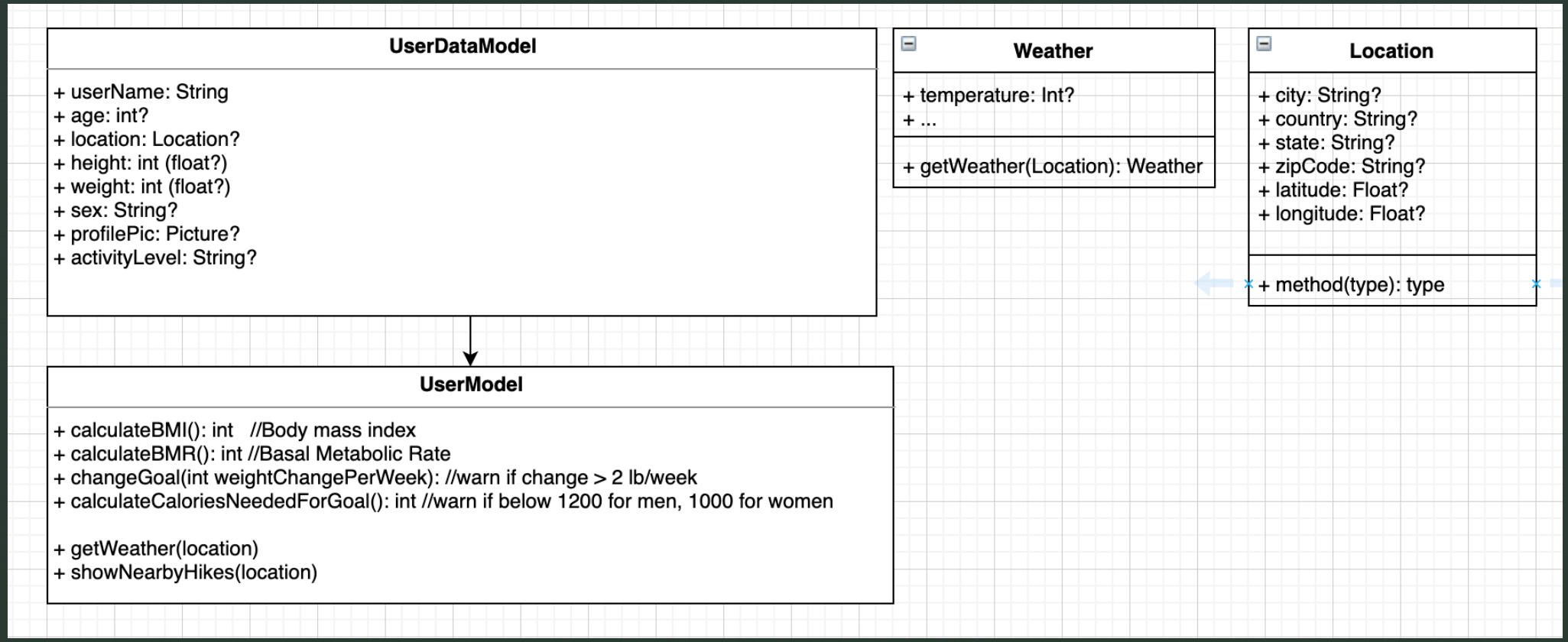
Chad Hurst

Team Lead

Class Diagram



Data encapsulation

```
data class UserDataModel(
    var userName: String,
    var age: Int? = null,
    var city: String? = null,
    var country: String? = null,
    var heightInches: Int? = null,
    var weightLbs: Int? = null,
    var male: Boolean? = null,
    var profilePicture: Picture? = null,
    var activityLevel: String? = null,
    var weightChangeGoalPerWeek: Float? = null,
)
```

```
data class Weather(
    @Json(name = "coord")
    val coord: Coord,
    @Json(name = "main")
    val mainWeather: MainWeather,
    @Json(name = "visibility")
    val visibility: Int,
    @Json(name = "wind")
    val wind: Wind,
)
```

```
return Klaxon().parse<Weather>(reader = StringReader(jsonText))
```

Extensibility

- Everything is highly modular and tested. This allows you to add and remove components without adverse results.
- For example, we can easily add and remove different weather parameters or change the weather source.

Class design choices

- We decided not to use the Trail class that we made and instead just send a “trails near {user.city ?: "me"}” to google maps instead.
- Weather.kt has the same structure as the JSON response from openweather to make for easy parsing.

Coroutines

```
val job : Job  = GlobalScope.launch(Dispatchers.IO) { this: CoroutineScope
    weather = HeavyWorker().suspendGetWeather(Location(city = city, country = country))
    updateView()
}
job.start()
```

```
class HeavyWorker(private val dispatchers: DispatcherProvider = DefaultDispatcherProvider()) {
    suspend fun suspendGetWeather(location: Location): Weather? {
        return withContext(dispatchers.io()) { this: CoroutineScope
            getWeather(location)
        }
    }
}
```