Peak Conditions Mountain Weather Application

C1. Design Document

Design diagram 3-8

UML class diagram 9-12

ERD diagram 13

C2. Link to hosted web app 13

C4. Developers guide to setting up and running the application 15

Introduction 15

Requirements 15

Setting up the development environment 15

Accessing the database 16

C5 . Users guide: running the application from a users perspective 17

The landing page 17

Logging in to your account 20

Searching for mountains or mountain ranges 22

Search results page 23

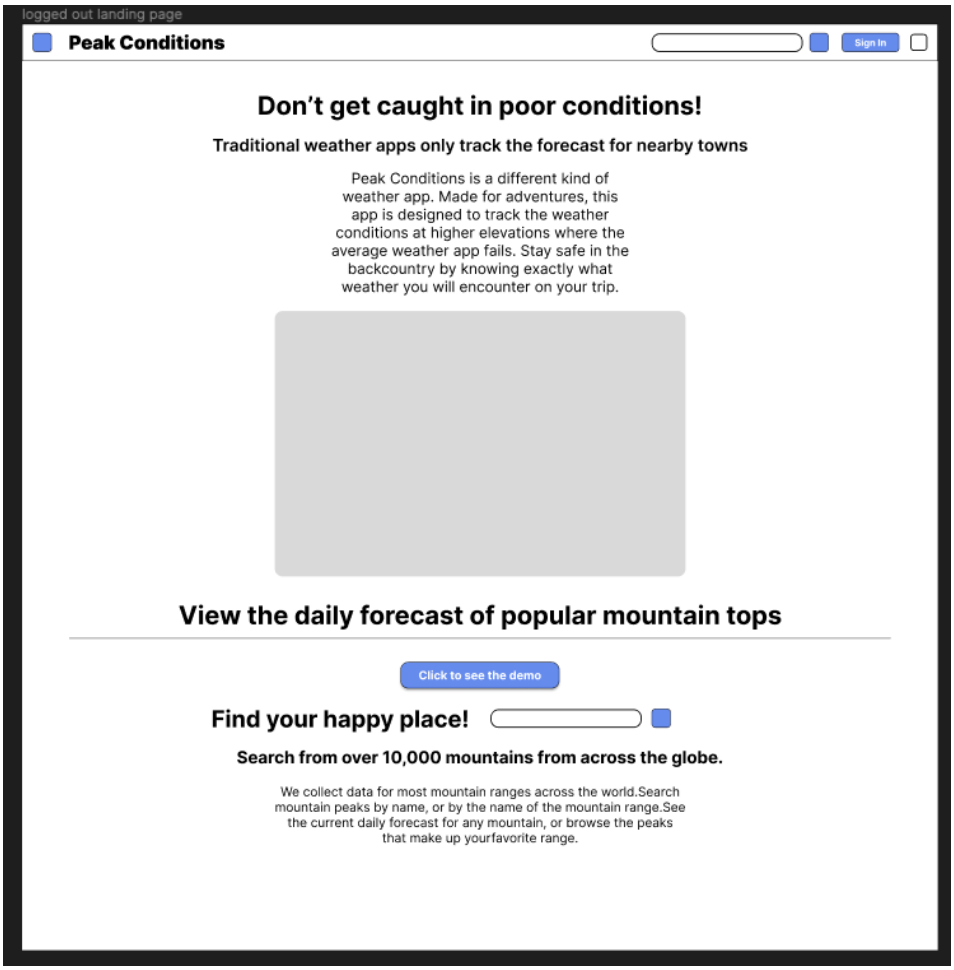
Displaying forecasts of searched mountains 24

- Saving mountains to the favorites list 24
- Getting forecasts of mountains from favorites list 25
- Changing users weather preferences profile 26
- Getting forecasts of mountains matching weather preferences 28
- Logging out of the application 29

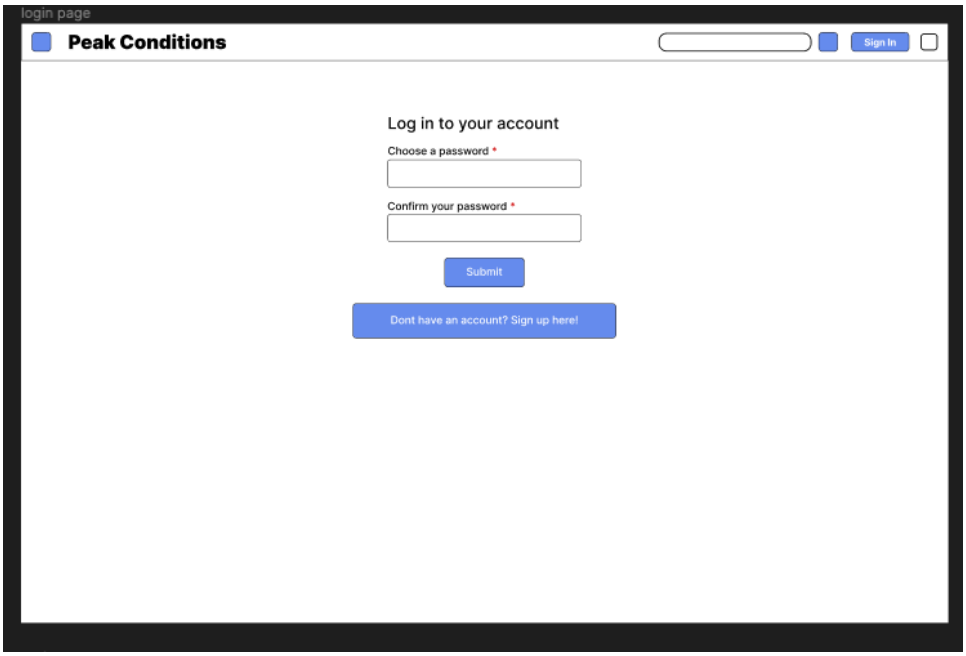
C1 - Design Document.

Design Diagram

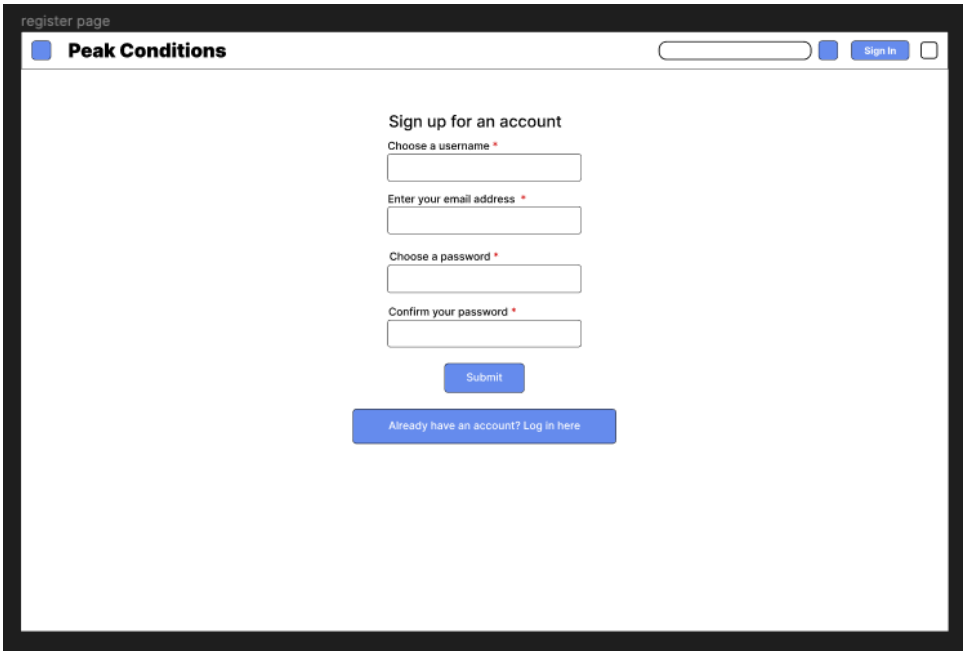
The unauthenticated landing page is the entry point to the application.



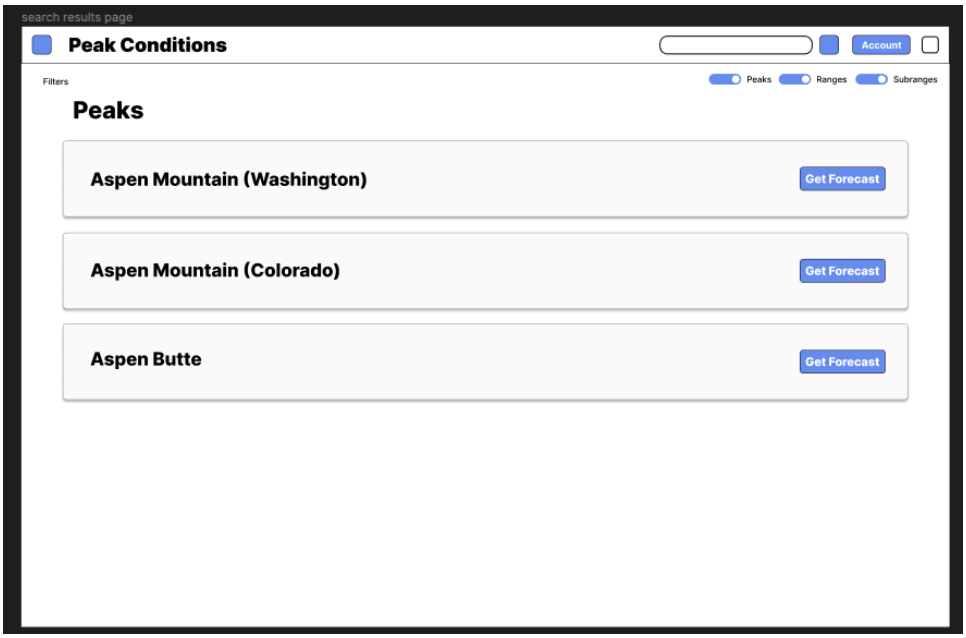
The user can click the sign in button to get to the login page.



clicking the ’dont have an account? sign up here!’ button will bring the user to the registration page.

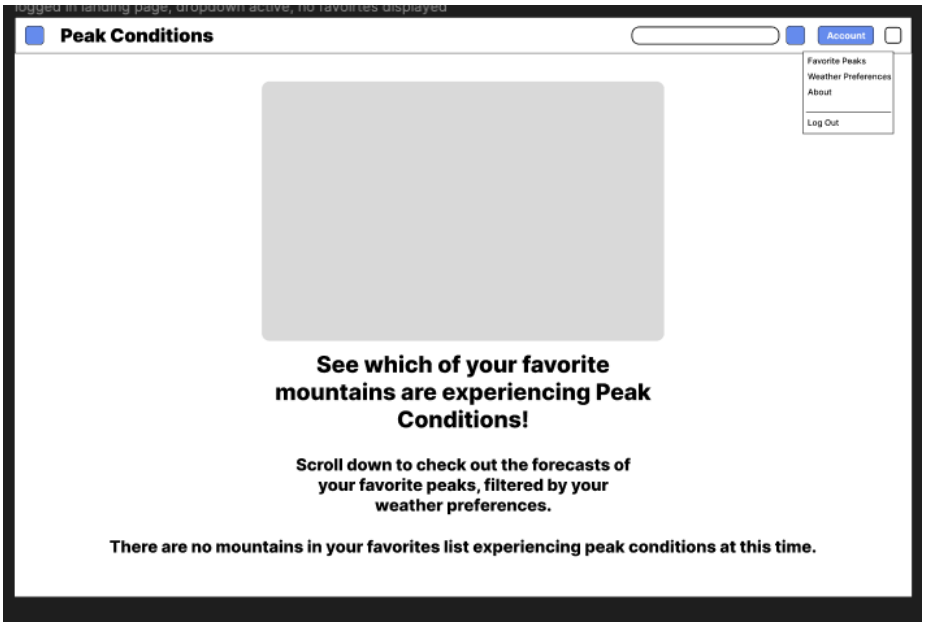


Back on the home page, clicking the ’see the demo’ button will display the days forecast for several random mountains. If the user wants to see the days forecast of a specific mountain they can search for it by name with the search bar on the top of the page in the navbar or the search bar in the middle of the page. The user is redirected to the search results page once the search has been entered.



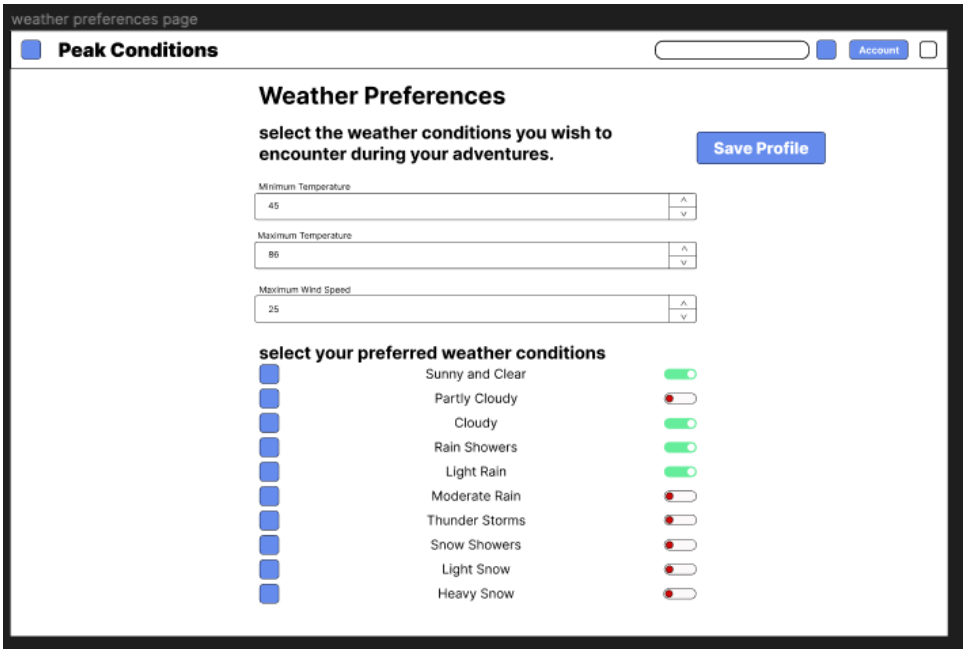
When a user registers for an account or logs in to an existing account they will be redirected to the authenticated home page.

The homepage shows the users favorites mountains filtered by their weather preferences. If the user has no favorites selected, or none of the favorites are expecting conditions defined by the weather preferences, there is a message explaining that no mountains are experiencing peak conditions.



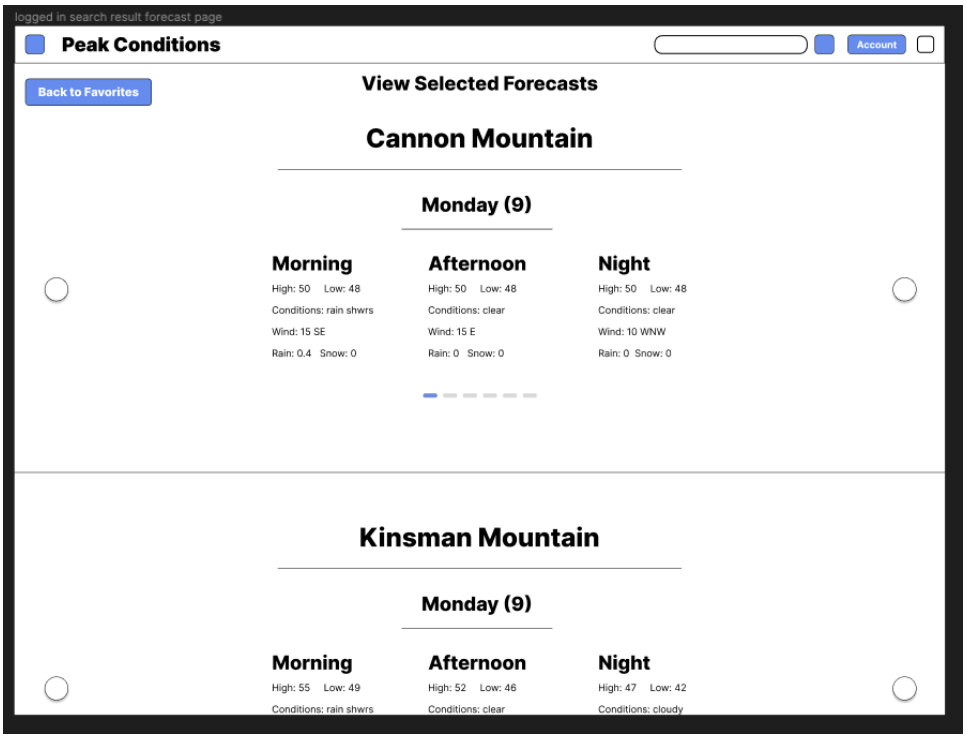
To set the users favorite weather preferences, they click the weather preferences option from the drop

down menu on the navbar.

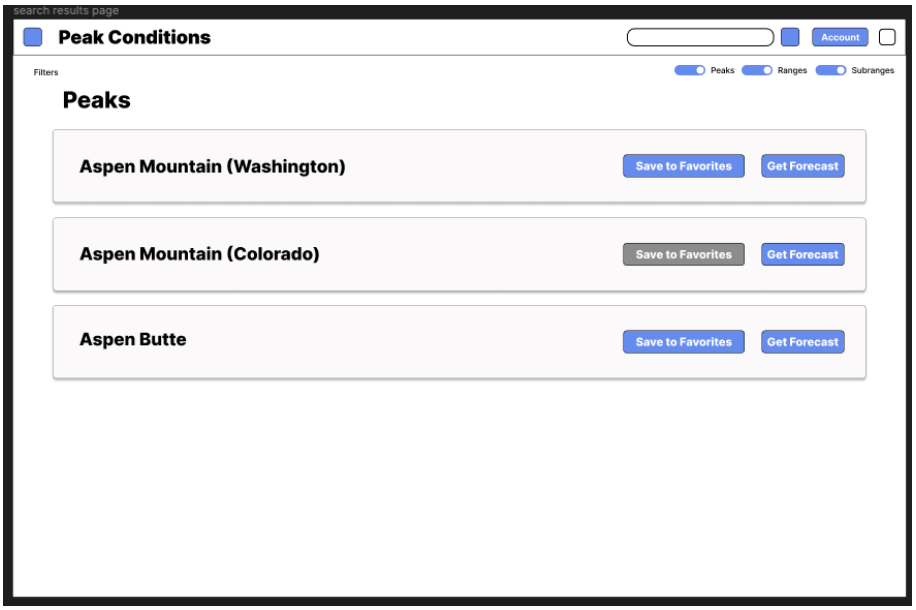


Clicking the favorites option in the dropdown menu will bring the user to their favorite mountain peaks list.

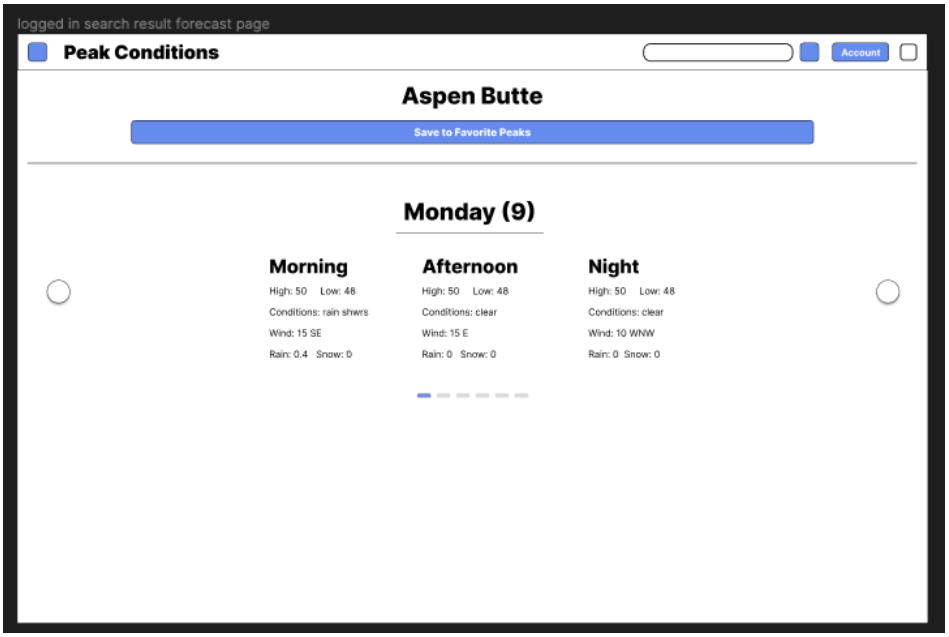
If the user clicks the 'remove' button next to an entry in the list, that entry will be removed. The user can also select one or more mountains to see the forecasts for at once. Clicking the see all selected forecasts button will redirect the user to the results page.



If a logged in user wishes to add mountains to their favorites list they can search for them using the search bar on the navbar or with the search bar on the home page. The search results page for logged in users has the extra option to save mountain peaks to the favorites list. When the peaks are saved the button will be disabled.

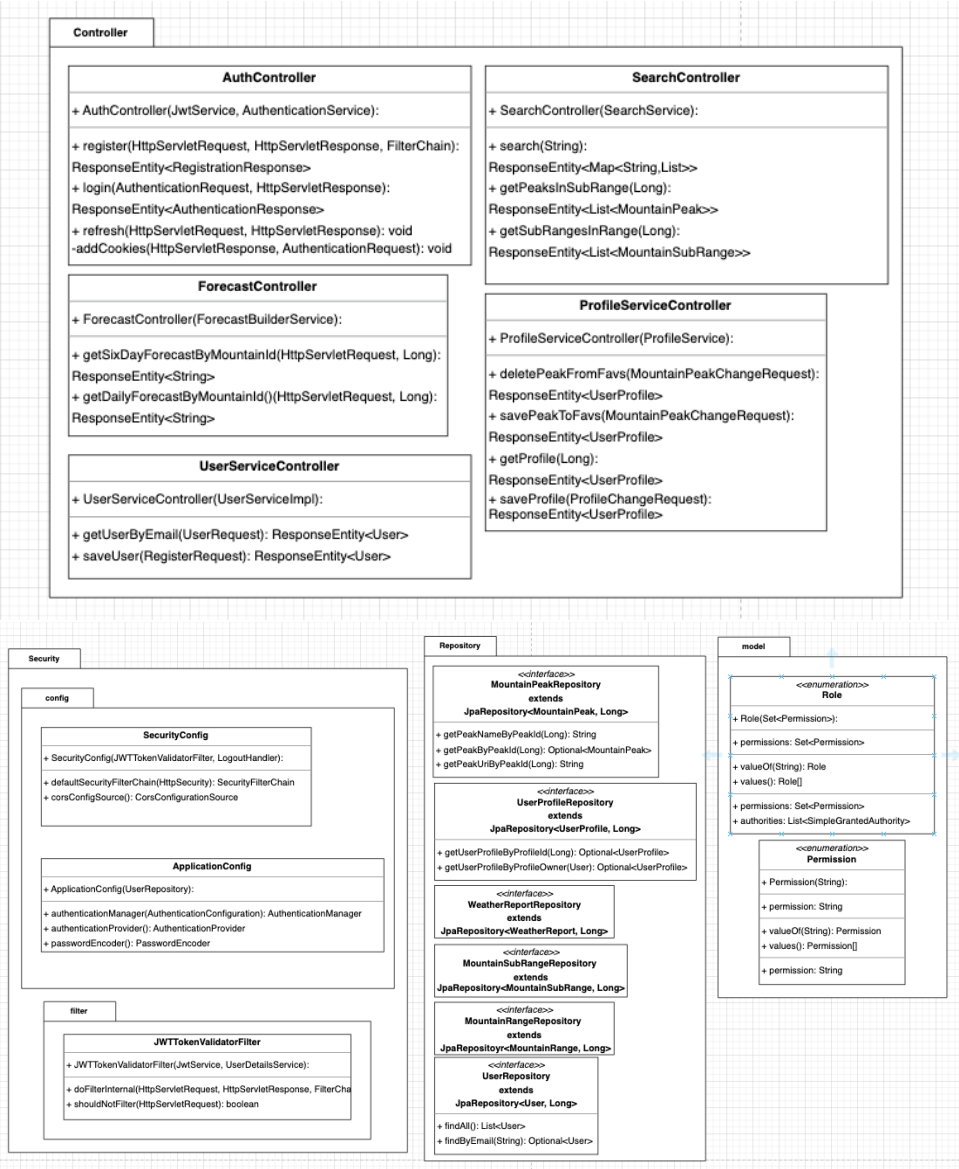


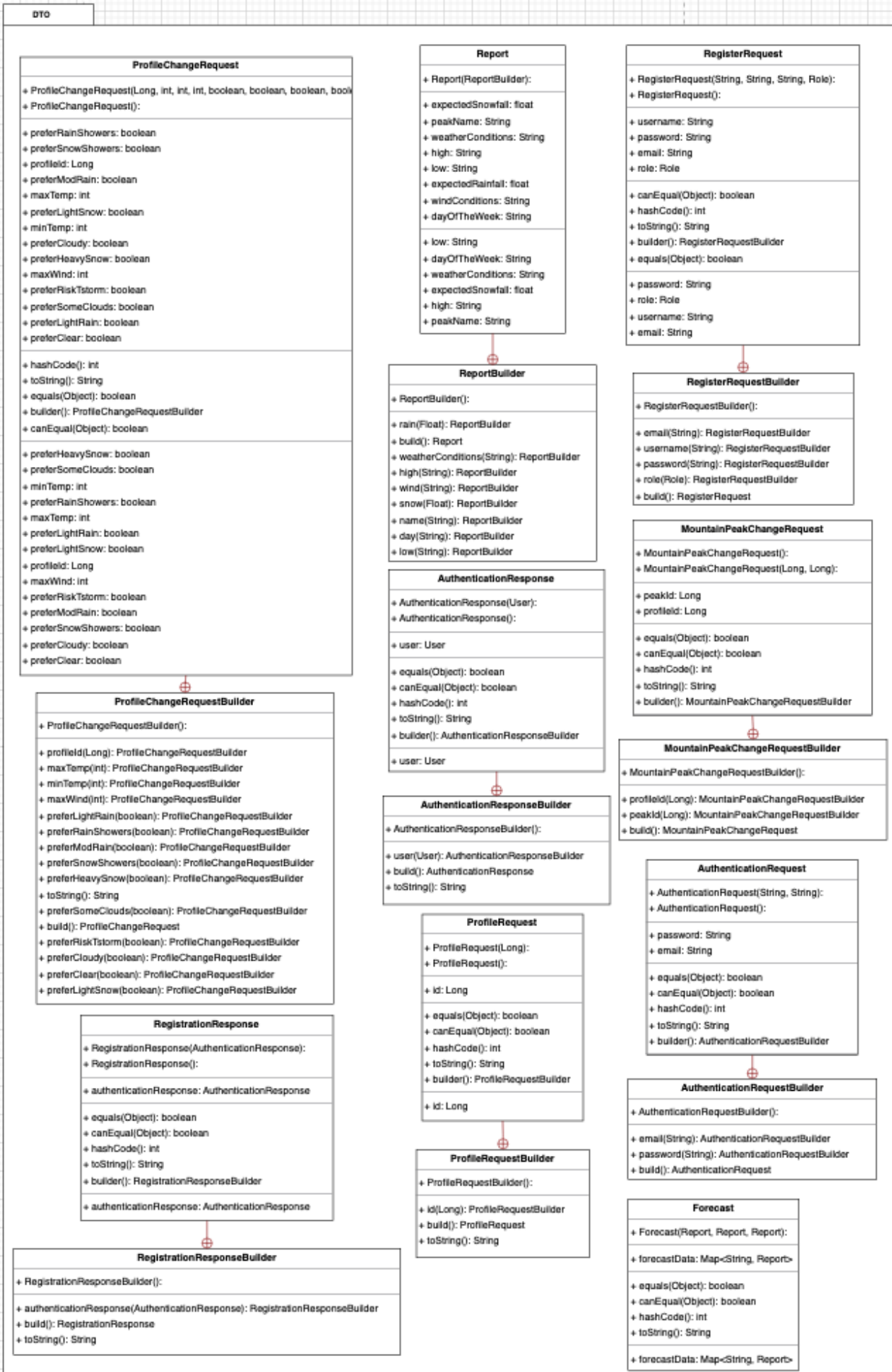
If the user clicks the get forecast button on one of the search results, they will be directed to a display of the forecast, with a button giving the option to save the peak to the forecast.

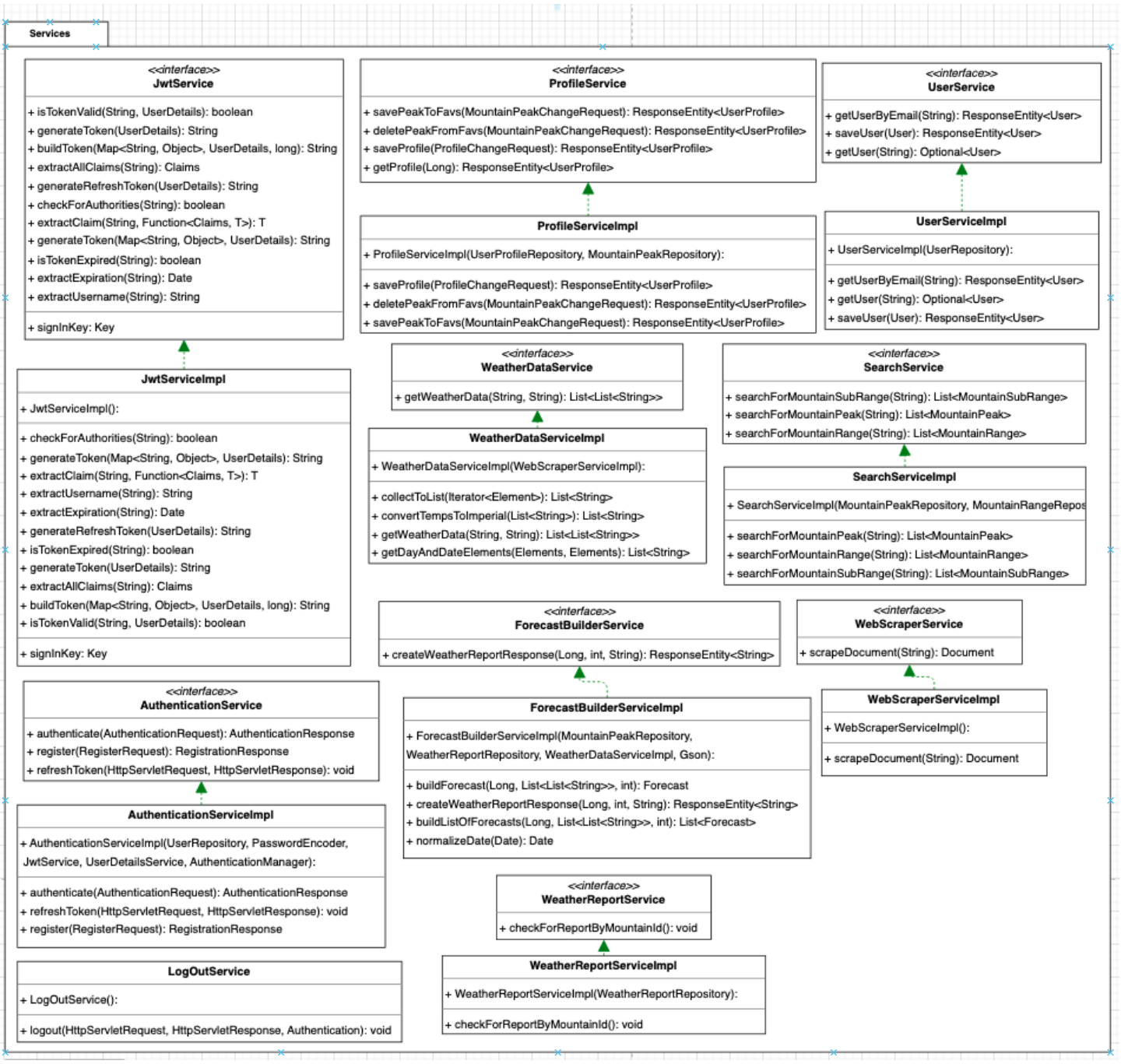


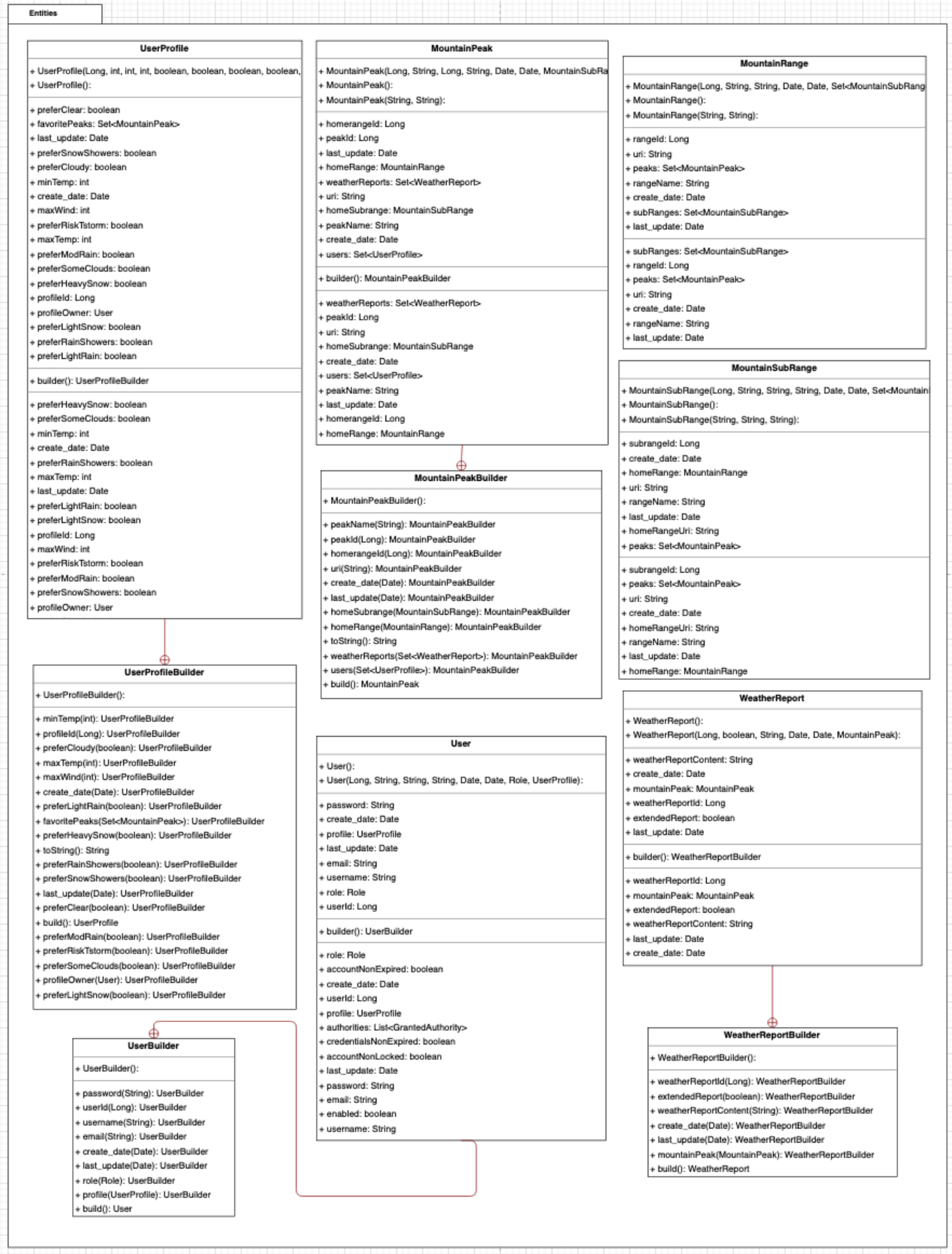
UML Class Diagram:

The following UML class diagrams represent the back-end server code. The back-end code base is a spring-boot application written in java. The server controls interactions between the application client and database, handles business logic related to creation and validation of users and their profiles, as well as the weather data service layer which gathers and processes information into weather reports. It also handles application security by issuing and validating JWT access tokens, and protecting certain API routes from being accessed by unauthenticated or unauthorized users.



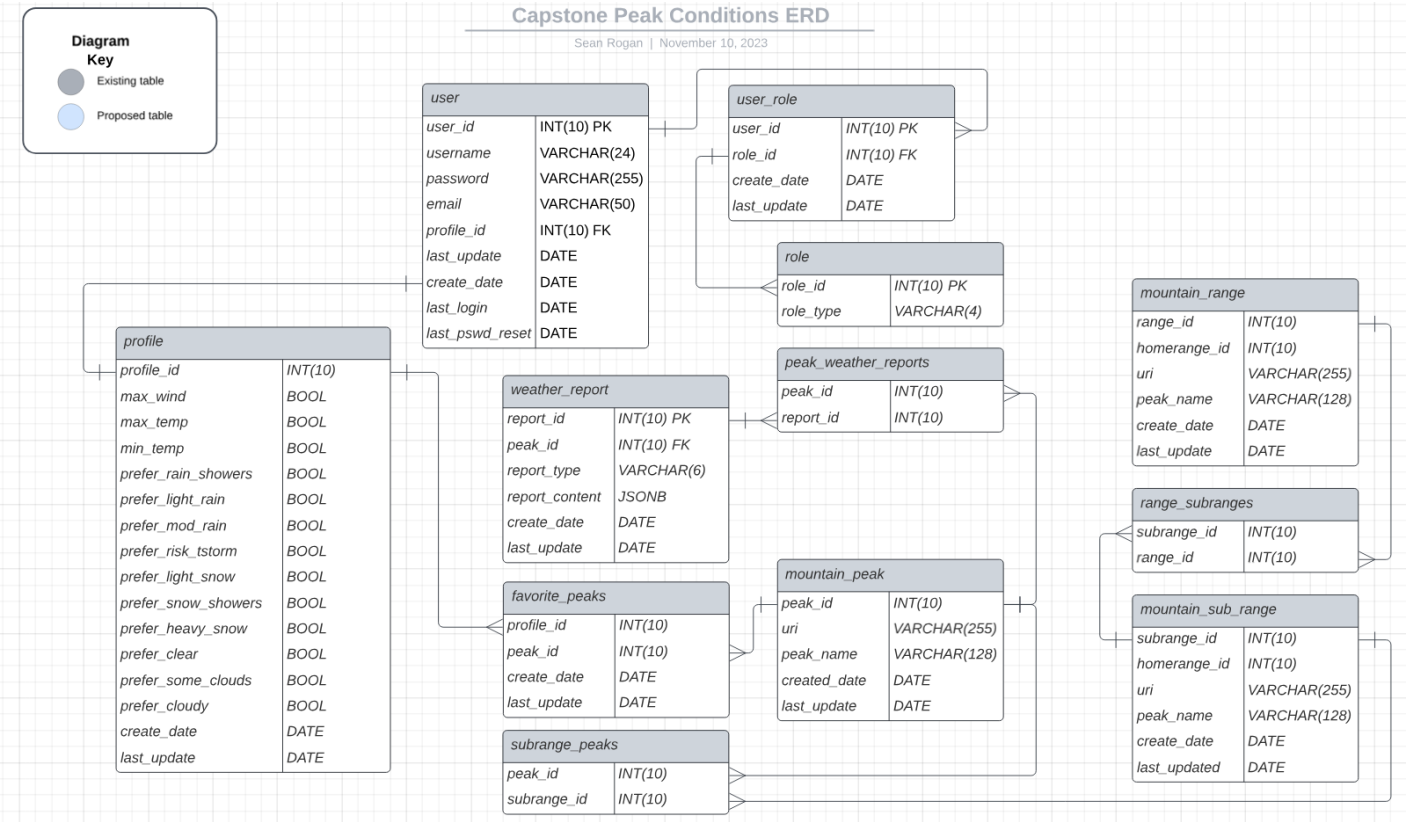

DTO
ProfileChangeRequest
+ ProfileChangeRequest(Long, int, int, int, boolean, boolean, boolean, bool
+ ProfileChangeRequest():
+ preferRainShowers: boolean
+ preferSnowShowers: boolean
+ profileId: Long
+ preferModRain: boolean
+ maxTemp: int
+ preferLightSnow: boolean
+ minTemp: int
+ preferCloudy: boolean
+ preferHeavySnow: boolean
+ maxWind: int
+ preferRiskTstorm: boolean
+ preferSomeClouds: boolean
+ preferLightRain: boolean
+ preferClear: boolean
+ hashCode(): int
+ toString(): String
+ equals(Object): boolean
+ builder(): ProfileChangeRequestBuilder
+ canEqual(Object): boolean
+ preferHeavySnow: boolean
+ preferSomeClouds: boolean
+ minTemp: int
+ preferRainShowers: boolean
+ maxTemp: int
+ preferLightRain: boolean
+ preferLightSnow: boolean
+ profileId: Long
+ maxWind: int
+ preferRiskTstorm: boolean
+ preferModRain: boolean
+ preferSnowShowers: boolean
+ preferCloudy: boolean
+ preferClear: boolean
ProfileChangeRequestBuilder
+ ProfileChangeRequestBuilder():
+ profileId(Long): ProfileChangeRequestBuilder
+ maxTemp(int): ProfileChangeRequestBuilder
+ minTemp(int): ProfileChangeRequestBuilder
+ maxWind(int): ProfileChangeRequestBuilder
+ preferLightRain(boolean): ProfileChangeRequestBuilder
+ preferRainShowers(boolean): ProfileChangeRequestBuilder
+ preferModRain(boolean): ProfileChangeRequestBuilder
+ preferSnowShowers(boolean): ProfileChangeRequestBuilder
+ preferHeavySnow(boolean): ProfileChangeRequestBuilder
+ toString(): String
+ preferSomeClouds(boolean): ProfileChangeRequestBuilder
+ build(): ProfileChangeRequest
+ preferRiskTstorm(boolean): ProfileChangeRequestBuilder
+ preferCloudy(boolean): ProfileChangeRequestBuilder
+ preferClear(boolean): ProfileChangeRequestBuilder
+ preferLightSnow(boolean): ProfileChangeRequestBuilder
RegistrationResponse
+ RegistrationResponse(AuthenticationResponse):
+ RegistrationResponse():
+ authenticationResponse: AuthenticationResponse
+ equals(Object): boolean
+ canEqual(Object): boolean
+ hashCode(): int
+ toString(): String
+ builder(): RegistrationResponseBuilder
+ authenticationResponse: AuthenticationResponse
RegistrationResponseBuilder
+ RegistrationResponseBuilder():
+ authenticationResponse(AuthenticationResponse): RegistrationResponseBuilder
+ build(): RegistrationResponse
+ toString(): String
Report
+ Report(ReportBuilder):
+ expectedSnowfall: float
+ peakName: String
+ weatherConditions: String
+ high: String
+ low: String
+ expectedRainfall: float
+ windConditions: String
+ dayOfTheWeek: String
+ low: String
+ dayOfTheWeek: String
+ weatherConditions: String
+ expectedSnowfall: float
+ high: String
+ peakName: String
ReportBuilder
+ ReportBuilder():
+ rain(Float): ReportBuilder
+ build(): Report
+ weatherConditions(String): ReportBuilder
+ high(String): ReportBuilder
+ wind(String): ReportBuilder
+ snow(Float): ReportBuilder
+ name(String): ReportBuilder
+ day(String): ReportBuilder
+ low(String): ReportBuilder
AuthenticationResponse
+ AuthenticationResponse(User):
+ AuthenticationResponse():
+ user: User
+ equals(Object): boolean
+ canEqual(Object): boolean
+ hashCode(): int
+ toString(): String
+ builder(): AuthenticationResponseBuilder
+ user: User
AuthenticationResponseBuilder
+ AuthenticationResponseBuilder():
+ user(User): AuthenticationResponseBuilder
+ build(): AuthenticationResponse
+ toString(): String
ProfileRequest
+ ProfileRequest(Long):
+ ProfileRequest():
+ id: Long
+ equals(Object): boolean
+ canEqual(Object): boolean
+ hashCode(): int
+ toString(): String
+ builder(): ProfileRequestBuilder
+ id: Long
ProfileRequestBuilder
+ ProfileRequestBuilder():
+ id(Long): ProfileRequestBuilder
+ build(): ProfileRequest
+ toString(): String
RegisterRequest
+ RegisterRequest(String, String, String, Role):
+ RegisterRequest():
+ username: String
+ password: String
+ email: String
+ role: Role
+ canEqual(Object): boolean
+ hashCode(): int
+ toString(): String
+ builder(): RegisterRequestBuilder
+ equals(Object): boolean
+ password: String
+ role: Role
+ username: String
+ email: String
RegisterRequestBuilder
+ RegisterRequestBuilder():
+ email(String): RegisterRequestBuilder
+ username(String): RegisterRequestBuilder
+ password(String): RegisterRequestBuilder
+ role(Role): RegisterRequestBuilder
+ build(): RegisterRequest
MountainPeakChangeRequest
+ MountainPeakChangeRequest():
+ MountainPeakChangeRequest(Long, Long):
+ peakId: Long
+ profileId: Long
+ equals(Object): boolean
+ canEqual(Object): boolean
+ hashCode(): int
+ toString(): String
+ builder(): MountainPeakChangeRequestBuilder
MountainPeakChangeRequestBuilder
+ MountainPeakChangeRequestBuilder():
+ profileId(Long): MountainPeakChangeRequestBuilder
+ peakId(Long): MountainPeakChangeRequestBuilder
+ build(): MountainPeakChangeRequest
AuthenticationRequest
+ AuthenticationRequest(String, String):
+ AuthenticationRequest():
+ password: String
+ email: String
+ equals(Object): boolean
+ canEqual(Object): boolean
+ hashCode(): int
+ toString(): String
+ builder(): AuthenticationRequestBuilder
AuthenticationRequestBuilder
+ AuthenticationRequestBuilder():
+ email(String): AuthenticationRequestBuilder
+ password(String): AuthenticationRequestBuilder
+ build(): AuthenticationRequest
Forecast
+ Forecast(Report, Report, Report):
+ forecastData: Map<String, Report>
+ equals(Object): boolean
+ canEqual(Object): boolean
+ hashCode(): int
+ toString(): String
+ forecastData: Map<String, Report>


Services
<<interface>>
JwtService
+ isTokenValid(String, UserDetails): boolean
+ generateToken(UserDetails): String
+ buildToken(Map<String, Object>, UserDetails, long): String
+ extractAllClaims(String): Claims
+ generateRefreshToken(UserDetails): String
+ checkForAuthorities(String): boolean
+ extractClaim(String, Function<Claims, T>): T
+ generateToken(Map<String, Object>, UserDetails): String
+ isTokenExpired(String): boolean
+ extractExpiration(String): Date
+ extractUsername(String): String
+ signInKey: Key
JwtServiceImpl
+ JwtServiceImpl():
+ checkForAuthorities(String): boolean
+ generateToken(Map<String, Object>, UserDetails): String
+ extractClaim(String, Function<Claims, T>): T
+ extractUsername(String): String
+ extractExpiration(String): Date
+ generateRefreshToken(UserDetails): String
+ isTokenExpired(String): boolean
+ generateToken(UserDetails): String
+ extractAllClaims(String): Claims
+ buildToken(Map<String, Object>, UserDetails, long): String
+ isTokenValid(String, UserDetails): boolean
+ signInKey: Key
<<interface>>
ProfileService
+ savePeakToFavs(MountainPeakChangeRequest): ResponseEntity<UserProfile>
+ deletePeakFromFavs(MountainPeakChangeRequest): ResponseEntity<UserProfile>
+ saveProfile(ProfileChangeRequest): ResponseEntity<UserProfile>
+ getProfile(Long): ResponseEntity<UserProfile>
ProfileServiceImpl
+ ProfileServiceImpl(UserProfileRepository, MountainPeakRepository):
+ saveProfile(ProfileChangeRequest): ResponseEntity<UserProfile>
+ deletePeakFromFavs(MountainPeakChangeRequest): ResponseEntity<UserProfile>
+ savePeakToFavs(MountainPeakChangeRequest): ResponseEntity<UserProfile>
<<interface>>
UserService
+ getUserByEmail(String): ResponseEntity<User>
+ saveUser(User): ResponseEntity<User>
+ getUser(String): Optional<User>
UserServiceImpl
+ UserServiceImpl(UserRepository):
+ getUserByEmail(String): ResponseEntity<User>
+ getUser(String): Optional<User>
+ saveUser(User): ResponseEntity<User>
<<interface>>
WeatherDataService
+ getWeatherData(String, String): List<List<String>>
WeatherDataServiceImpl
+ WeatherDataServiceImpl(WebScraperServiceImpl):
+ collectToList(Iterator<Element>): List<String>
+ convertTempsToImperial(List<String>): List<String>
+ getWeatherData(String, String): List<List<String>>
+ getDayAndDateElements(Elements, Elements): List<String>
<<interface>>
SearchService
+ searchForMountainSubRange(String): List<MountainSubRange>
+ searchForMountainPeak(String): List<MountainPeak>
+ searchForMountainRange(String): List<MountainRange>
SearchServiceImpl
+ SearchServiceImpl(MountainPeakRepository, MountainRangeRepos
+ searchForMountainPeak(String): List<MountainPeak>
+ searchForMountainRange(String): List<MountainRange>
+ searchForMountainSubRange(String): List<MountainSubRange>
<<interface>>
ForecastBuilderService
+ createWeatherReportResponse(Long, int, String): ResponseEntity<String>
ForecastBuilderServiceImpl
+ ForecastBuilderServiceImpl(MountainPeakRepository,
WeatherReportRepository, WeatherDataServiceImpl, Gson):
+ buildForecast(Long, List<List<String>>, int): Forecast
+ createWeatherReportResponse(Long, int, String): ResponseEntity<String>
+ buildListOfForecasts(Long, List<List<String>>, int): List<Forecast>
+ normalizeDate(Date): Date
<<interface>>
WebScraperService
+ scrapeDocument(String): Document
WebScraperServiceImpl
+ WebScraperServiceImpl():
+ scrapeDocument(String): Document
<<interface>>
AuthenticationService
+ authenticate(AuthenticationRequest): AuthenticationResponse
+ register(RegisterRequest): RegistrationResponse
+ refreshToken(HttpServletRequest, HttpServletResponse): void
AuthenticationServiceImpl
+ AuthenticationServiceImpl(UserRepository, PasswordEncoder,
JwtService, UserDetailsService, AuthenticationManager):
+ authenticate(AuthenticationRequest): AuthenticationResponse
+ refreshToken(HttpServletRequest, HttpServletResponse): void
+ register(RegisterRequest): RegistrationResponse
<<interface>>
WeatherReportService
+ checkForReportByMountainId(): void
WeatherReportServiceImpl
+ WeatherReportServiceImpl(WeatherReportRepository):
+ checkForReportByMountainId(): void
LogOutService
+ LogOutService():
+ logout(HttpServletRequest, HttpServletResponse, Authentication): void


Entities
UserProfile
+ UserProfile(Long, int, int, int, boolean, boolean, boolean, boolean,
+ UserProfile():
+ preferClear: boolean
+ favoritePeaks: Set<MountainPeak>
+ last_update: Date
+ preferSnowShowers: boolean
+ preferCloudy: boolean
+ minTemp: int
+ create_date: Date
+ maxWind: int
+ preferRiskTstorm: boolean
+ maxTemp: int
+ preferModRain: boolean
+ preferSomeClouds: boolean
+ preferHeavySnow: boolean
+ profileId: Long
+ profileOwner: User
+ preferLightSnow: boolean
+ preferRainShowers: boolean
+ preferLightRain: boolean
+ builder(): UserProfileBuilder
+ preferHeavySnow: boolean
+ preferSomeClouds: boolean
+ minTemp: int
+ create_date: Date
+ preferRainShowers: boolean
+ maxTemp: int
+ last_update: Date
+ preferLightRain: boolean
+ preferLightSnow: boolean
+ profileId: Long
+ maxWind: int
+ preferRiskTstorm: boolean
+ preferModRain: boolean
+ preferSnowShowers: boolean
+ profileOwner: User
UserProfileBuilder
+ UserProfileBuilder():
+ minTemp(int): UserProfileBuilder
+ profileId(Long): UserProfileBuilder
+ preferCloudy(boolean): UserProfileBuilder
+ maxTemp(int): UserProfileBuilder
+ maxWind(int): UserProfileBuilder
+ create_date(Date): UserProfileBuilder
+ preferLightRain(boolean): UserProfileBuilder
+ favoritePeaks(Set<MountainPeak>): UserProfileBuilder
+ preferHeavySnow(boolean): UserProfileBuilder
+ toString(): String
+ preferRainShowers(boolean): UserProfileBuilder
+ preferSnowShowers(boolean): UserProfileBuilder
+ last_update(Date): UserProfileBuilder
+ preferClear(boolean): UserProfileBuilder
+ build(): UserProfile
+ preferModRain(boolean): UserProfileBuilder
+ preferRiskTstorm(boolean): UserProfileBuilder
+ preferSomeClouds(boolean): UserProfileBuilder
+ profileOwner(User): UserProfileBuilder
+ preferLightSnow(boolean): UserProfileBuilder
UserBuilder
+ UserBuilder():
+ password(String): UserBuilder
+ userId(Long): UserBuilder
+ username(String): UserBuilder
+ email(String): UserBuilder
+ create_date(Date): UserBuilder
+ last_update(Date): UserBuilder
+ role(Role): UserBuilder
+ profile(UserProfile): UserBuilder
+ build(): User
MountainPeak
+ MountainPeak(Long, String, Long, String, Date, Date, MountainSubRa
+ MountainPeak():
+ MountainPeak(String, String):
+ homerangeId: Long
+ peakId: Long
+ last_update: Date
+ homeRange: MountainRange
+ weatherReports: Set<WeatherReport>
+ uri: String
+ homeSubrange: MountainSubRange
+ peakName: String
+ create_date: Date
+ users: Set<UserProfile>
+ builder(): MountainPeakBuilder
+ weatherReports: Set<WeatherReport>
+ peakId: Long
+ uri: String
+ homeSubrange: MountainSubRange
+ create_date: Date
+ users: Set<UserProfile>
+ peakName: String
+ last_update: Date
+ homerangeId: Long
+ homeRange: MountainRange
MountainPeakBuilder
+ MountainPeakBuilder():
+ peakName(String): MountainPeakBuilder
+ peakId(Long): MountainPeakBuilder
+ homerangeId(Long): MountainPeakBuilder
+ uri(String): MountainPeakBuilder
+ create_date(Date): MountainPeakBuilder
+ last_update(Date): MountainPeakBuilder
+ homeSubrange(MountainSubRange): MountainPeakBuilder
+ homeRange(MountainRange): MountainPeakBuilder
+ toString(): String
+ weatherReports(Set<WeatherReport>): MountainPeakBuilder
+ users(Set<UserProfile>): MountainPeakBuilder
+ build(): MountainPeak
User
+ User():
+ User(Long, String, String, String, Date, Date, Role, UserProfile):
+ password: String
+ create_date: Date
+ profile: UserProfile
+ last_update: Date
+ email: String
+ username: String
+ role: Role
+ userId: Long
+ builder(): UserBuilder
+ role: Role
+ accountNonExpired: boolean
+ create_date: Date
+ userId: Long
+ profile: UserProfile
+ authorities: List<GrantedAuthority>
+ credentialsNonExpired: boolean
+ accountNonLocked: boolean
+ last_update: Date
+ password: String
+ email: String
+ enabled: boolean
+ username: String
MountainRange
+ MountainRange(Long, String, String, Date, Date, Set<MountainSubRang
+ MountainRange():
+ MountainRange(String, String):
+ rangeId: Long
+ uri: String
+ peaks: Set<MountainPeak>
+ rangeName: String
+ create_date: Date
+ subRanges: Set<MountainSubRange>
+ last_update: Date
+ subRanges: Set<MountainSubRange>
+ rangeId: Long
+ peaks: Set<MountainPeak>
+ uri: String
+ create_date: Date
+ rangeName: String
+ last_update: Date
MountainSubRange
+ MountainSubRange(Long, String, String, String, Date, Date, Set<Mountain
+ MountainSubRange():
+ MountainSubRange(String, String, String):
+ subrangeId: Long
+ create_date: Date
+ homeRange: MountainRange
+ uri: String
+ rangeName: String
+ last_update: Date
+ homeRangeUri: String
+ peaks: Set<MountainPeak>
+ subrangeId: Long
+ peaks: Set<MountainPeak>
+ uri: String
+ create_date: Date
+ homeRangeUri: String
+ rangeName: String
+ last_update: Date
+ homeRange: MountainRange
WeatherReport
+ WeatherReport():
+ WeatherReport(Long, boolean, String, Date, Date, MountainPeak):
+ weatherReportContent: String
+ create_date: Date
+ mountainPeak: MountainPeak
+ weatherReportId: Long
+ extendedReport: boolean
+ last_update: Date
+ builder(): WeatherReportBuilder
+ weatherReportId: Long
+ mountainPeak: MountainPeak
+ extendedReport: boolean
+ weatherReportContent: String
+ last_update: Date
+ create_date: Date
WeatherReportBuilder
+ WeatherReportBuilder():
+ weatherReportId(Long): WeatherReportBuilder
+ extendedReport(boolean): WeatherReportBuilder
+ weatherReportContent(String): WeatherReportBuilder
+ create_date(Date): WeatherReportBuilder
+ last_update(Date): WeatherReportBuilder
+ mountainPeak(MountainPeak): WeatherReportBuilder
+ build(): WeatherReport

ERD Diagram

The ERD (Entity Relationship Diagram) represents the relationships between different database entities.



C2 - Link to hosted web app :

https://peak-conditions.com

C3 - Developer's Guide for Setting Up and Running the Application

Introduction

This guide is intended for developers who will set up, maintain, and debug the 'Peak Conditions' application. The application back-end is a Spring Boot project with PostgreSQL as the database with some additional dependencies for web, security, and testing. The front-end is a NextJs/ReactJs client application. The application uses Docker to containerize all the services for deployment to the cloud.

Prerequisites – the following software must be installed to run the application.

Docker: The application uses docker to run the application in containers, thus avoiding the need for the developer to download and install the software required by the application to their local machine.

Integrated Development Environment (IDE): An IDE like IntelliJ IDEA, Eclipse, or Spring Tool Suite (STS) is recommended for ease of development and debugging. This guide will focus on using IntelliJ IDEA, as it is the IDE used by the dev team on the project and is the most modern JDK IDE which is widely used today.

Setting Up the Development Environment

Clone the Repository: clone the repository to your local machine. Alternatively, you can download the source code directly.

Build the Project with Docker:

Open your IDE and navigate to the docker-compose.yml file. If using intellij IDEA, there is a GUI option to run the file by right clicking on it to open the options menu, then selecting the 'run docker compose' option. Alternatively the developer can select the file in the project file explorer, and type the keyboard shortcut 'ctrl + shift + R' to start the docker-compose process.

The docker compose process can also be started from the terminal. The developer, once they have confirmed they are in the correct directory containing the docker-compose.yml file, can type the command ' docker compose up –d '. One consideration for the developer to take before launching the docker process is to ensure that the ports used by the application are not in use on their local machine or the build and deployment process will fail. The application uses the port 8080 for the backend, 5432 for the database, and 3000 for the client. Ensure all these ports are free before running the docker compose build to avoid complications.

Running the Application

Once the docker builds are completed, the application will automatically start within the containers. The first time the database container starts, the database will need to be populated with data by the backend server, which will take roughly 10 minutes. The developer should wait until this process is complete before attempting any testing. The process can be monitored for completion by viewing the container logs for the springboot-app service. This can be easily found in the services section of the IntelliJ IDE, under docker.

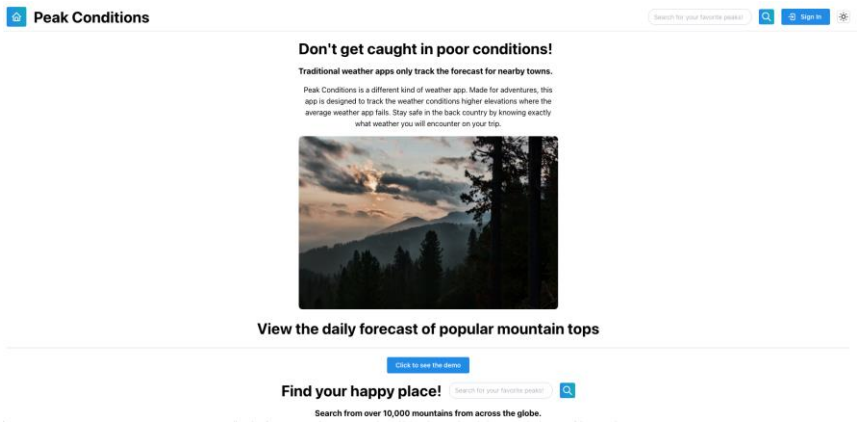
Accessing the database:

To access the database for testing or other purposes, the user must first access the container the database is running inside. To do this, the user must type into their terminal the command ' docker exec –it d424-software-engineering-capstone-db-1 bash '. If for some reason the database service name, or the name of the container has been changed, replace the section 'd424-software-engineering-capstone-db-1' with the correct name of the container. Once inside the container, the developer can use the standard psql commands to interact with the database. To log into the database type the command 'psql -U postgres –d postgres-peak-conditions' and the user will be signed in so they can enter PostgreSQL queries.

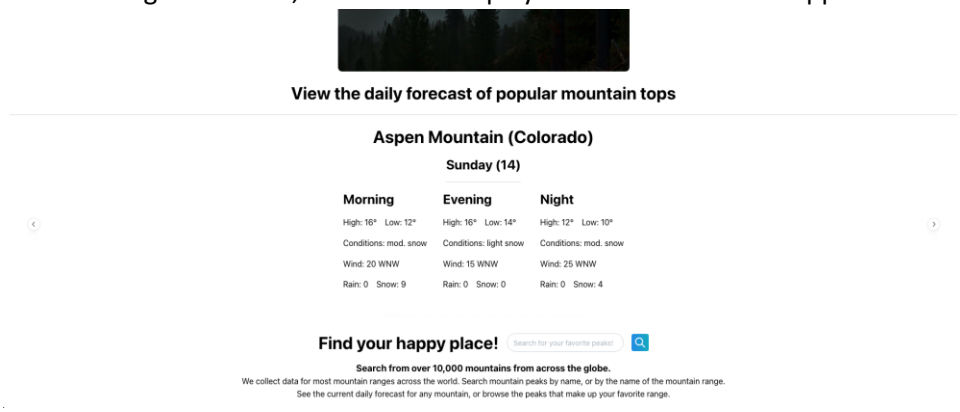
C4 - Users Guide: running the application from a user perspective.

The landing page:

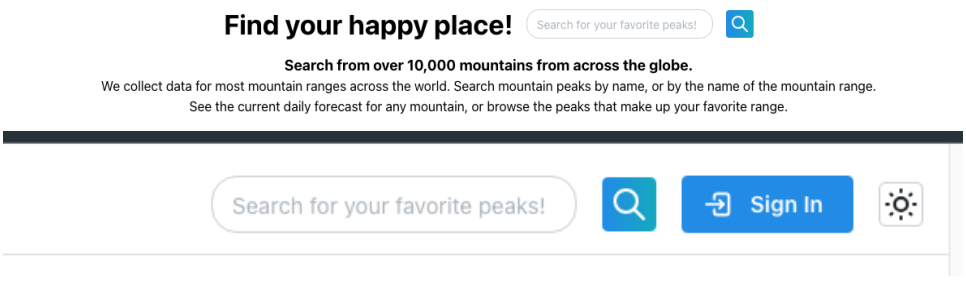
The ‘Peak Conditions’ weather app features a public landing page where unauthenticated users can choose to view a demo where the daily forecast of popular mountains are displayed. The demo is displayed by clicking the blue ‘click to see the demo’ button.



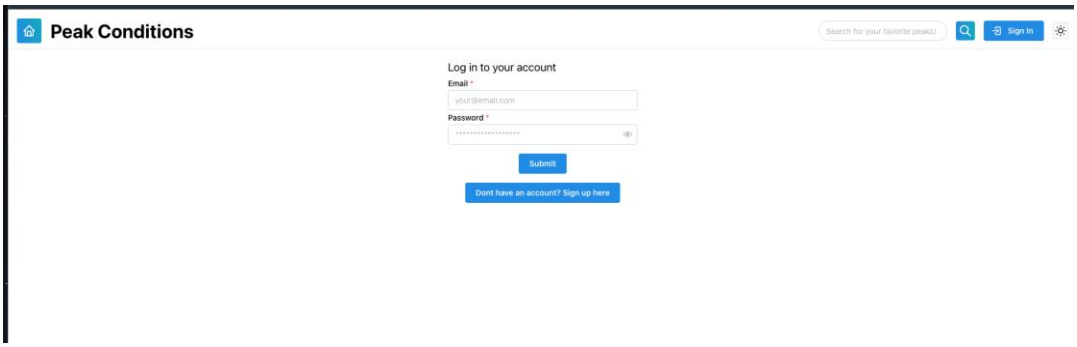
After clicking the button, the demo is displayed and the button disappears.



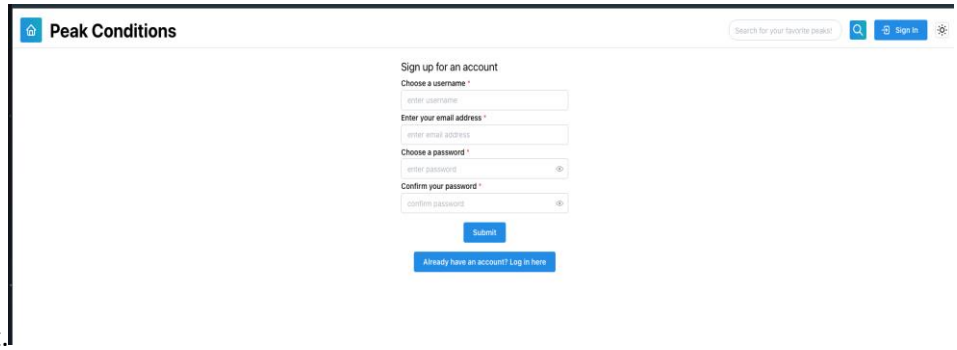
The landing page includes a search bar feature below the demo, which can be used to locate other mountains and see their forecasts. The nav bar also has a search bar with the same functionality.



Users visiting the public landing page will have the opportunity to sign-in to their account by clicking the ‘sign in’ button located on the navbar. Upon clicking the sign in button, the user will be redirected to the login page.



If the user does not yet have an account registered, they can click the 'register here' link, below the sign in button on the login page, to be redirected to the registration page where new users can sign up for

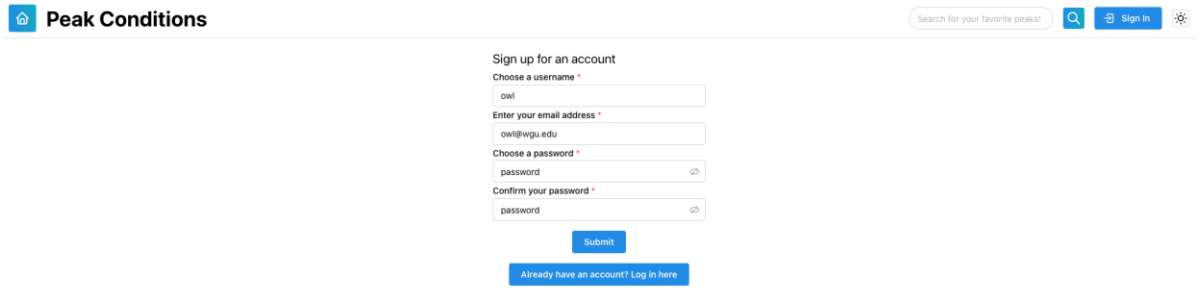


their account.

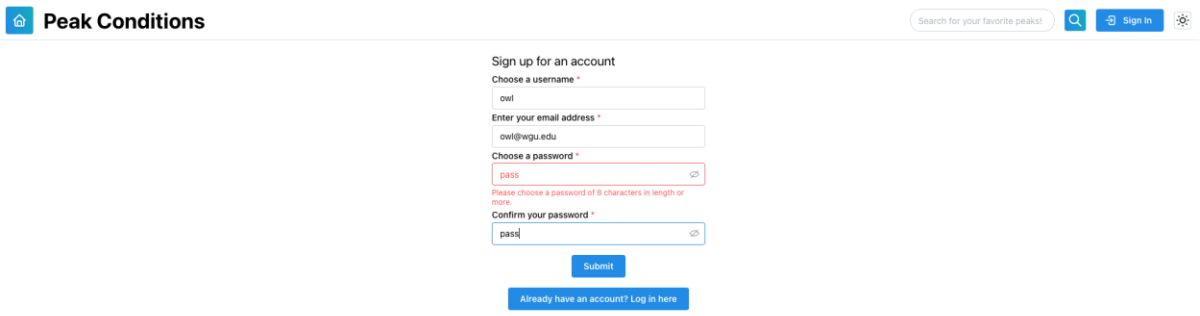
If at any point the user wishes to return to the landing page, they can do so by clicking the home icon button at the left side of the navbar.

Registering an account:

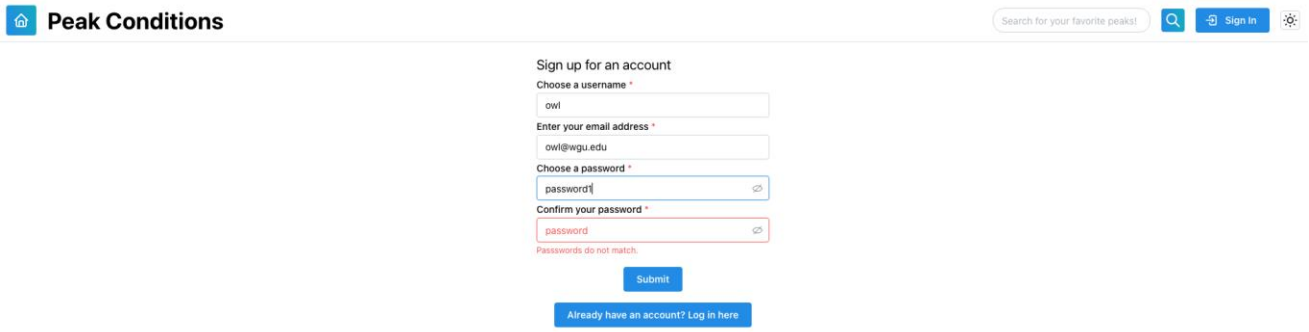
When a user first uses the application they will need to register an account first. Once on the registration page, the user is prompted to select a username, enter your email address, enter a password, and then re-enter the password to confirm your choice.



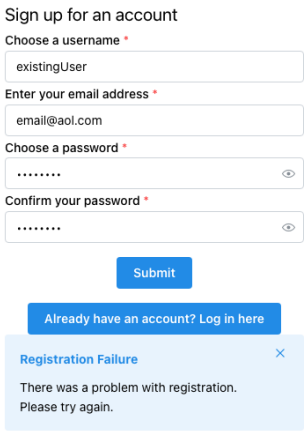
The password must be at least 8 characters long, and if it is fewer than 8 characters long the form will not be submitted and an alert will be displayed.



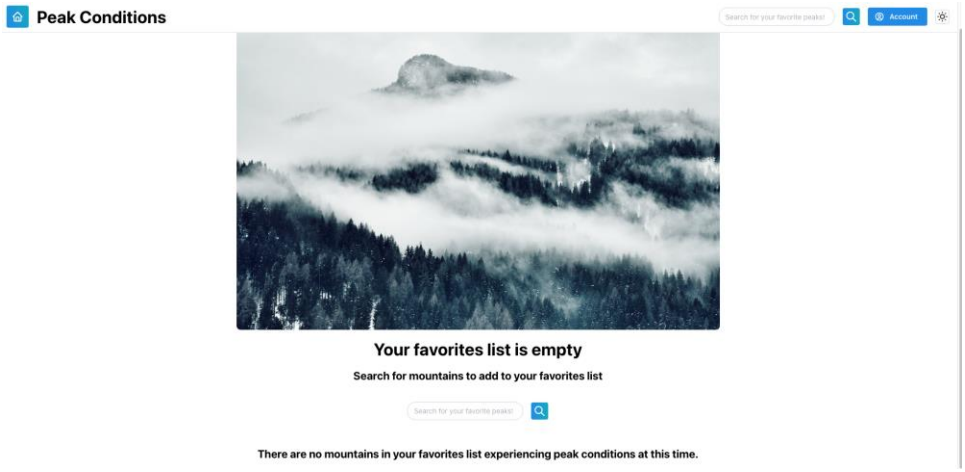
Both the ‘choose a password‘ and ‘confirm your password‘ fields must match or an alert will be displayed and the form will not be submitted.



If a user attempts to register an account with an email address that already is registered, the registration will fail and an alert message will be displayed.

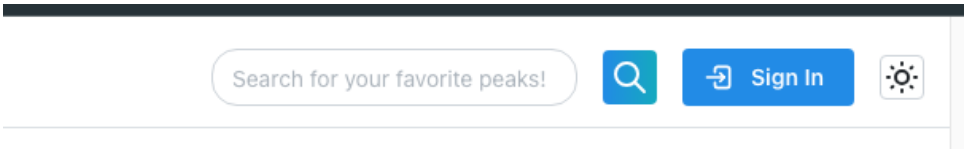


If valid credentials are entered, then the user will be automatically logged in to their new account and they will be redirected to the landing page for logged in users.

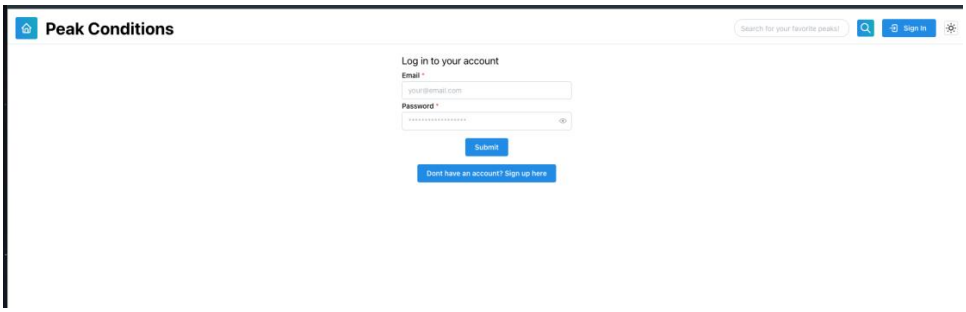


Logging in to your account:

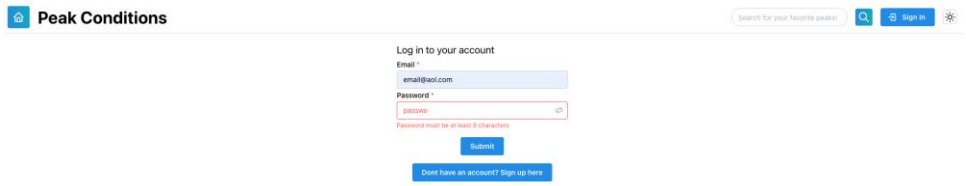
If a user already has an account, they can access the login page by clicking the ‘sign in’ button on the navbar.



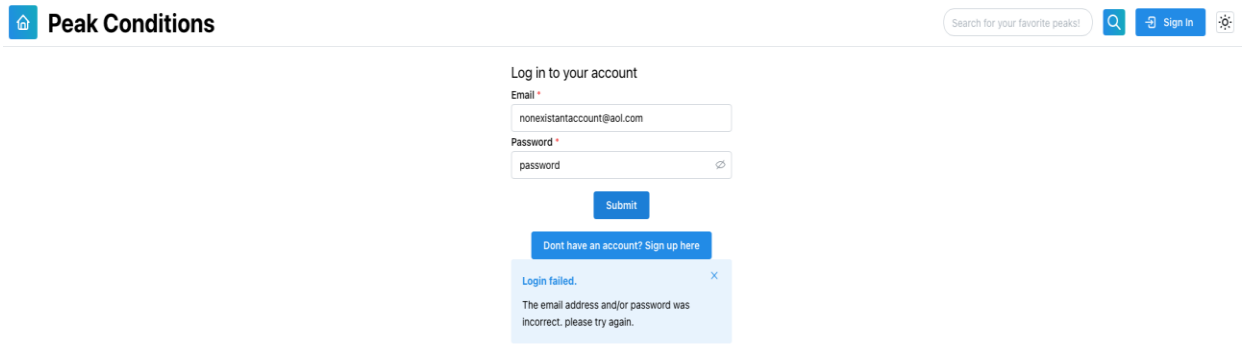
Once on the login page, the user is prompted to enter their credentials in the form of their email address and their password. Once credentials are entered, the user presses the ‘submit’ button to submit the data to the server and log in to the account.



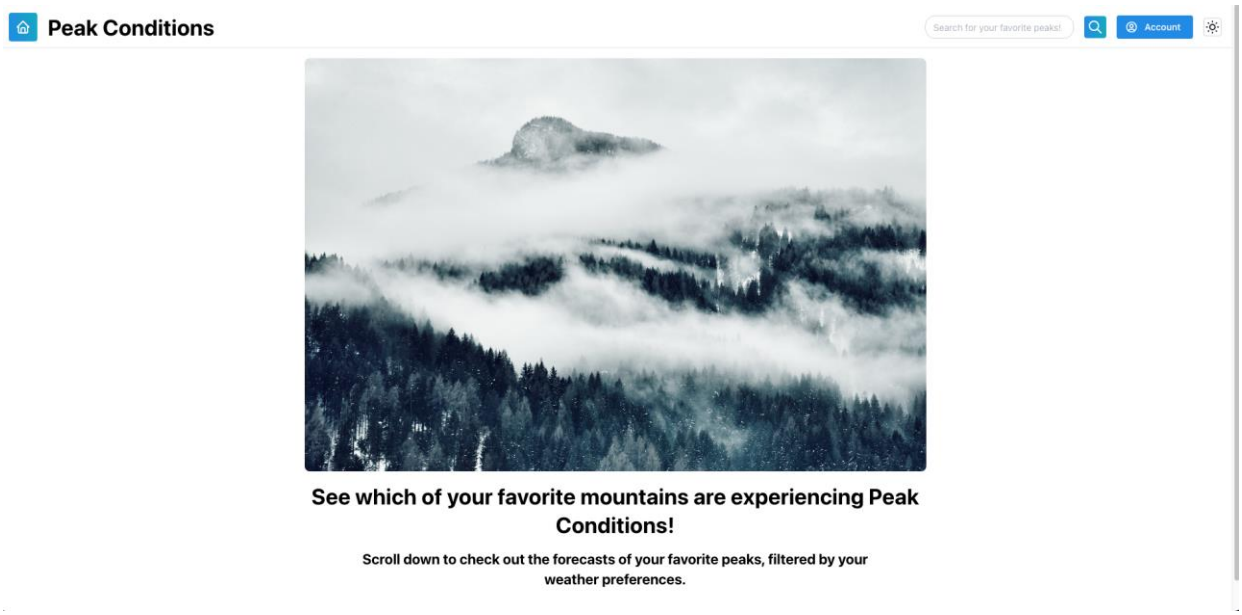
All passwords must be 8 characters in length or more, so if the user attempts to enter a password less than 8 characters the form will not be submitted, and an error message will display.



If there is an error with either the username or password that is submitted and the server rejects the login attempt, log in will fail and an error message will be displayed. The user can dismiss the error by clicking the dismiss button on the error display. The user is not required to dismiss the error to reattempt a login.

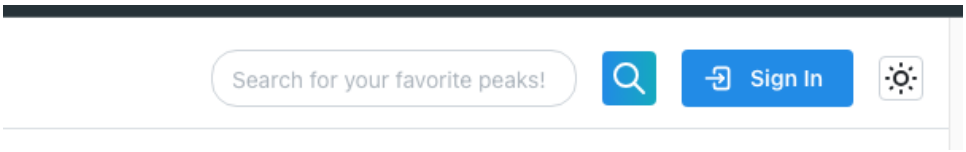


If the user enters valid credentials, they will be redirected to the logged-in home page upon submission.



Searching for mountains or mountain ranges

To see a forecast, first the user must find the mountain peak they would like to see information for. To find mountains, use the search bar in the center of the navbar.



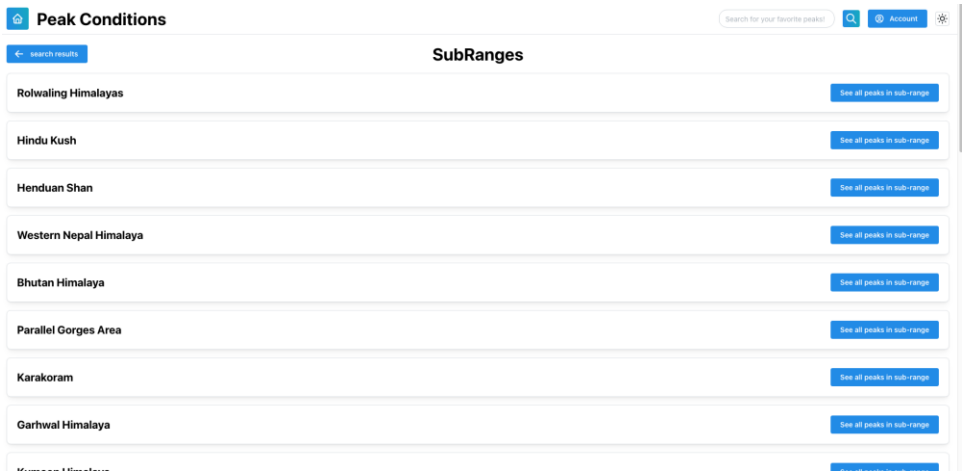
There is also a search bar on the main landing page, below the demo section.

The search bar can be used to search for mountains and mountain ranges by name. The user enters a search query into the search bar text field, and presses the search icon to submit the search query. Once the user has submitted their search, they will be automatically redirected to the search results page where they will see a list of all the mountain peaks, major mountain ranges and smaller mountain subranges.

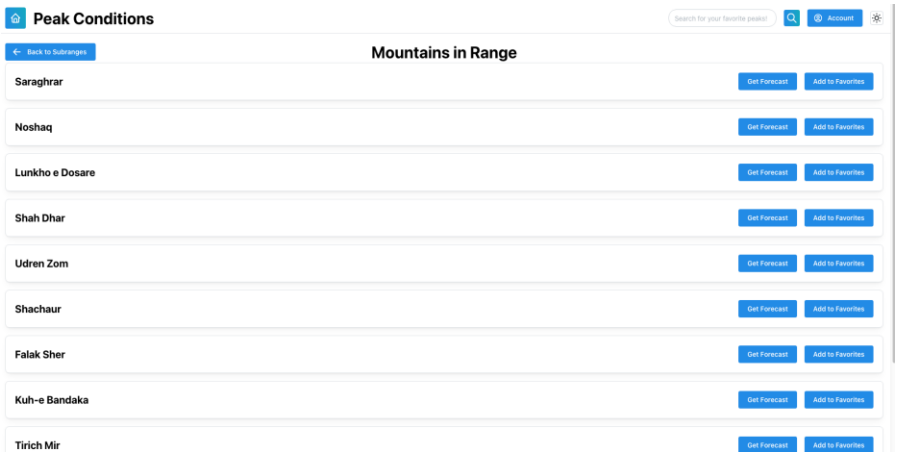
Search results page

The search results page is where the mountain peaks and mountain ranges that in full or in part matched the search query will be displayed. There are filter toggle buttons at the top of the page to select whether to display the mountain peaks, mountain ranges, or mountain subranges, or any combination of the three.

Major mountain ranges displayed in the search results will have a button to display all smaller subranges that make up the major range.



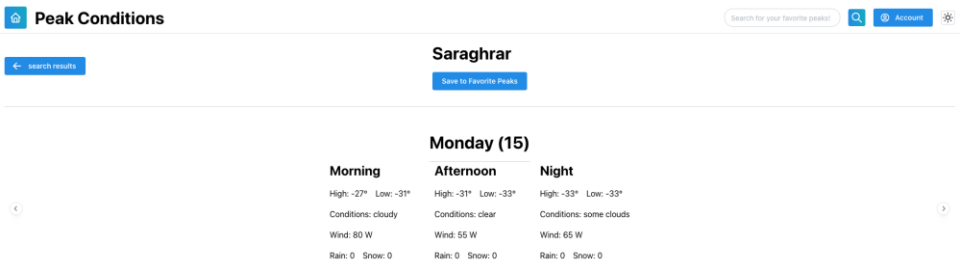
The user can return from the subranges page to the search results with the ‘<= search results’ button. Each subrange entry will have a ‘see all peaks in sub-range’ button that will redirect the user to a list of all the mountains that make up the subrange. the user can return to the subranges page with the ’back to subranges’ button.



If the user is logged in to their account, the ’add to favorites’ button will display next to mountain peak entries in the search results. Clicking these buttons will add the peak to the users favorites list. Upon clicking the button, the button will become disabled to indicate the peak has been added to the list.

Displaying forecasts of searched mountains

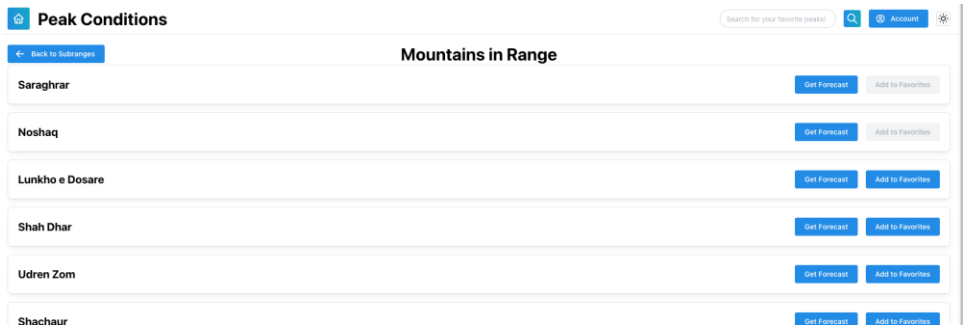
Mountain peak entries will have a ‘Get forecast” button which will redirect the user to a single day forecast page if not logged in, or a 6 day forecast when they are logged in to their account. If a user is logged in, the ’save to favorites’ button will show below the mountain's name. The user can return to the search results with the ’search results’ button.



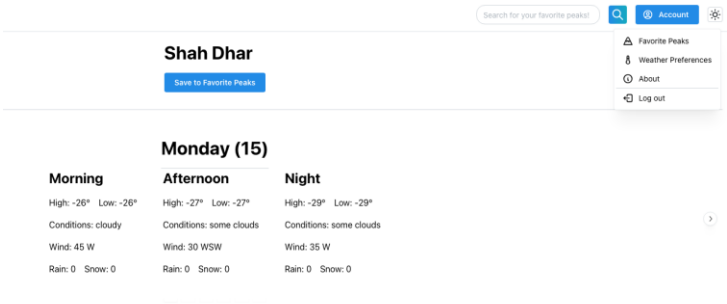
Saving mountains to the favorites list

If the user is logged in to their account when searching for mountains, any mountain peak entry in the search results will have a ‘save to favorites’ button next to it. The user can select as many mountain peaks to save as they choose, the ‘save to favorites’ button will become disabled once it has been

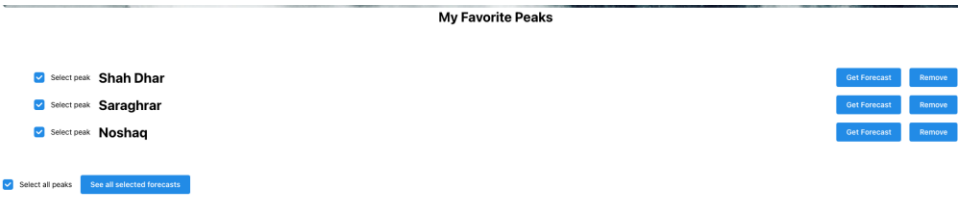
clicked to indicate the peak has been saved.



The user can then navigate to the 'favorite peaks' section using the 'account' drop down on the navbar



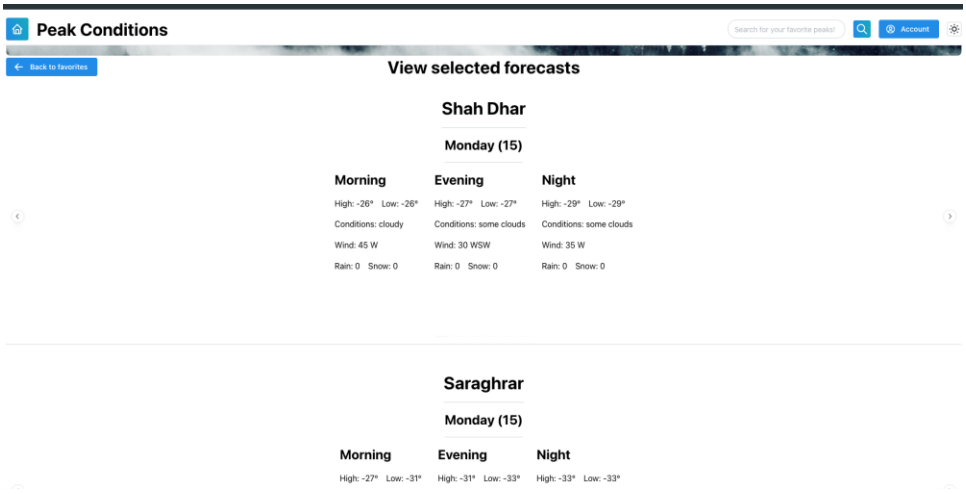
Clicking the 'Favorite Peaks' section on the drop down will redirect the user to the favorite peaks list.



Getting forecasts of mountains from favorites list.

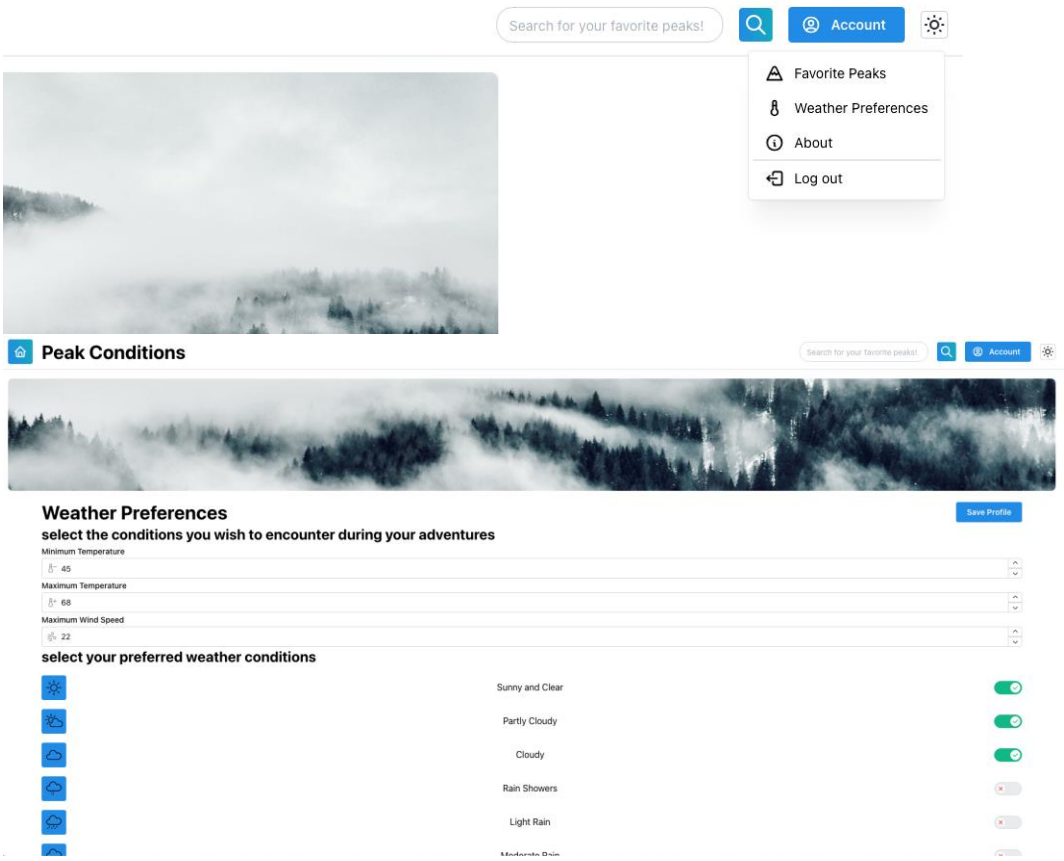
The favorites list page allows the user to see forecasts for one or any combination of mountain peaks from the list. If you want to see the forecast for all of the mountain peaks on the list, select the 'Select all peaks' checkbox next to the 'see all selected forecasts' button.

Select the checkboxes next to each mountain peak whose weather forecast you want to see, and then click on the ‘see all selected forecasts’ button to display the forecasts of the mountain peaks selected from the list.

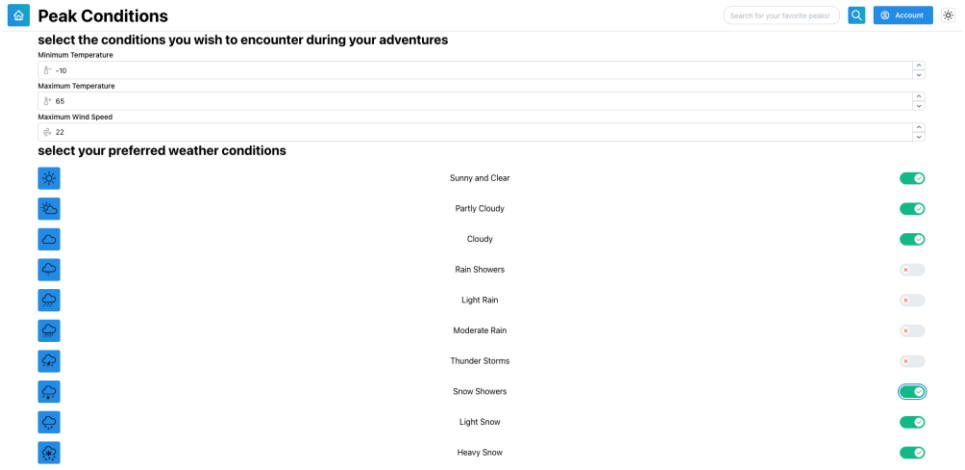


Changing users weather preferences profile.

When the user is logged in to their account, they can access the weather preferences section by clicking on the ‘account’ drop down menu on the navbar, and choosing the ‘weather preferences’ option. The user will then be redirected to the weather preferences page.

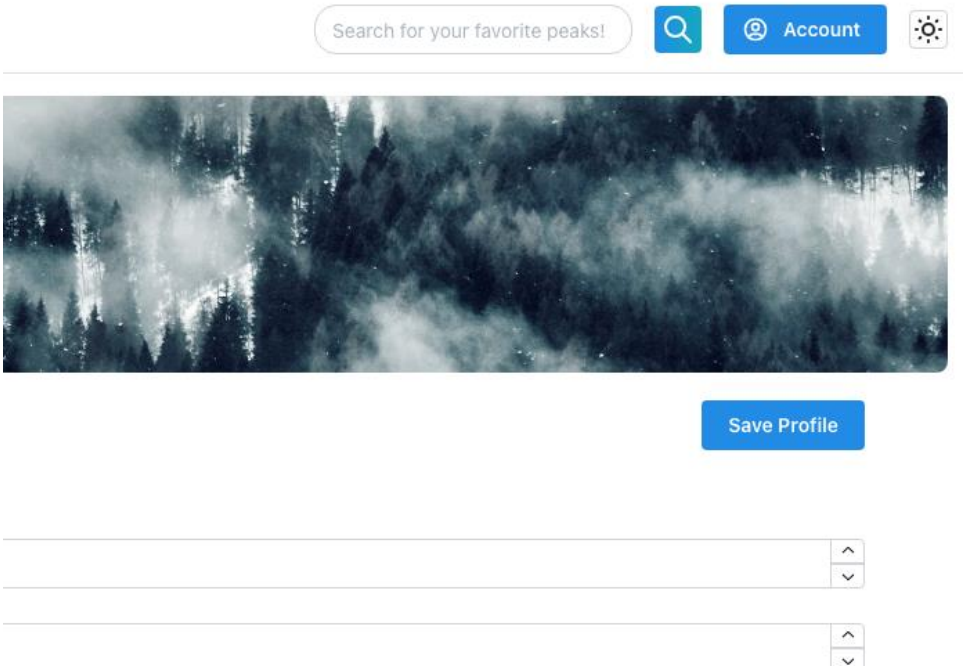


On this page, the user can choose what weather conditions they wish to encounter on their travels. The user can choose the maximum temperature they wish to encounter, as well as the minimum temperature they wish to encounter. Temperatures can be chosen between –35 degrees and 120 degrees. All temperatures are in Fahrenheit. The user can also choose the maximum preferred windspeed. There are toggle switches for weather conditions like Sunny and clear, cloudy, and partly cloudy, as well as various levels of rainy and snowy weather. The user should toggle on whatever weather conditions they are willing to hike in. Toggling 'on' a switch is indicated by the switch turning green. When a toggle is 'off' the switch is grey with a red x.



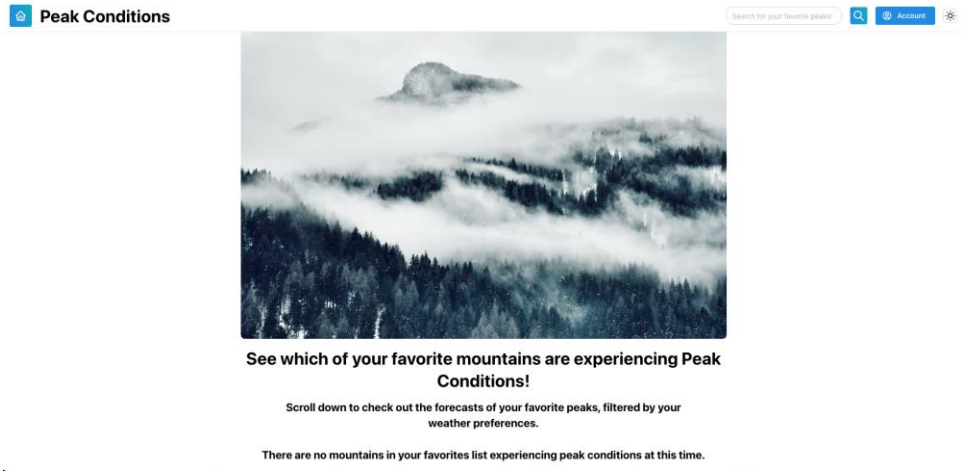
When the use has chosen their preferred weather conditions, they must click the blue 'save profile'

button in the top right corner of the page. This button will save the profile and redirect the user to the logged in home screen.



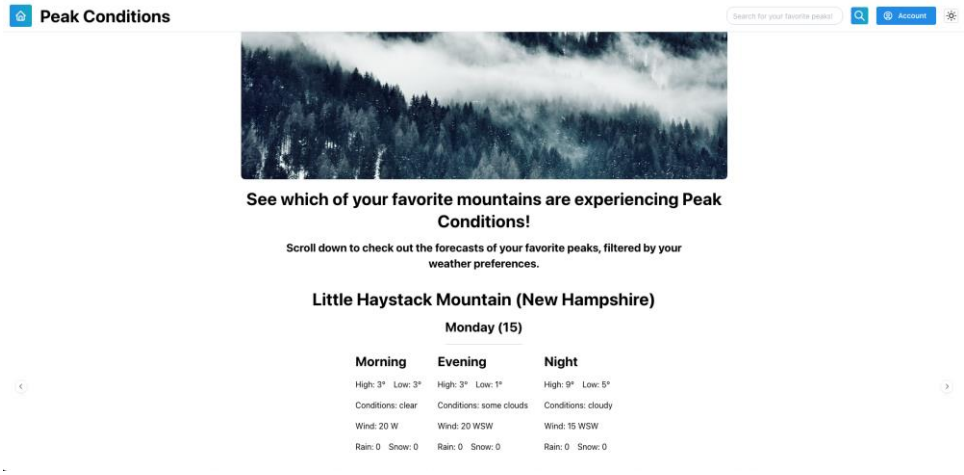
Getting forecasts of all mountains matching weather preferences.

When the user is on the logged in home page, any forecast from their favorites page which also meet the criteria specified in the weather preferences will be displayed. Sometimes, even with mountains in their favorites list, the weather does not align with the users preferences that week. In that case, the user is presented with a message that 'there are no mountains in your favorites list experiencing peak conditions at this time.

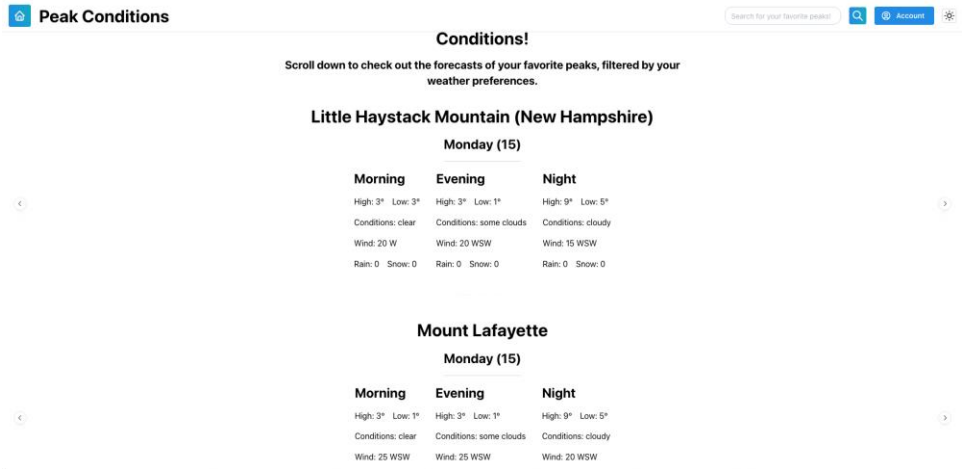


If the user has a mountain or more than one mountain in their favorites list that is expecting weather that meets the criteria defined by the users weather preferences, it will be displayed on the logged in

home page.



Forecasts will be displayed in a vertical list.



Logging out of the application

when the user is finished using the application, or wants to switch to another users account, they can find the log out button in the account drop down menu. Clicking the ‘Log out’ button will sign the user out and return them to the default landing page. The user can also simply close the window, and they

will be automatically logged out of their account.