DTO’s

DTOs dienen dazu, Wetterdaten von einem externen Wetterdienst abzurufen, in einer strukturierten Form zu speichern und sie in unserer Anwendung zu verwenden. Sie erleichtern die Verarbeitung und Darstellung von Wetterinformationen und ermöglichen es unserer Anwendung, auf genaue Wetterdaten zuzugreifen und diese an Benutzer anzuzeigen.

AlertDTO:

Einsatz zur Verarbeitung von Wetterwarnungen aus API-Antworten. Beispielhafte Nutzung beinhaltet das Auslesen von Warnungsdetails wie Ereignisname, Dauer und Beschreibung.

CurrentWeatherDTO:

Die `CurrentWeatherDTO`-Klasse dient dazu, die Antwort auf einen API-Aufruf zur Abfrage aktueller Wetterdaten zu modellieren. Sie enthält verschiedene Attribute, um Informationen über die aktuellen Wetterbedingungen für einen bestimmten Ort zu speichern.

DailyTemperatureAggregationDTO:

Die DailyTemperatureAggregationDTO-Klasse dient dazu, die Antwort auf einen API-Aufruf zur Abfrage von täglichen Temperaturdaten zu modellieren. Sie speichert verschiedene Temperaturen(morning temperature,evening temperature etc.) im Laufe eines Tages für einen bestimmten Ort.

DailyWeatherDTO:

Die DailyWeatherDTO-Klasse dient dazu, die Antwort auf einen API-Aufruf zur Abfrage täglicher Wetterdaten zu modellieren. Sie enthält verschiedene Attribute(sunrise,temp,feels like,humidity etc.), um Informationen über die Wetterbedingungen über den Verlauf eines Tages für einen bestimmten Ort zu speichern

HourlyWeatherDTO:

Die *HourlyWeatherDTO*-Klasse dient dazu, die Antwort auf einen API-Aufruf zur Abfrage stündlicher Wetterdaten zu modellieren. Sie enthält verschiedene Attribute, um Informationen über die Wetterbedingungen für jede Stunde des Tages für einen bestimmten Ort zu speichern.

PrecipitationByTimestampDTO:

Die *PrecipitationByTimestampDTO*-Klasse dient dazu, die Antwort auf einen API-Aufruf zur Abfrage von Niederschlagsdaten zu modellieren. Sie speichert Informationen über die Niederschlagsmenge in Millimetern pro Stunde (mm/h) zu einem bestimmten Zeitpunkt

TemperatureAggregationDTO:

Die *TemperatureAggregationDTO*-Klasse dient dazu, die Antwort auf einen API-Aufruf zur Abfrage von Temperaturdaten zu modellieren. Sie speichert Informationen über die Temperaturwerte für verschiedene Tageszeiten in Grad Celsius.

CurrentAndForecastAnswerDTO:

Die CurrentAndForecastAnswerDTO-Klasse ermöglicht es Entwicklern, umfassende Wetterdaten für einen Ort abzurufen und in ihrer Anwendung zu verwenden. Sie wird beim ersten API-Aufruf verwendet, um aktuelle Wetterdaten und Vorhersagen zu erhalten. Sie dient als Container für andere DTOs(HourlyWeather-,DailyWeather- ,Alert- and PrecipiationDTO), die spezifische Aspekte der Wetterdaten repräsentieren

DailyAggregationDTO

Die DailyAggregationDTO-Klasse wird verwendet, um tägliche Wetteraggregationen für einen bestimmten geografischen Ort abzurufen und zu speichern, insbesondere für den Urlaubs-Teil Ihrer Anwendung. Diese Klasse wird beim zweiten API-Aufruf verwendet, um spezifische Wetterdetails für einen Urlaubsort zu erhalten

GeocodingDTO:

Es handelt sich da um eine effiziente Möglichkeit, Geocoding-Daten von der API abzurufen, in Java darzustellen und zu verarbeiten.. Dieses Objekt enthält den Ortsnamen, lokale Namen in verschiedenen Sprachen, geografische Koordinaten (Breite und Länge), das Land und gegebenenfalls den Bundesstaat oder die Region.

WeatherDTO:

Dieses DTO ermöglicht es in der Anwendung, Wetterinformationen von einer API abzurufen und in einer strukturierten Form zu speichern. Die Informationen können dann verwendet werden, um dem Benutzer aktuelle Wetterbedingungen anzuzeigen oder für andere Wetterbezogene Funktionen

Deserialisation:

→ PrecipiationDeseralizier: Dieser Deserializer wandelt Niederschlagsdaten aus einer speziellen JSON-Struktur in ein Java-Objekt um. Dies ist hilfreich, da wir die empfangenen Wetterdaten so in ein für unsere Anwendung geeignetes Format bringen können. Dadurch können wir die Wetterdaten effizient nutzen und für verschiedene Zwecke in unserer App verwenden.

→ WeatherDeseralizier:

Die `WeatherDeserializer`-Klasse ermöglicht es unserer Anwendung, die ungewöhnlich verpackten Wetterdaten aus der JSON-Antwort des Servers in ein geeignetes Format für die weitere Verarbeitung umzuwandeln. Diese Anpassung ist wichtig, um sicherzustellen, dass die Wetterdaten effizient in unserer Anwendung genutzt werden können.

Services:

→ GeocodingApiRequestService:

Diese Klasse dient dazu, Geocoding-Daten für einen bestimmten Ort abzurufen. Sie verwendet eine externe Geocoding-API, um Informationen über den angegebenen Ort zu erhalten. Die Klasse bietet Methoden zum Codieren des Ortsnamens, zum Aufbau der API-URL, zum Durchführen des API-Aufrufs und zur Verarbeitung der API-Antwort. Sie enthält auch eine Fehlerbehandlungsfunktion, um auf API-Fehler zu reagieren.

→ WeatherApiRequestService:

Diese Klasse dient dazu, Wetterdaten von einer externen Wetter-API abzurufen. Sie ermöglicht das Abfragen von aktuellen Wetterdaten sowie Vorhersagen für einen bestimmten geografischen Ort

=> retrieveCurrentAndForecastWeather Funktion: stellt eine API-Anfrage an die Wetter-API, um aktuelle Wetterdaten und Vorhersagen für einen bestimmten Ort anhand von Längen- und Breitengradkoordinaten abzurufen. Sie überprüft den API-Antwortstatus und wirft eine WeatherApiException, wenn ein Fehler auftritt.

=> retrieveDailyAggregationWeather Funktion: macht genau das gleiche wie die 1. Funktion aber ruft die tägliche aggregierte Wetterdaten, nicht current. Diese Methode wird dann später für Vacation Planing nützlich sein.

User-Management:

Dient dem Abspeichern und dem Manipulieren der Daten für die Benutzer der Webapp.

Entities:

Userx : enthält alle Daten, die zu einem gewissen User gespeichert werden können.
Enthält auch die Verweise auf die anderen Entitys die in einer direkten Beziehung zu einem User stehen.

UserxRole ist ein ENUM mit der Auflistung der verschiedenen Rollen, die ein User annehmen kann.

Repository:

UserxRepository : Über dieses Repository wird auf die verschieden Daten für die User zugegriffen.

Service:

UserxService: Enthält die gängigsten Methoden zum Speichern, laden, bearbeiten der User aus der Db.
Hat des Weiteren die Logik um einen neuen User anzulegen wenn dieser noch nicht in der Db ist und man sich neu Registriert.

Controller:

UserDetailController: Enthält die Schnittstellen zu dem Userxservice sowie einige Methoden für die Validierung von Eingabe Daten die einen Zugriff auf die Daten aus der DB benötigen.

Payroll:

Speicherung von Zahlungsdaten der Vergangenheit und aktuell:

Entitys:

PaymentHistory dient zur Speicherung des PaymentStatus mit den entprechenden Zeitdaten und betroffenen User

PaymentStatus wir in der PaymentHistory gespeichert und gibt an wie die Abbuchung verlaufen is

Repository:

PaymentHistoryRepository erlaubt die Persistentierung und Abfragen.

Service:

PaymentHistoryService

createPaymentHistory() zum anlegene einer neuen PaymentHistory

updatePaymentStatus() zum Updaten einer PaymentHistory

getAllByYearAndMonth() zum zugirff auf die DB

Controller:

PaymentHistoryController

getPaymentHistoryYearMonth zum DB auslesen ober den Servise mit getAllByYearAndMonth()

Speicherung von Zeiträume in denen ein User Premium aboniert hatte/hat:

Entitys:

PremiumHistory dient der Speicherung des PremiumStatuses mit Zeitstampel und betroffenen User.

Repository:

PremiumHistoryRepository erlaubt die Persistentierung und Abfragen.

Service:

PreimumHistoryService

savePremiumHistory() zum anlegen einer neuen PremiumHistory

getPremiumChangedByName() zum zugriff auf die DB

UserUpdater

updateUser() zum speichern eines Users aus dem PremiumStatusListener

Controller:

PremiumStatusListener

propertyChange() legt per Observer eine neue PremiumHistory an, wenn sich ein PremiumStatus in eine Users ändert

getPremiumIntervalByName() zum DB auslesen über den Service mit getPremiumChangedByName()

getTimePremiumInterval() nimmt eine Liste an PremiumHistories und berechnet eine Liste aus Zeitspannen in den der User premium war

getTotalPremiumTimeByName() nimmt einen User und berechnet mit hilfe von getTimePremiumInterval() die gesamte Zeit in der dieser User premium war (wird nur zum anzeigen Verwendet)

filterDatesByMonthAndYear() nimmt einen User ein Jahr und ein Monat ung gibt eine Liste von PremiumHistories in diesem Zeitraum von diesem User zurück

chargedDaysFromStartToEndCurrentMonth() nimmt eine Liste an PremiumHistories und berechnet den zu bezahlenden Preis für den Premiumzeitraum aus dem Parameter bis zum Ende des aktuellen Monats

cashUpTillEndCurrentMonth() nimmt einen User und berechnet der Preis mit hilfe von chargedDaysFromStartToEndCurrentMonth() und filterDatesByMonthAndYear() sowie priceForChargedDays() für den aktuellen Monat stellt eine Rechnugn an die Credidkarte des Users.

Wenn die Zahlung fehlschlägt wird dem User der Premiumstatus entzogen und er bekommt eine Mail das die Zahlung fehlgeschlagen ist

Landing Page

Autovervollständigung bei der Suche:

AutocompleteBean:
getAutocompletion liefert bei jeder Eingabe mittels der GeocodingAPI fünf Vorschläge zurück.
Als Typ fungiert hier *GeocodingDTO,* dem User angezeigt wird allerdings nur Name und Land.
Wenn ein User einen Vorschlag per Klick auswählt, wird dieser mittels *onItemSelect* im Feld *selectedGeocodingDTO gespeichert.*

Wetterdaten Allgemein:

WeatherBean löst bei Betägigung des „Search“ Buttons den Aufruf der API 1 aus und verwaltet die erhaltenen Wetterdaten. Dabei wird das zuvor gewählte *selectedGeocodingDTO* verwendet. Falls dies nicht gesetzt erhählt der User eine Nachricht.

Diagramme:

ChartBean:
createStackedBarModel erstellt ein Balkendiagramm für die Tagesvorschau
createMixedModel erstellt ein gemischtes Balken- /Liniendiagramm für die nächsten 24 Stunden

Stündliches Wetter im Detail:

HourlyWeatherBean:
Holt sich primät aus der *WeatherBean* die relevanten stündlichen Daten.
getTitle und *getHourlyWeather* liefern die Daten an das Frontend, abhängig davon ob der User den Premium Status hat oder nicht (48h Vorschau vs. 24h Vorschau).

Tägliches Wetter Vorschau:

ForecastBean
Holt sich primät aus der *WeatherBean* die relevanten Daten der nächsten Tagen.
getTitle und *getHourlyWeather* liefern die Daten an das Frontend, abhängig davon ob der User den Premium Status hat oder nicht (7 Tage Vorschau vs. 3 Tage Vorschau).

E-Mail Service:

Setup:

application.properties und *MailConfig* setzen die E-Mail Konfigurationen
Dabei wird auf das Paket *spring-boot-starter-email* zurückgegriffen

Implementierung:

EmailService verwendet *JavaMailSender* um einfache E-Mails über das Gmail-Konto des Projektteams zu versenden (noreply.weatherapp.uibk@gmail.com).
In dieser Klasse wird das Strategy Design Pattern verwendet, um das erstellen der unterschiedlichen E-Mails zu vereinfachen. Durch *setEmailStrategy* muss eine Strategie gesetzt werden. Aktuell gibt es zwei Möglichkeiten, nämlich *ConfirmationMailStrategy* und *PasswordChangeMailStrategy*. Danach kann man mittels *sendMail* den Empfänger und den Token übergeben und die E-Mail versenden.

Token:

Allgemein:

Die Entität *Token* wird für diverse Verifikationszwecke genützt und speichert im wesentlichen einen zufälligen Wert *tokenValue* und einen assozierten User *user*.

Repository:

TokenRepository erlaubt die Persistierung und Abfragen.

Service Klasse:

TokenService:
createVerificationToken erstellt und persistiert eine neue *Token* Entität
getUserByConfirmationToken liefert den assozierten User eines Tokens

Passwort Vergessen Funktion:

Start des Prozesses:

ResetPasswordController:
requestPasswordChange nimmt eine E-Mail Adresse entgegen und verifiziert diese.
Falls diese Adresse keinem User zugeordnet ist, wird eine *FacesMessage* ausgegeben.
Andernfalls, wird ein Token erstellt und ein Link dem User zugesendet

Überprüfung Token / Passwort Änderung:

ResetPasswordBean:
Bei Initialisierung wird der Token des Links überprüft und ein Boolean *validToken* gesetzt.
Dieser Wahrheitswert bestimmt ob das Formular zum Passwort ändern nutzbar ist.
resetPasswort führt die Änderung durch (delegiert an Service Klassen)

Favourite Location

FavoriteLocationController:

Es wird im *FavoriteLocationController* überprüft, ob der aktuelle User diese Location bereits gespeichert hat. Wenn nicht so wird der Location Name an den *FavoriteLocationService* zum Speichern übergeben.

Service:

Diese Klasse speichert die FavLocation (inklusiver der Höhen- und Breitengrade, welche von der GeocodingApi abgerufen werden und dem dazugehörigen User).
Außerdem wird hier der Index der Locations erneuert und gespeichert. Dies führt dazu das die Reihenfolge der Liste trotz Ausloggen bestehen bleibt.
Zusätzlich können Locations aus dem Repository gelöscht und geladen werden.

Repository:

Diese Klasse ermöglicht die Persistierung und Abfrage von Daten.

FavLocationConverter:

Wandelt eine FavLocation zu einem String um damit dieser in der Präsentation richtig angezeigt wird.

ReorderListBean

Diese Bean ermöglicht ordnen von Objekten in einer Liste. Die Favoriten Liste wird dabei persistent geordnet, da ein Index mit abgespeichert wird.
Die User Listen werden nur temporär sortiert.

FilterListBean

Diese Bean ermöglicht das temporäre Filtern einer Liste nach einem angegebenen Wert. Welche Attribute genau gefiltert werden kann man in dieser Klasse festlegen.

Vacation Planing

Wetterdaten Allgemein

DateBean: löst bei Betätigung des „Submit“ Buttons den Aufruf API2 aus und verwaltet die erhaltenen Wetterdaten. Dabei wird das zuvor gewählte selectedGeocodingDTO verwendet. Bei Betätigung des Buttons werden einmal die Average Weather kalkuliert, dafür haben wir einen Service(WeatherService) und für die von User gewählten Zeitraum einen Forecast gezeigt

Average berechnung:

Dafür haben wir da 2 Service Klassen. Nachdem der User einen Start und Enddate wählt, ruft der DateBean DateRangeService auf.

DateRangeService: Diese Klasse berechnet lediglich den Durchschnittswert (Mittelwert) zwischen dem Start- und Enddatum, das vom Benutzer ausgewählt wurde. Der Durchschnitt wird dann an den WeatherService weitergegeben.

WeatherService:In dieser Klasse wird der Durchschnittswert verwendet, um die API-Aufrufe durchzuführen und Wetterdaten für die letzten 5 Jahre für das ausgewählte Datum und den ausgewählten Ort abzurufen. Die Wetterdaten werden dann verwendet, um den Durchschnitt der vergangenen 5 Jahre zu berechnen und auszugeben.

Diagramme:

DailyWeatherChartBean:

verwendet die PrimeFaces-Bibliothek, um die Diagramme zu erstellen, und enthält Konfigurationsoptionen für das Aussehen des Balkendiagramms, einschließlich der Skalierung der Y-Achse und der Festlegung von Labels für die verschiedenen Tageszeiten

→ createDailyTemperatureChart: erstellt ein Balkendiagramm (BarChartModel) für die täglichen Temperaturen zu verschiedenen Tageszeiten (Morgen, Nachmittag, Abend, Nacht) anhand der Daten aus einer DailyAggregationDTO