EdWelcome

Design Document - A tour app for the University of Edinburgh

INF2C Software Engineering (inf2c-se) — Coursework 2

Team:
- Elena Lapinskaite (s1605619@inf.ed.ac.uk)
- Qais Patankar (s1620208@inf.ed.ac.uk)

The application

EdWelcome is a GPS-based mobile tour app for prospective students of The University of Edinburgh, who want to get to know the university campuses. The tours include long tours around the main university areas, and more in-depth tours of individual Schools and various University buildings. The Student Ambassadors create and manage the tours. All tours are taken by foot.

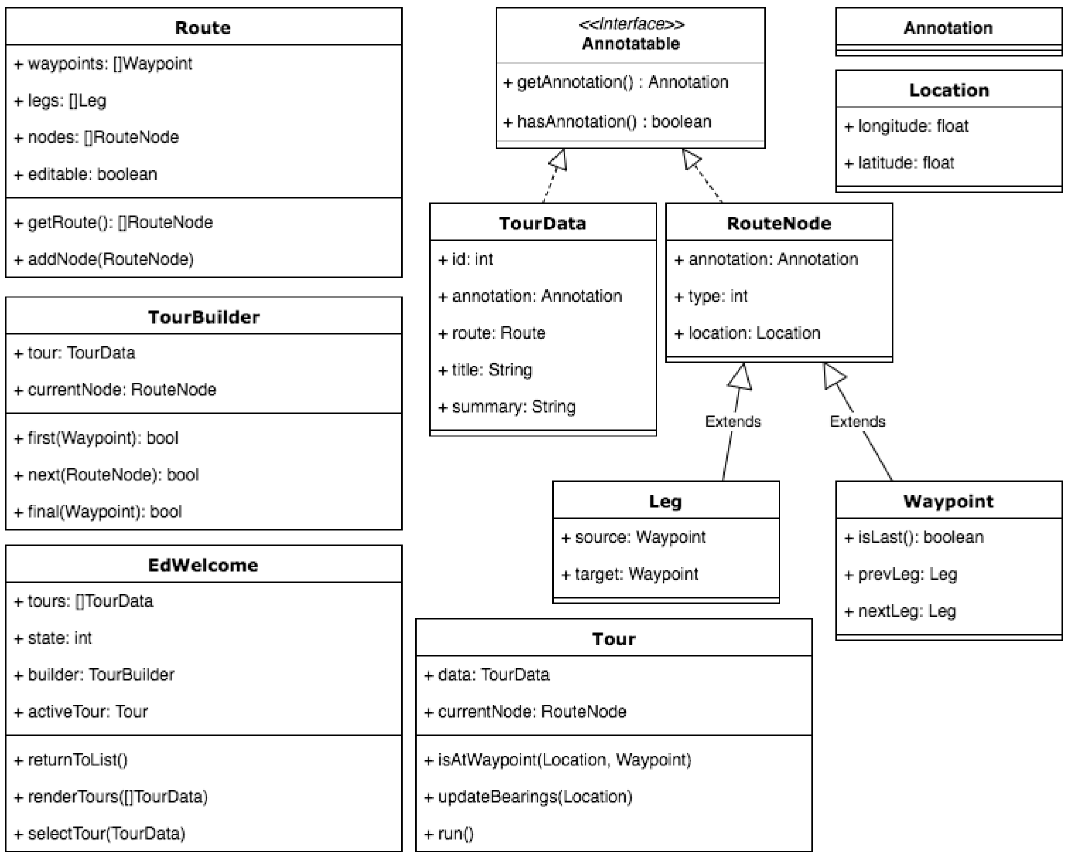
This document outlines the design for the software part of our application. For information about this document, please visit these instructions ("Spec 2"). Further general information about the nature of the app, can be found here ("Spec 1").

Static Model

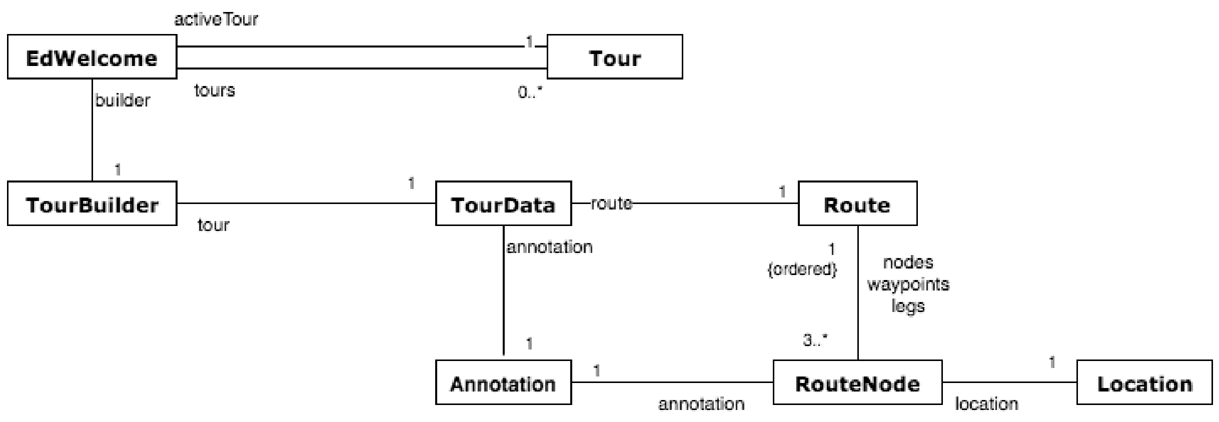
UML class model

Overview

An overview of all the classes, with their attributes/methods. This also includes generalisation arrows or inheritance arrows, where required.



Associations



High-level description

Annotatable

This interface is implemented by anything that could have an annotation, and was created to make it easier to read annotation data from anything that is "annotatable".

At first we noticed that *Waypoint*, *Leg*, and *TourData* all had an *annotation* field, and were implementing the interface fields (without there actually being an interface), and so we created this interface to prevent duplication.

As annotations are all likely to be rendered in a standardised way, having an interface "Annotatable" allows the code that interacts with annotations to be written once (adhering to DRY principles), rather than being written for a tour overview, a waypoint, and a leg.

Annotation

This class represents an annotation, which according to Spec 1 "may be some combination of text, audio, pictures and video giving information".

Spec 2 states that we should completely ignore all details of annotations.

Location

As both there are other entities that have a location associated with them, this utility class represents a physical location with the attributes *longitude* and *latitude*. These attributes are stored as floats, as integers would result in a lack of precision.

Given that this class is written for EdWelcome, it is reasonable for the developer to assume that coordinates refer to a location on Earth.

RouteNode

This class was initially called *RouteComponent*, but was renamed to *RouteNode* as it is more concise. The TourBuilder needs to ensure that each node has a Leg node in between them, and so a *type* field is exposed (in the form of a getter, but not shown here) to help facilitate this.

“Node” also conveys the idea that it is part of a “track” (i.e, part of a sequence of nodes), rather than just a component placed someplace in a system.

As all *RouteNode*s are (optionally) annotatable, we ensured that this class implements *Annotatable* to keep annotation logic DRY.

Note: *type* here should actually be an enum, but would internally be represented as an integer.

Leg

This class extends a *RouteNode* and has a *source* and a *target*.

Whilst a tour is being built, it may not have a *target*. In all other cases, a *RouteNode* will have a *target*. A Leg will always have a *source*.

Since this class extends *RouteNode*, it also implements *Annotatable*.

Waypoint

In a similar fashion to *Leg*, this class maintains references to the previous/next *Leg*, and extends RouteNode.

If this is the final waypoint in the tour, *nextLeg* will not exist. Therefore, *isLast()* can simply figure out whether it is the final Leg by checking whether *nextLeg* is *null*.

As above, since this class extends RouteNode, it also implements *Annotatable*.

TourData

This class was initially called *Tour*, but was renamed to *TourData* as it no longer contains a mixture of general data and stateful data. The stateful data here refers to a tour being followed, information of which can be found here.

Each TourData has an *id*, which may be used in the future as a lightweight reference to this particular tour. It would be particularly useful if any networking support was implemented.

All tours have an annotation associated with them, and so this class implements *Annotatable*.

Tour

The *Tour* class (backstory above) is stateful, and used whilst the user is currently following a tour.

Since an external API would be used, we don't need to store GPS or Location data here, so we only track what the current node is.

Route

Having a *nodes* field might feel counterintuitive as we already implement a "track" in the form of a "linked list". The "linked list" method works well whilst following nodes during the lifetime of a followed tour, but it does not work well when trying to view an overview of all the nodes, or trying to visit or find a particular node. This *nodes* field is a remedy to this problem.

Under no circumstances should the *nodes* field fall out of sync with the "track" represented by "linked list".

Note: When we say it is implemented in the form of a linked list (or refer to "linked list"), we are just saying that each leg refers to the neighbouring waypoints, and each waypoint has a reference to the neighbouring legs.

TourBuilder

This class is used during for the Author Tour use case, and builds a new piece of *TourData*. It enforces the ordering of nodes when creating a *Route*, and sets the previous/next legs/waypoints for the corresponding nodes.

This separates the logic for regular use (following tours) from the logic for creating tours, making the code much cleaner. This may even potentially result in a performance boost as memory will not be allocated for any of *TourBuilder*'s fields when simply following a tour.

EdWelcome

This is the primary controller class for the entire application (since we have modelled only one controller class, we have omitted "Controller" from the class name).

This contains fields suitable for all three use-cases, and not all fields would have a value in all situations. For example, the *builder* field would only have a *TourBuilder* assigned if the user is currently authoring a tour.

The *state* field allows the application to know what mode it is in. In the diagram this is represented as integer, but is actually be an enum, and is only represented as an integer as that is how it would be done internally.

Dynamic Models

Follow Tour

General idea for Follow Tour

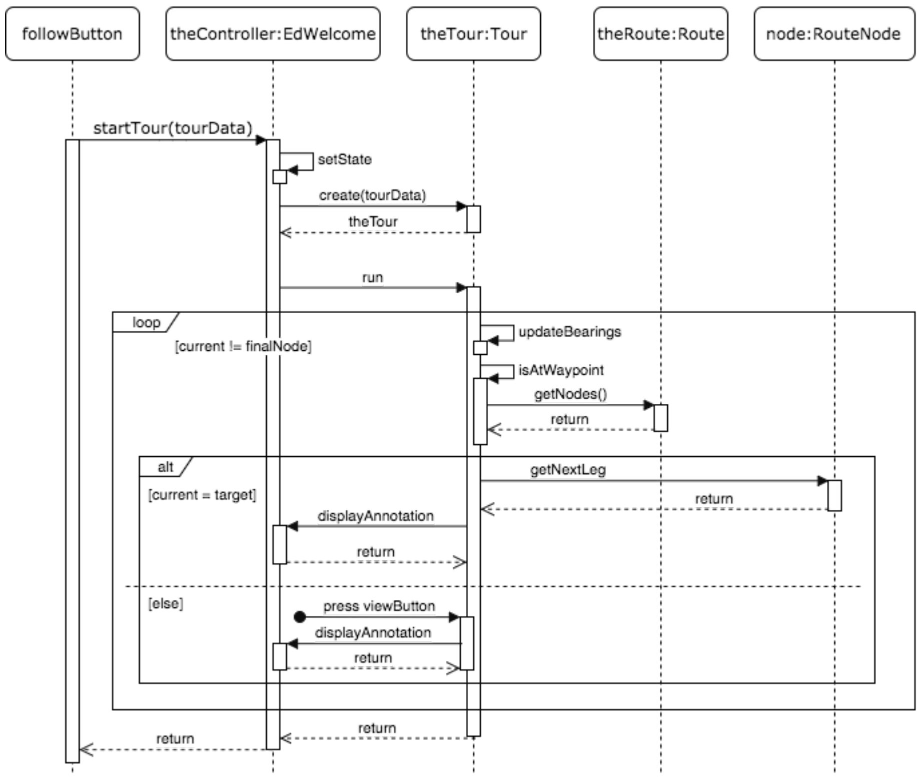
```
followButton -- startTour(tourData) --> theController
        theController -- setState(FOLLOW_TOURS) --> theController
        theController -- create(tourData) --> Tour
        theController <- - - - - - - - - - -  Tour // EdWelcome.activeTour now set

        theController -- run --> Tour
                LOOP [whilst current is not at the final node]
                        theTour -- updateBearings(location) --> theTour
                        theTour <- - - - - - - - - - - - - - -  theTour

                        theTour -- isAtWaypoint(location, targetWaypoint) --> theTour
                                theTour -- getNodes() --> theRoute
                        theTour <- - - - - - - - - - - - - - - - - - - - -  theTour

                        IF [current is at targetWaypoint] THEN // reached target waypoint
                                theTour -- getNextLeg() --> node
                                theTour <- - - - - - - - -  node
                                theTour -- displayAnnotation(node) --> theController
                        ELSE // on a leg
                                aUser -- press() --> viewButton
                                        viewButton -- displayAnnotation(current) -> theTour
                        ENDIF
                ENDLOOP
```

Sequence Diagram



Browse Tours (behaviour description)

```
aUser -- press() --> browseToursButton
        browseToursButton -- setState(BROWSE_TOURS) --> theController
                theController -- getTourDatas() --> theController
                theController <- - - - - - - -  theController
                theController -- renderTours(tourDatas) --> theController
                        LOOP [for each tourData in tourDatas]
                                theController -- addData(tourData) --> theTourList
                        ENDLOOP

                LOOP [while user not decided to follow a tour]
                        tourButton -- selectTour(tourData) --> theController
                                theController -- displayAnnotation --> theController
                                        followButton -- setState(FOLLOW_TOURS) --> theController
                                                // see Follow Tour sequence diagram
                                        closeButton -- returnToList() --> theController
                ENDLOOP
```

Author Tour (behaviour description)

```
aUser -- press() --> authorTourButton
        authorTourButton -- setState(AUTHOR_TOUR) --> theController
                theController -- create() --> TourBuilder
                theController <- - - - - - -  TourBuilder // EdWelcome.builder now set

                aUser -- updateAnnotation() --> tourAnnotation
                aUser -- first(Waypoint) --> theTourBuilder

                LOOP [whilst final waypoint not created]
                        theTour -- updateBearings(location) --> theTour
                        theTour <- - - - - - - - - - - - - - -  theTour

                        aUser -- press() --> viewButton
                                viewButton -- displayAnnotation(currentLeg) --> theTourBuilder

                        aUser -- press() --> writeButton
                                aUser -- updateAnnotation() --> legAnnotation

                        aUser -- press() --> addWaypointButton
                                aUser -- updateAnnotation() --> nodeAnnotation

                        aUser -- press() --> addFinalWaypointButton
                                aUser -- updateAnnotation() --> nodeAnnotation
                ENDLOOP

                theTourBuilder -- getTourDatas() --> theController
                        theTourBuilder -- addTourData -->  theTourDatas
```