sorgen, die notwendig wären, ein solches System selbst zu unterhalten.

Preise für Unterkünfte werden wir auch über einen kommerzielle Partner beziehen, welcher idealerweise gleichzeitig eine Schnittstelle für Buchungen anbietet. Somit können wir dann auf echte Preise zurückgreifen. Durch regelmäßiges Abrufen und Speichern der Preise, können wir für verschiedene Prioritäten (Budget-/Luxusunterkunft, Preis-Leistung, ...) für jede Region ein sehr akkurates Bild zeichnen und unseren Kunden sogar helfen die günstigste Zeit zum Buchen zu finden.

Für die Lebenshaltungs- und andere wichtige Kostenfaktoren, werden wir auch später budgetyourtrip.com als Datenquelle behalten. Die Kosten für diese Daten sind mit 200€ im Monat sehr überschaubar.
Trotz dieser Entscheidungen behalten wir uns vor auf lange Sicht und bei entsprechendem Erfolg und Wachstum des Projekts eigene Datensammlungen für Klima und Kostenfaktoren aufzubauen um Abhängigkeiten zu externen Lieferanten abzubauen.

Bedienungsanleitung

Bedienungsanleitung für die TravOpti-Webanwendung, die unter https://travopti.de/ erreichbar ist.

Startseite

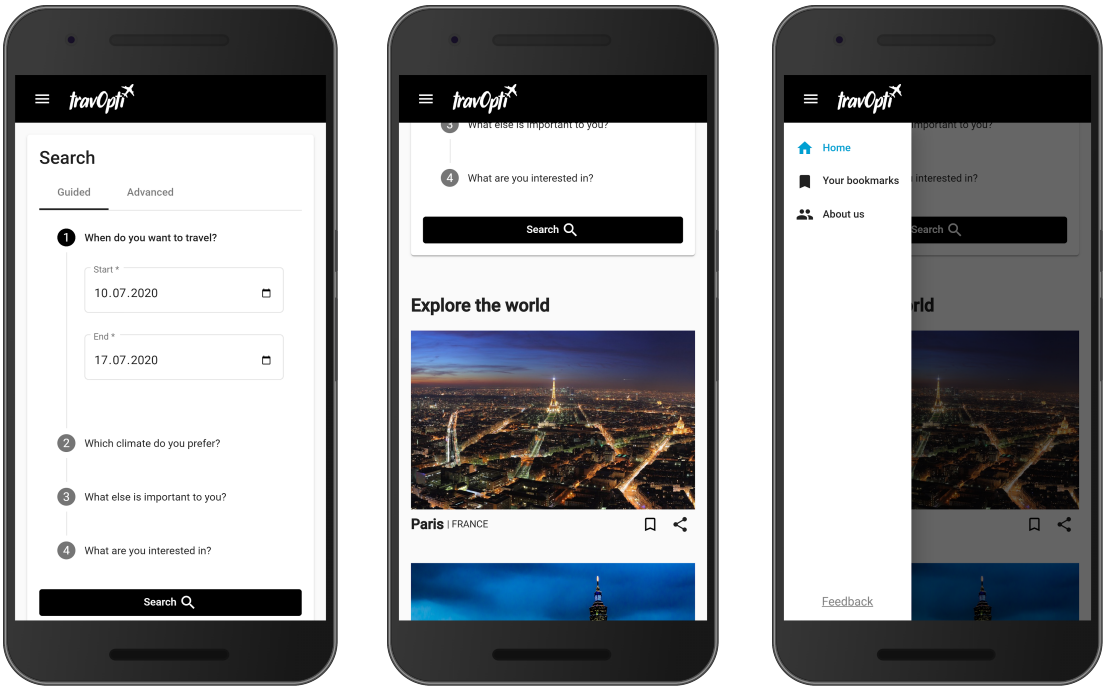


Abb. 1: TravOpti Startseite

Die Startseite von TravOpti besteht aus zwei Teilen. Im oberen Teil ist die Suchmaske zu sehen, hier kann sowohl im geführten als auch im erweiterten Suchmodus gesucht werden. Unter der Suche befindet sich der Abschnitt “Explore the world”. In diesem Abschnitt werden bei jedem Laden der Seite zufällig zehn Regionen ausgewählt und angezeigt. So können Sie auch ohne konkrete Anforderungen durch die ersten Regionen streifen.

Um die anderen Seiten der Webanwendung aufzurufen, befindet sich in der Toolbar der Mobilversion das bekannte Menü-Icon (≡). Ein Tippen führt zum Öffnen der Sidebar, in welcher Sie die Möglichkeit haben sich Ihre Lesezeichen anzusehen oder Infos über das Team abzurufen. Auf der Desktopversion wird die Sidebar dauerhaft angezeigt.

Suche

Die Suchfunktion ist das Hauptfeature von TravOpti. Hier wird Ihnen die Möglichkeit geboten die vorhanden Regionen aufgrund Ihrer Angaben bewerten zu lassen. So kann auch bei

vielen Zielen das Richtige ausgewählt werden. In der Suche können viele Parameter angegeben werden, darunter sind unter anderem das Klima unter Berücksichtigung des Reisezeitraums, durchschnittliche Kosten für den Aufenthalt sowie von anderen Nutzern bewertete Kategorien. Um jedem Nutzer die Eingabe seiner Wünsche möglichst einfach zu machen, ist die Suche über zwei Masken durchführbar.

Geführte Suche

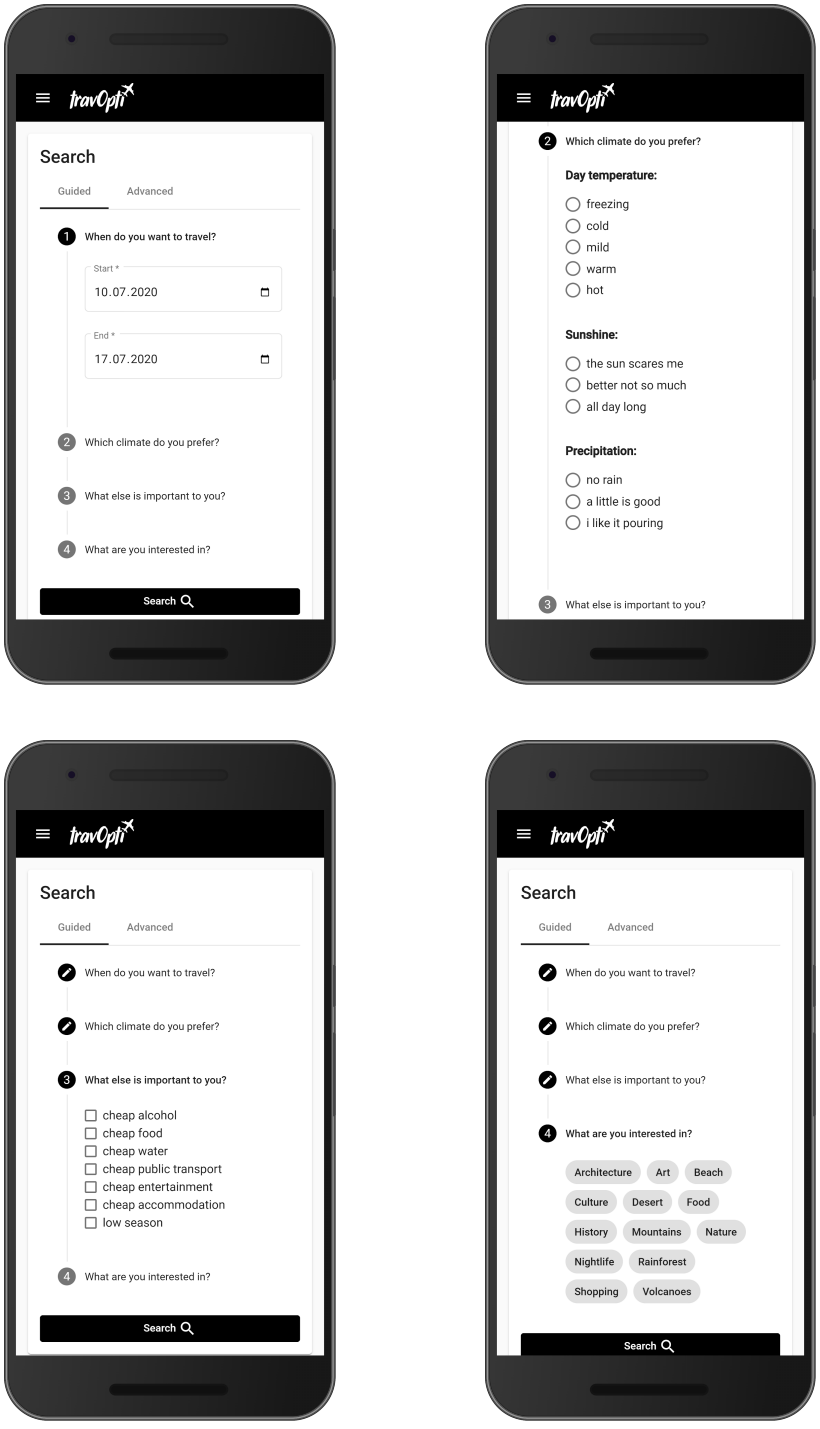


Abb. 2: Geführte Suche

Die geführte Suche soll dem durchschnittlichen Nutzer das Bedienen der Suche vereinfachen. Dazu werden Sie durch vier vorgegeben Schritte geführt. Im ersten Schritt muss der Reisezeitraum eingegeben werden. Die Reisedauer darf dabei nicht kürzer als ein Tag sein. Der Reisezeitraum wird dabei sowohl für die Berechnung der Schätzung für die Reisekosten als auch für die Auswahl der Wetterdaten verwendet.
Im zweiten Schritt wird Ihnen die Möglichkeit gegeben ein gewünschtes Klima für Ihren Zielort anzugeben, dabei haben Sie die Auswahl je einer Option in den Kategorien Temperatur, Sonnenstunden und Niederschlag.
Im nächsten Abschnitt können weitere Präferenzen der Suche mitgegeben werden. Hier ist eine Mehrfachauswahl aus vielen Optionen möglich, zum Beispiel günstige Preise für verschiedene Aktivitäten oder der Wunsch außerhalb der Saison zu reisen.
Der letzte Abschnitt bietet Ihnen eine Reihe von Tags. Jede Region kann in Zukunft von anderen Nutzern in verschiedenen Gesichtspunkten positiv oder negativ bewertet werden. Sie können nun mehrere Tags auswählen, die Ihnen wichtig sind. Tags können sehr viele Aspekte abdecken, beispielsweise Landschaft oder Kultur.

Bitte beachten Sie, dass alle Angaben außer die Reisedaten nicht verpflichtend sind. Die besten Ergebnisse werden Sie erhalten, wenn Sie nur die Dinge auswählen, die Ihnen wirklich wichtig sind.

Erweiterte Suche

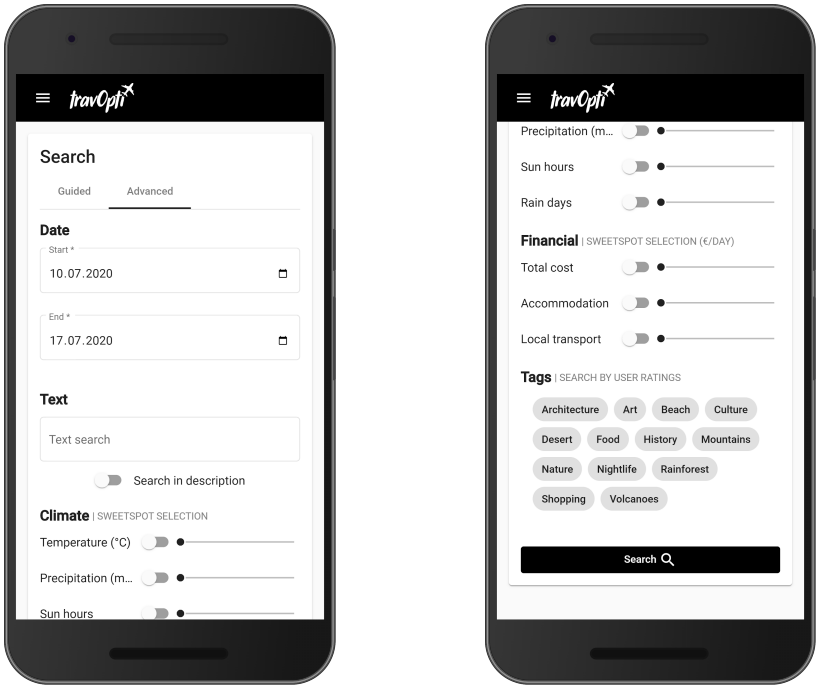


Abb. 3: Erweiterte Suche

Die erweiterte Suche kann durch ein Tippen auf “Advanced” in jeder Suchmaske ausgewählt werden. Sie soll erfahrenen Nutzern die Möglichkeit bieten Beschränkungen der geführten Suche zu umgehen und völlig frei nach bestimmten Werten suchen zu können.

Zunächst muss analog zur geführten Suche der Reisezeitraum eingegeben werden. Der erste große Unterschied zur geführten Suche bietet die Textsuche, hier kann entweder nur der Regionsname und das Land oder zusätzlich auch die Beschreibung der Region durchsucht werden. Wird das Textfeld leer gelassen, wird keine Textsuche durchgeführt.

Im nächsten Abschnitt haben Sie die Möglichkeit alle Suchparameter über Slider einzustellen. Zu Beginn sind alle Slider deaktiviert, es wird also bei der Suche kein Wert berücksichtigt. Um der Suche einen Parameter hinzufügen, kann der entsprechende Slider über den nebenstehenden Schalter aktiviert werden.

Zum Schluss können auch bei der erweiterten Suche die Tags ausgewählt werden.

Suchergebnisse

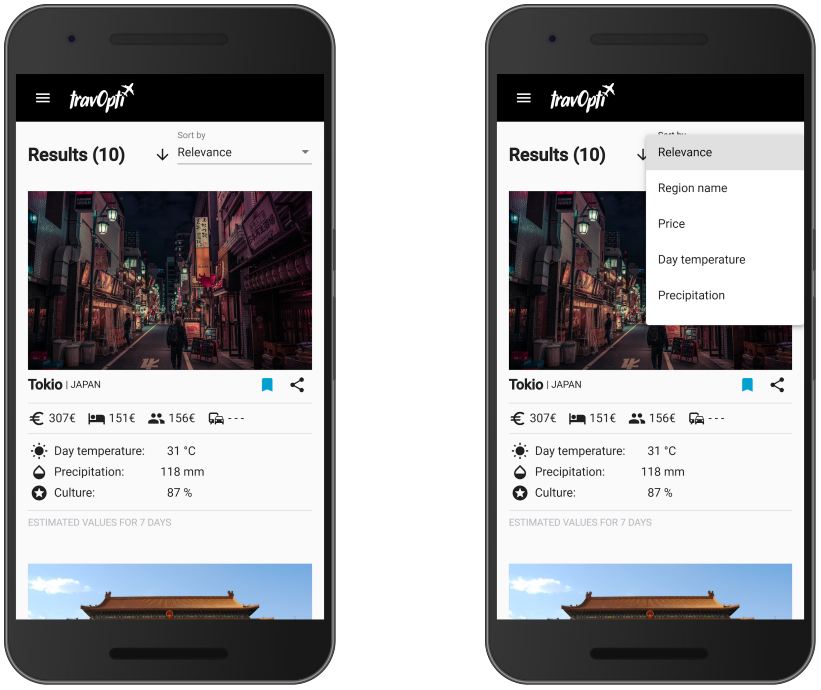


Abb. 4: Suchergebnisse

Nach jeder ausgeführten Suche werden Sie auf eine Seite weitergeleitet, welche die Suchergebnisse anzeigt. Auf dieser Seite bekommen Sie einen ersten Überblick über die Ergebnisse Ihrer Suche. Die Ergebnisse sind dabei standardmäßig nach absteigender Relevanz sortiert. Die Seite versucht hierbei alle eingegebenen Parameter zu berücksichtigen (siehe Abschnitt: Scoring-Algorithmus). Durch ein Tippen auf “Relevance” kann jedoch auch nach Name, Preis oder alle in der Suche enthaltenen Parametern sortiert

werden. Mit Hilfe des angrenzenden Pfeils kann die Sortierungsrichtung geändert werden, dabei ist bei jeder Sortierungsoption bereits die sinnvollste Richtung hinterlegt.

Jede Ergebnis-Karte enthält nun neben dem Namen und Land der Region im Abschnitt darunter auch die aufsummierten Preise für die Gesamtkosten, die Unterkunft und andere Ausgaben. Der letzte Punkt in der Zeile soll zukünftig die Kosten für An- und Abreise anzeigen, momentan führt ein Antippen zur Weiterleitung auf die Seite eines externen Anbieters. Im letzten Abschnitt sind die Durchschnittswerte der in der Suche enthaltenen Parameter für den ausgewählten Reisezeitraum zu finden.

Debugging

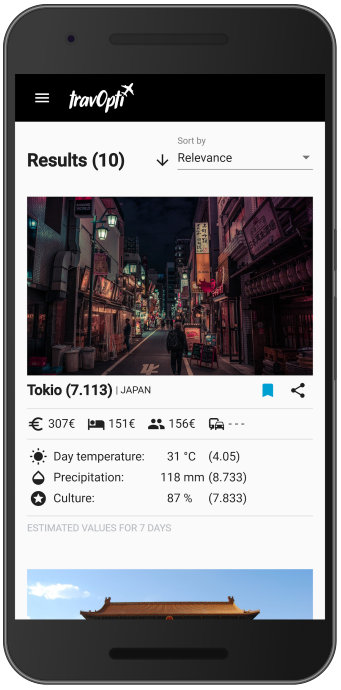


Abb. 5: Debug-Ansicht

Die Debug-Ansicht erlaubt das Einsehen der vom System vergebenen Scores (siehe Abschnitt: Scoring-Algorithmus). Im Debug-Modus wird sowohl der Gesamtscore der Region für die aktuelle Suchanfrage, als auch die Einzelscores der Suchparameter angezeigt.

Um in die Debug-Ansicht zu gelangen muss in der Adresszeile Ihres Browsers folgendes angehängt werden, wenn Sie sich auf der Suchergebnisseite befinden: “`&debug=true`”.

Detailseite

Abb. 6: Detailseite

Das Aufrufen der Detailseite kann durch das Antippen eines Suchergebnisses, einer Region aus dem Abschnitt “Explore the world” (siehe: Startseite) oder eines Lesezeichens erfolgen. Die Detailseite soll dem Nutzer einen Einblick in die Region bieten und ihm genauere Informationen zu Preisen, Wetter und Sehenswürdigkeiten liefern.

Beschreibung

Die Beschreibung gibt Ihnen einen kurzen Einblick in die Region. Sie enthält geographische, kulturelle oder historische Informationen. Um Platz zu sparen wird die Beschreibung zu Beginn auf eine bestimmte Länge gekürzt, durch das Antippen von “more” kann die komplette Beschreibung eingesehen werden.

Tagespreise

Unterhalb der Beschreibung finden Sie die Aufschlüsselung der Kosten für die jeweilige Region. So kann individuell entschieden werden welche Preisanteile die eigene Reise betreffen.

Monatliche Daten

Jede Region verfügt über monatliche Daten, wie die Preisabweichung gegenüber des Durchschnittspreises oder verschiedene Klimadaten. Diese werden dem Nutzer in mehreren Diagrammen dargestellt die in verschiedenen Tabs ausgewählt werden können. Um dem Nutzer bei der Einschätzung der Daten zu helfen sind alle Klimadiagramme in der gleichen Skala bei allen Regionen abgebildet.

Orte

Die letzte Sektion auf der Detailseite ist eine Auswahl von Orten die Sie in der jeweiligen Region besuchen können. Diese Übersicht soll Ihnen zusätzlich zur Beschreibung helfen sich einen Eindruck von der Region machen zu können. Das Antippen eines Ortes führt aktuell zu Weiterleitung auf eine externe Karte.

Region teilen

Wenn Sie eine interessante Region gefunden haben, wollen Sie diese eventuell mit anderen Personen teilen. TravOpti macht Ihnen dieses Vorgehen so einfach wie möglich. Überall in der Anwendung finden die zu jeder Region einen Teilen-Button ( ). Durch das Berühren auf der mobilen Version der Webanwendung wird das native Teilen-Menü ihres Mobiltelefons aufgerufen. Dadurch können Sie die Region in wenigen Schritten über installiere Messenger auf ihrem Telefon teilen. In der Desktopversion erscheint ein Pop-up, das Ihnen erlaubt den Link zur Region in die Zwischenablage zu kopieren.

Lesezeichen

Die Lesezeichen-Funktion erlaubt Ihnen sich interessante Regionen in der Webanwendung zu merken. So können sie sich Regionen erneut ansehen, ohne die Suche erneut ausführen zu müssen. Um sich ihre Lesezeichen anzusehen wechseln Sie über die Sidebar (siehe Startseite) auf die Seite “Bookmarks”. Da die Lesezeichen in ihrem Browser gespeichert werden, bleiben diese auch erhalten wenn Sie die Seite schließen und zu einem späteren Zeitpunkt erneut aufsuchen, solange sie Ihre Browserdaten nicht gelöscht haben oder nicht den Inkognitomodus nutzen.

Lesezeichen hinzufügen oder löschen

Analog zum Teilen-Button ist auch der Lesezeichen-Button (🔖) bei jeder Region oder jedem Suchergebnis zu finden. Wird er berührt so wechselt er vom leeren Symbol (🔖) zum ausgefülltem Symbol (🔖). Sie haben nun erfolgreich eine Region zu Ihren Lesezeichen hinzugefügt, berühren Sie den Lesezeichen-Button erneut wechselt er wieder zu seiner ursprünglichen Erscheinung und die Region ist nicht länger in Ihren Lesezeichen.

Anhang

Artefakte

- Link zu Trello:
  https://trello.com/b/E2zfCOgF/ppm
- Link zu Artefakten in Google-Drive, inkl. Meeting Notes und Meilensteine
  https://drive.google.com/drive/folders/1NF_nwOnlpXJzwc7PTXb_5MqZTBvVFK6Z?usp=sharing

Datenquellen

- Buchungs- und Hotelschnittstellen:
  https://www.partners.skyscanner.net/affiliates/affiliate-products
- Kommerzielle Klima und Wetterdaten:
  https://openweathermap.org/price
- Trivago Hotelpreisindex:
  https://businessblog.trivago.com/de/trivago-hotel-price-index/
- Meteostat:
  https://meteostat.net/en
- budgetyourtrip.com:
  https://www.budgetyourtrip.com/

Nginx-Konfiguration für travopti.de

```
server {

  server_name travopti.de;

  root /var/www/travopti.de;

  location / {
    try_files $uri /index.html =404;
  }

  location /api {
    proxy_pass http://127.0.0.1:3000/api;
  }

    listen [::]:443 ssl; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/travopti.de/fullchain.pem; # managed $
    ssl_certificate_key /etc/letsencrypt/live/travopti.de/privkey.pem; # manage$
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
  listen 80;
  listen [::]:80;

  server_name travopti.de;

  if ($host = travopti.de) {
      return 301 https://$host$request_uri;
  } # managed by Certbot

  return 404; # managed by Certbot
}
```

Docker-compose-yml für das Frontend

```
version: '3'
services:
	travopti:
	image: lhinderb/travoptijs:latest
	container_name: travopti
	ports:
		- 3000:3000
	restart: unless-stopped
	environment:
		# - DB_HOST=172.26.0.1
		- DB_HOST=maria
		- DATABASE=travopti
	depends_on:
		- maria
	networks:
		- offline

	maria:
	image: mariadb/server:10.3
	container_name: maria
```

DSL-Skript für das Frontend

Frontend:

```
package CcData.buildTypes

import jetbrains.buildServer.configs.kotlin.v2019_2.*
import jetbrains.buildServer.configs.kotlin.v2019_2.buildSteps.dockerCommand
import jetbrains.buildServer.configs.kotlin.v2019_2.triggers.vcs

object CcData_Frontend : BuildType({
       templates(NodeJs)
       id("Frontend")
       name = "Frontend"

       allowExternalStatus = true
       enablePersonalBuilds = false
       type = BuildTypeSettings.Type.DEPLOYMENT
       maxRunningBuilds = 1
       publishArtifacts = PublishMode.SUCCESSFUL
       steps {
       step {
              name = "NPM build"
              id = "RUNNER_2"
              type = "jonnyzzz.npm"
              param("teamcity.build.workingDir", "frontend")
              param("npm_commands", """
              install
              run build
              """.trimIndent())
       }
       dockerCommand {
              name = "Docker build"
              id = "RUNNER_3"
              enabled = false
              commandType = build {
                   source = file {
                     path = "Dockerfile"
              }
              namesAndTags = "travoptijs:0.0.%build.counter%"
              commandArgs = "--pull"
              }
       }
       step {
              name = "Upload"
             id = "RUNNER_6"
              type = "ssh-deploy-runner"
              param("jetbrains.buildServer.deployer.username", "tc-deploy")
              param("jetbrains.buildServer.sshexec.port", "2222")
              param("teamcitySshKey", "ci.id_rsa")
              param("jetbrains.buildServer.deployer.sourcePath",
"frontend/dist/frontend/**")
              param("jetbrains.buildServer.deployer.targetUrl",
"172.17.0.1:/var/www/travopti.de")
```

```
            param("jetbrains.buildServer.sshexec.authMethod", "UPLOADED_KEY")
            param("jetbrains.buildServer.deployer.ssh.transport",
"jetbrains.buildServer.deployer.ssh.transport.scp")
      }
      }
      triggers {
      vcs {
            id = "vcsTrigger"
            triggerRules = "+:frontend/**"
            branchFilter = ""
            perCheckinTriggering = true
            enableQueueOptimization = false
      }
      }

      disableSettings("RUNNER_4")
})
```

DSL-Skript für das Backend

```
package CcData.buildTypes

import jetbrains.buildServer.configs.kotlin.v2019_2.*
import jetbrains.buildServer.configs.kotlin.v2019_2.buildFeatures.dockerSupport
import jetbrains.buildServer.configs.kotlin.v2019_2.buildSteps.dockerCommand
import jetbrains.buildServer.configs.kotlin.v2019_2.triggers.VcsTrigger
import jetbrains.buildServer.configs.kotlin.v2019_2.triggers.vcs

object CcData_Backend : BuildType({
	templates(NodeJs)
	id("Backend")
	name = "Backend"

	enablePersonalBuilds = false
	type = BuildTypeSettings.Type.DEPLOYMENT
	maxRunningBuilds = 1

	steps {
	step {
		name = "NPM build"
		id = "RUNNER_2"
		type = "jonnyzzz.npm"
		param("teamcity.build.workingDir", "backend")
		param("npm_commands", """
		install
		build
		""".trimIndent())
	}
	dockerCommand {
		name = "Docker push"
		id = "RUNNER_4"
		commandType = push {
		namesAndTags = "lhinderb/travoptijs"
		}
	}
	dockerCommand {
		name = "Docker build"
		id = "RUNNER_3"
		commandType = build {
		source = file {
			path = "Dockerfile"
		}
		namesAndTags = """
			lhinderb/travoptijs:0.0.%build.counter%
			lhinderb/travoptijs:latest
		""".trimIndent()
		commandArgs = "--pull"
		}
	}
	step {
		name = "Build frontend"
```

```
            id = "RUNNER_5"
            type = "jonnyzzz.npm"
            enabled = false
            param("teamcity.build.workingDir", "frontend")
            param("npm_commands", """
            install
            run build
            """.trimIndent())
        }
        }

        triggers {
        vcs {
            id = "vcsTrigger"
            quietPeriodMode = VcsTrigger.QuietPeriodMode.USE_DEFAULT
            triggerRules = "+:backend/**"
            branchFilter = ""
            perCheckinTriggering = true
            groupCheckinsByCommitter = true
            enableQueueOptimization = false
        }
        }

        features {
        dockerSupport {
            id = "DockerSupport"
            loginToRegistry = on {
            dockerRegistryId = "PROJECT_EXT_2"
            }
        }
        }
})
```