

1 Componenti e classi

1.1 BlueWhere

Namespace globale dell'applicazione. Le relazioni tra i package Model, View e Presenter rappresentano le relazioni tipiche del design pattern MVP.

1.1.1 Package contenuti

- BlueWhere::Model;
- BlueWhere::View;
- BlueWhere::Presenter.

1.2 BlueWhere::Model

1.2.1 Struttura del package

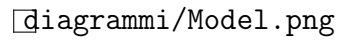
diagrammi/Model.png

Figura 1: Struttura del pacchetto Model

1.2.2 Descrizione

Package che contiene tutte le classi e package appartenenti al Model dell'applicazione.

1.2.3 Package contenuti

- BlueWhere::Model::UserSetting;
- BlueWhere::Model::Navigator;
- BlueWhere::Model::Beacon;
- BlueWhere::Model::DatabaseServices.

1.3 BlueWhere::Model::UserSetting

1.3.1 Struttura del package

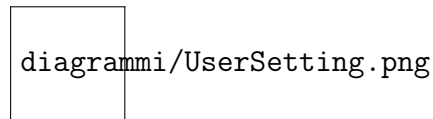


Figura 2: Struttura del pacchetto UserSetting

1.3.2 Descrizione

Componente del Model per la gestione delle preferenze riguardanti il percorso, le impostazioni di fruizione delle informazioni di navigazione e per gestire se un utente può accedere alle funzioni di sviluppatore oppure no.

1.3.3 Classi

1.3.3.1 BlueWhere::Model::UserSetting::Setting Classe per accedere alle impostazioni di un utente.

Viene utilizzata per mantenere un riferimento in memoria e accedere a queste informazioni quando serve (per esempio quando viene effettuato il calcolo del percorso). Le preferenze vengono salvate tramite la classe "SharedPreference" di Android.

1.3.3.2 BlueWhere::Model::UserSetting::PathPreference Classe che definisce le preferenze impostabili riguardanti il percorso preferito dall'utente.

È un semplice enumeratore utile solamente per definire tutti e soli i valori che possono essere utilizzati in BlueWhere::Model::UserSetting::Setting. I valori previsti da questa classe sono:

- DEFAULT;
- NO_STAIR; NO_ELEVATOR.

1.3.3.3 BlueWhere::Model::UserSetting::InstructionPreference Classe che definisce le preferenze impostabili riguardanti la fruizione delle istruzioni di navigazione.

È un semplice enumeratore utile solamente per definire tutti e soli i valori che possono essere utilizzati in BlueWhere::Model::UserSetting::Setting. I valori previsti da questa classe sono:

- DEFAULT;
- TTS;
- SONAR.

1.3.3.4 BlueWhere::Model::UserSetting::DeveloperCodeManager Classe statica per la verifica dei codici sviluppatore.

Questa classe esporrà un metodo pubblico che ricevuta una stringa ritornerà un booleano che specificherà se il codice passato come parametro è valido oppure no. In prima battuta possiamo fare questo controllo hard-coded, con dei raffinamenti invece possiamo mettere sul server i codici sviluppatore.

1.4 BlueWhere::Model::Beacon

1.4.1 Struttura del package

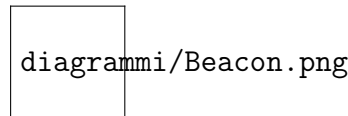


Figura 3: Struttura del pacchetto Beacon

1.4.2 Descrizione

Componente del Model per la gestione dei beacon.

1.4.3 Interfacce

1.4.3.1 BlueWhere::Model::Beacon::MyBeacon Interfaccia per accedere alle informazioni di un beacon.

Viene utilizzata per specificare il contratto delle classi che la implementano, definendo i metodi che devono essere implementati. Praticamente serve per offrire un livello di astrazione al posto di avere direttamente una classe concreta e quindi poter creare un "adapter" per il beacon della proposto da Altbeacon.

1.4.4 Classi

1.4.4.1 BlueWhere::Model::Beacon::MyBeaconImp Classe che implementa l'interfaccia BlueWhere::Model::Beacon::MyBeacon.

Questa classe rappresenta un beacon e permette di accedere alle sue informazioni tramite i metodi definiti nell'interfaccia da cui deriva.

1.5 *****Manca la parte di rilevamento dei beacon*****

1.6 BlueWhere::Model::Navigator

1.6.1 Struttura del package Navigator


 diagrammi/Navigator.png

Figura 4: Struttura del pacchetto Navigator

1.6.2 Descrizione

Componente del Model che si occupa della parte di navigazione. Questo pacchetto ha il compito di calcolare il percorso che l'utente deve seguire ed offrire le classi che permettono di rappresentare la mappa.

1.6.3 Interfacce

1.6.3.1 BlueWhere::Model::Navigator::Vertex Interfaccia che rappresenta un vertice di un grafo.

Viene utilizzata per dare un livello di astrazione molto alto rispetto a ciò che effettivamente utilizzeremo durante la navigazione. Definisce il contratto di metodi molto semplici come metodi per accedere all'identificativo di un certo vertice.

1.6.3.2 BlueWhere::Model::Navigator::Edge Interfaccia che rappresenta un arco pesato di un grafo.

Viene utilizzata per dare un livello di astrazione molto alto rispetto a ciò che effettivamente utilizzeremo durante la navigazione. Definisce il contratto di metodi molto semplici come metodi per accedere al peso dell'arco, ai nodi iniziali e finali dell'arco, impostare i nodi dell'arco.

1.6.3.3 BlueWhere::Model::Navigator::PathFinder Interfaccia che espone il contratto della classe che calcola il percorso.

Questa interfaccia viene utilizzata per definire la firma dei metodi per gli algoritmi che possono calcolare il percorso definendo uno "Strategy" (Non ho idea se in modo corretto).

1.6.3.4 BlueWhere::Model::Navigator::BuildingMap Interfaccia che rappresenta la mappa di un edificio.

Viene utilizzata per definire il contratto che le classi devono seguire per poter rappresentare una mappa utilizzabile dal nostro Model.

1.6.3.5 BlueWhere::Model::Navigator::NavigationInstruction Interfaccia che rappresenta le informazioni di navigazione.

1.6.4 Classi

1.6.4.1 BlueWhere::Model::Navigator::VertexImp Classe che implementa l'interfaccia BlueWhere::Model::Navigator::Vertex.

Questa classe rappresenta un vertice e permette di accedere alle sue informazioni tramite i metodi definiti nell'interfaccia da cui deriva.

1.6.4.2 BlueWhere::Model::Navigator::RegionOfInterest Classe che estende BlueWhere::Model::Navigator::VertexImp e BlueWhere::Model::Beacon::MyBeaconImp.

Questa classe rappresenta un "punto di navigazione" nel grafo rappresentante l'edificio, ovvero rappresenta un beacon come un vertice nel grafo dell'edificio.

1.6.4.3 BlueWhere::Model::Navigator::EnrichedEdge Classe che implementa l'interfaccia BlueWhere::Model::Navigator::Edge e deriva dalla classe org.jgrapht.graph.DefaultWeightedEdge della libreria JGraphT.

Viene utilizzata per la navigazione. Infatti questo tipo di arco contiene informazione sia sul peso dell'arco(utile per il calcolo del percorso), sia l'informazione di attraversamento(informazioni di base, descrizione lunga, URL delle fotografie).

1.6.4.4 BlueWhere::Model::Navigator::ElevatorEdge Classe che estende la classe BlueWhere::Model::Navigator::EnrichedEdge.

Rappresenta un arco che prevede un ascensore e viene utilizzata per ridefinire il peso utilizzato per calcolare il percorso sulla base delle preferenze dell'utente in base agli ascensori.

1.6.4.5 BlueWhere::Model::Navigator::StairEdge Classe che estende la classe BlueWhere::Model::Navigator::EnrichedEdge.

Rappresenta un arco che prevede delle scale e viene utilizzata per ridefinire il peso utilizzato per calcolare il percorso sulla base delle preferenze dell'utente in base alle scale.

1.6.4.6 BlueWhere::Model::Navigator::DefaultEdge Classe che estende la classe BlueWhere::Model::Navigator::EnrichedEdge.

Rappresenta un arco che non contiene particolari ostacoli e viene utilizzata

per ridefinire il peso utilizzato per calcolare il percorso nel caso in cui l'utente non imposti alcuna preferenza di percorso.

1.6.4.7 BlueWhere::Model::Navigator::BasicInstruction Classe che implementa l'interfaccia `BlueWhere::Model::Navigator::NavigationInstruction`. Rappresenta le indicazioni di base che possono guidare un utente (Navigazione di primo livello).

1.6.4.8 BlueWhere::Model::Navigator::DetailedInstruction Classe che implementa l'interfaccia `BlueWhere::Model::Navigator::NavigationInstruction`. Rappresenta delle informazioni aggiuntive che possono essere fornite ad un utente (descrizione lunga).

1.6.4.9 BlueWhere::Model::Navigator::PhotoInstruction Classe che implementa l'interfaccia `BlueWhere::Model::Navigator::NavigationInstruction`. Rappresenta delle informazioni visuali del prossimo punto da raggiungere per procedere con la navigazione (fotografie).

1.6.4.10 BlueWhere::Model::Navigator::PointOfInterest Classe che rappresenta un POI ovvero un area dell'edificio che può risultare interessante ad un utente a livello sia informativo che di navigazione (una possibile destinazione).

Questa classe offre la possibilità di accedere alle informazioni di un POI quali:

- nome del POI (esistente o dato da noi);
- descrizione del POI (qualcosa che almeno ne descriva le funzionalità), fatta tramite la classe `PointOfInterestInformation`;

1.6.4.11 BlueWhere::Model::Navigator::PointOfInterestInformation

Classe che rappresenta le informazioni relative ad un POI.

Viene utilizzato per mantenere la descrizione di un POI in modo tale che sia possibile modificare la struttura della stessa, al posto di avere la descrizione internamente alla classe POI e dover, in caso, modificare tale classe.

1.6.4.12 BlueWhere::Model::Navigator::BuildingMapImpl Classe che implementa l'interfaccia `BlueWhere::Model::Navigator::BuildingMapImpl`.

Viene utilizzata per rappresentare la mappa dell'edificio, racchiudendo le informazioni dell'edificio stesso. In particolare mantiene la corrispondenza tra `RegionOfInterest` e `PointOfInterest` (ad un ROI possono essere associati

più POI ed un POI può appartenere a più ROI. Esempio 1C150 della Torre Archimede).

1.6.4.13 BlueWhere::Model::Navigator::PointOfInterestInformation

Classe che rappresenta le informazioni relative ad un certo edificio.

Viene utilizzato per mantenere le informazioni di un edificio quali:

- nome;
- descrizione (che ne descriva le funzionalità);
- orari;
- indirizzo.

Questa classe è utile per mantenere la descrizione di un edificio in modo tale che sia possibile modificarne la struttura senza per forza dover modificare la classe BuildingMapImpl.

1.6.4.14 BlueWhere::Model::Navigator::DijkstraPathFinder Classe che implementa l'interfaccia BlueWhere::Model::Navigator::PathFinder.

Viene utilizzata per calcolare il percorso per portare un utente in un certo POI (destinazione della navigazione). Questa classe sfrutta gli algoritmi messi a disposizione dalla libreria JGraphT. Essendo che un POI può essere associato a più ROI e la navigazione è possibile solamente tramite i ROI (punti in cui abbiamo i beacon e quindi possiamo controllare se l'utente va nel percorso giusto oppure no) allora sarà necessario calcolare un percorso per ogni ROI associato ad un POI e successivamente scegliere quello con peso minore (in base al percorso stesso e alle preferenze dell'utente). Utilizzando l'algoritmo di Dijkstra non sarebbe necessario in generale ma l'implementazione fornita da JGraphT richiede un ricalcolo.

1.6.4.15 BlueWhere::Model::Navigator::Navigator Classe che si occupa di gestire la navigazione.

Viene utilizzata per controllare se l'utente segue le indicazioni fornite per la navigazione e gestire in generale i percorsi restituiti da Dijkstra (liste di archi, nel nostro caso possibilmente sarebbe meglio EnrichedEdge).

1.7 Dubbi & ToDo

- manca la parte del model riguardante il rilevamento dei beacon;
- manca la parte del model riguardante database e server;

- manca la parte del model riguardante le fotografie e la loro gestione;
- verifica della parte del model riguardante la gestione del codice sviluppatore;
- manca la parte del presenter;
- manca la parte della view;
- da valutare la gestione dei valori dei beacon che cambiano velocemente.
- orario una classe a sé stante