Bilkent University
Department of Computer Engineering

Senior Design Project

Project short-name: Overfind

Low Level Design Report

Group Members:
Asena Rana Yozgatlı
Bartu Soykök
Burak Şenel
Mehmet Emre Arıoğlu

Supervisor:
Prof. Dr. Varol Akman

Jury Members:
Prof. Dr. Özgür Ulusoy
Assoc. Prof. Dr. Çiğdem Gündüz Demir

Innovation Expert:
Bora Güngören (Portakal Teknoloji)

February 12, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

We are living in an era of information. Every one of us is bombarded with information around us in a sense of intensity. Events are happening in a specific time frame so it is important to notice the events in a timely manner. Most of the people don't have the time to look for events according to their interests or profession and sometimes they miss interesting events due to this intense information flow. The aim of our project is to utilize different event platforms by parsing information about events to make recommendation to our app's users.

System will offer some functionalities to improve the quality of usage. User can choose an event that fits his/her interests to see the event details. User can open external links to websites of individual events if available. User can ask for more recommendations. User can search the database for events. User can sort the search results by date or relevance. User can change the language and set a default location.

System consists of three layers: Presentation layer, application layer, and data layer.

Presentation layer represents the client application and it will run on Android OS version 4.4 (KitKat) [1] or later and can be downloaded from the Google Play Store [2]. This layer is composed of 3 parts: User interface for users to interact with the system; data manager to store and manipulate event and user information; controller that interacts with the server and manages data.

Mobile application will collect the initial user data by making the user fill a small survey, it will collect the rest of user data from user feedback and store locally in the device. It will communicate with the server by using https GET method and response will return in JSON format.

Mobile application will use the device ID to create an initial record when the app launches for the first time. When the mobile application is uninstalled from the device, it will not be possible to restore user data. Mobile application will require storage and network access of the device.

Application layer represents the server application and it will run on a Linux machine. This layer is composed of 3 parts. Client manager that interacts with clients and handles client requests; event manager that interacts with web API's to collect and extract meaning from event information; database handler that interacts with the database server to read or write data.

Server will get user data from client and generate recommendations using the tag relevancy information. Server will not store user data. Client requests will be

taken by https GET method and response will sent in JSON format. Server will query database by using neo4j.driver.

Data layer represents the database. Database will run on a Linux machine. We will use a graph database for our storage purpose. We will use Neo4J [3]. Only server can reach the database.

In this report, we aim to provide an overview of the low-level architecture and design of the system. Firstly design trade-offs are described. Guidelines for our documentation and the engineering standards we used in our project are listed. Packages in our system and a detailed class diagram is given. Also interfaces of all classes in all packages are included with their descriptions.

1.1. Object Design Trade-Offs

1.1.1. Usability vs. Functionality

In order to keep the application simple and easy to use, we degraded the functionalities to a small amount while having all of the necessary functionalities to achieve our objective.

1.1.2. Memory vs. Cost

We are storing only essential information but the amount of data will always increase. So, memory requirement of the application will be high. This will increase the cost of the system.

1.1.3. Performance vs. Data Usage

In order to improve the performance we do all graph management operations and queries on the server side. But doing so increases the amount of data transfer between client and server.

1.1.4. Portability vs. Cost

Due to time constraint, we will implement our mobile application only for devices that run on Android OS. 94.3% of Android users have versions 4.4 or above running on their devices [4]. For this reason, we have decided to support versions 4.4 and above.

1.2. Interface Documentation Guidelines

Class Name	Name of the class/interface
Class Description	Description of the class/interface
Package	Package of the class/interface
Attributes	Attribute Name : Attribute Type
Operations	Operation Name (Parameter Type) : Return Type

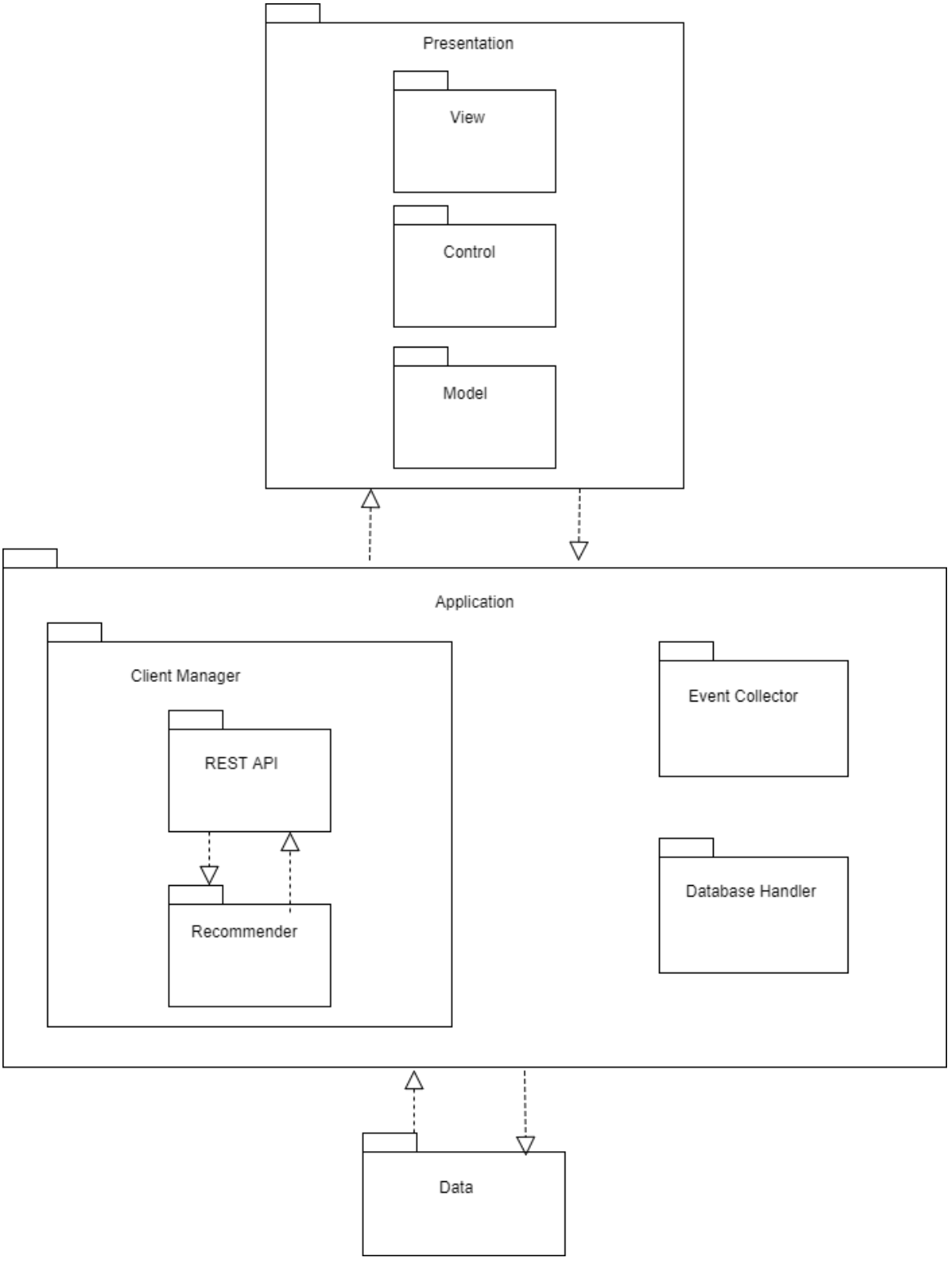
1.3. Engineering Standards

We are using IEEE Editorial Style Manual in this report as a guide for formatting [5]. Our purpose is to make our documentation credible and more international in terms of format. UML To describe our class interfaces, use cases and use case scenarios, we will use UML design specifications defined in UML 2.5.1 documentation in our reports [6].

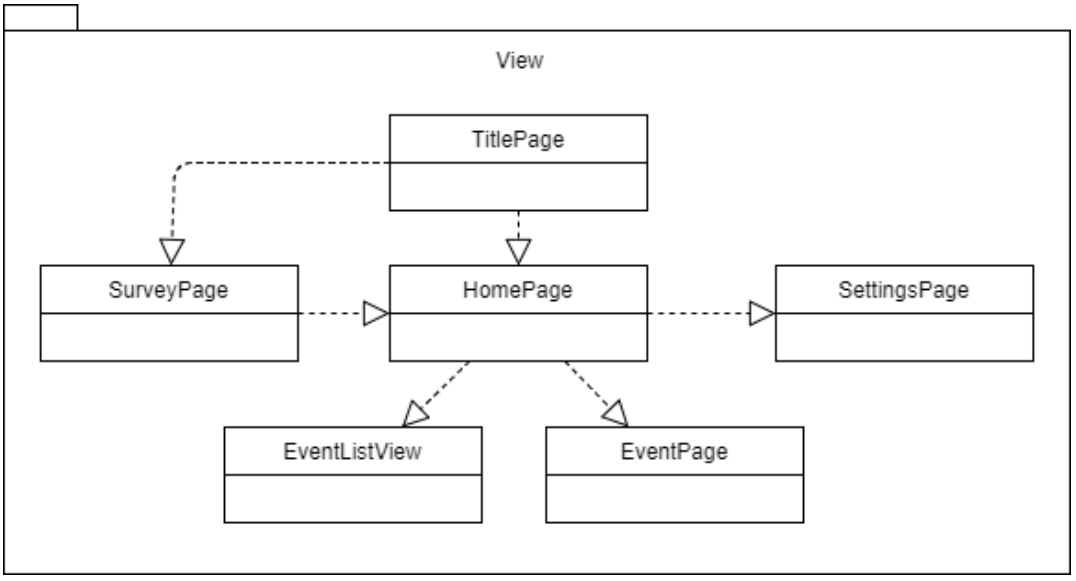
1.4. Definitions, Acronyms, and Abbreviations

API: Application Programming Interface
IEEE: Institute of Electrical and Electronics Engineers
OS: Operating System
REST: Representational State Transfer
UML: Unified Modeling Language

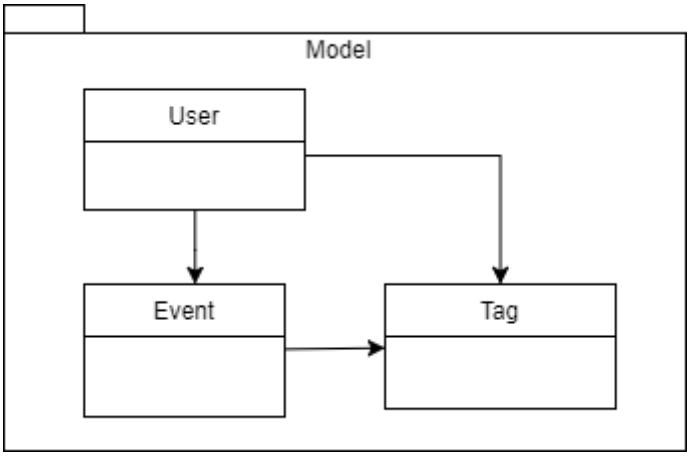
2. Packages


Presentation
View
Control
Model
Application
Client Manager
REST API
Recommender
Event Collector
Database Handler
Data

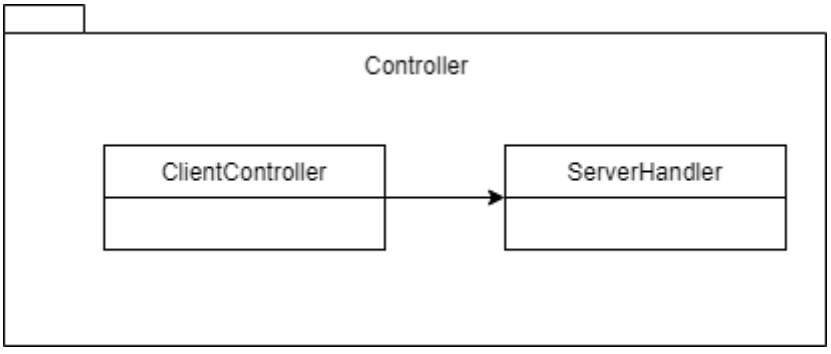
2.1. View



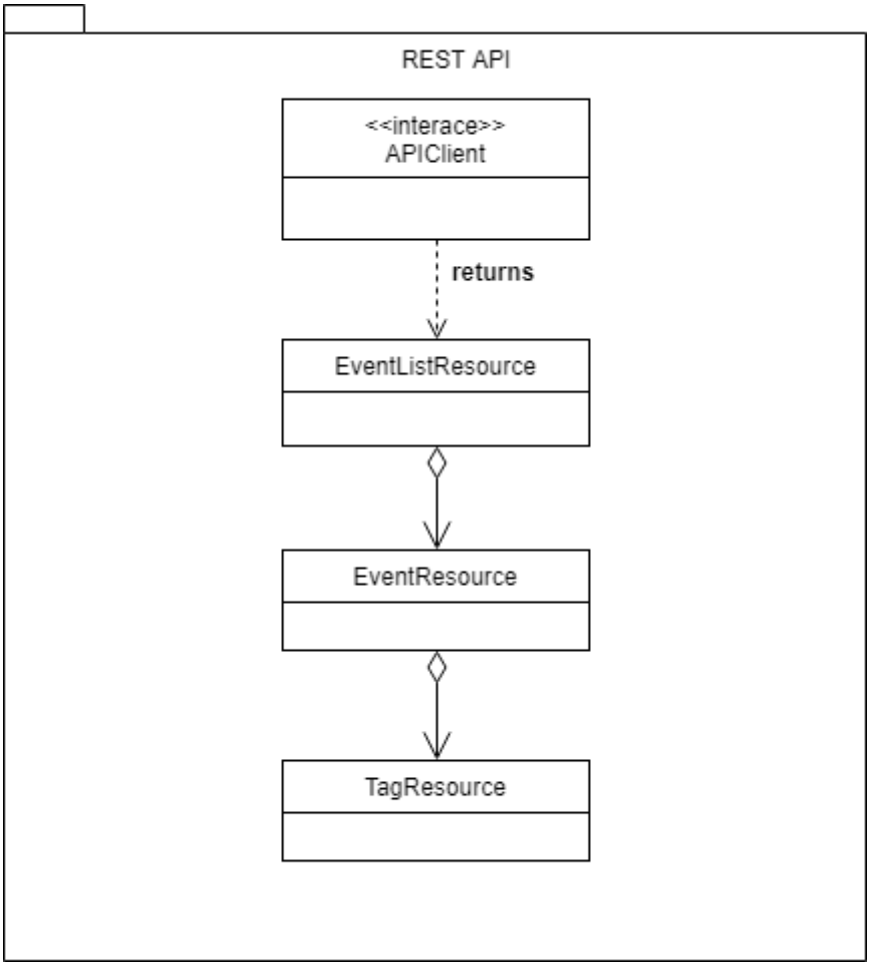
2.2. Model



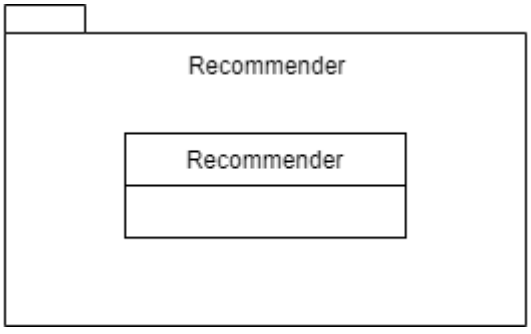
2.3. Controller



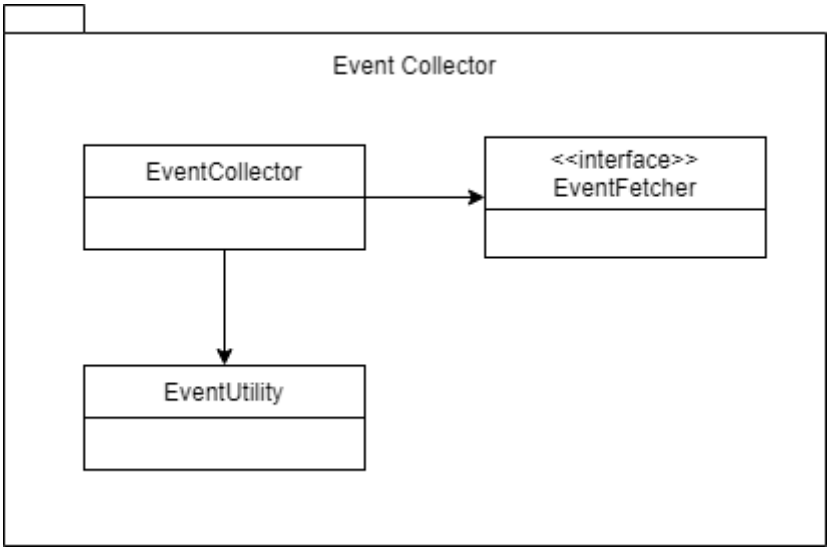
2.4. REST API



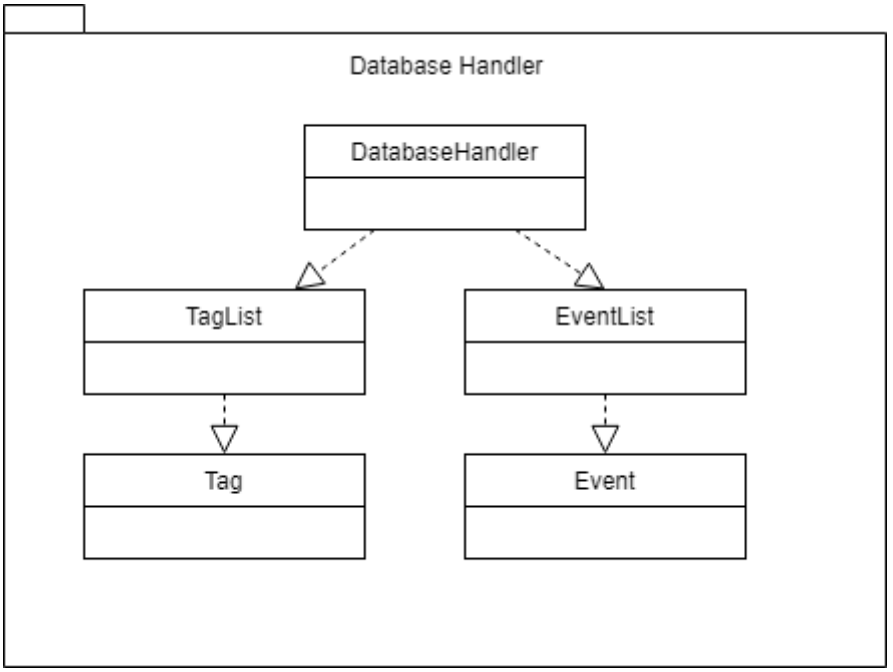
2.5. Recommender



2.6. Event Collector



2.7. Database Handler



3. Class Interfaces

3.1. Presentation Layer

3.1.1. Title Page

Class Name	TitlePage
Class Description	Creates the GUI for title page. This page also checks if the app is being used for the first time and redirect the user accordingly.
Package	View
Operations	+ onCreate()

3.1.2. Home Page

Class Name	HomePage
Class Description	Main page for user to browse and get its recommendations.
Package	View
Attributes	- eventListView: EventListView
Operations	+ onCreate()

3.1.3. Survey Page

Class Name	SurveyPage
Class Description	This page is only shown to first time users to find out their interests.
Package	View
Operations	+ onCreate()

3.1.4. Settings Page

Class Name	SettingsPage
Class Description	The page for user to change the application settings.
Package	View
Operations	+ onCreate()

3.1.5. Event Page

Class Name	EventPage
Class Description	The page to display a specific event. User can set up a notification and view detailed information about the event.
Package	View
Operations	+ onCreate()

3.1.6. Event List View

Class Name	EventListView
Class Description	This is not a page but a view that can be placed to the pages to display a number of events.
Package	View
Operations	+ onCreate()

3.1.7. User

Class Name	User
Class Description	Stores the user information.
Package	Model
Attributes	+ id: String + name: String + relevantTagList: List<Tag> + eventHistory: Map<eventID: int, status: intl>
Operations	+ addTag(tag: Tag): boolean

3.1.8. Event

Class Name	Event
Class Description	Stores the event information.
Package	Model
Attributes	+ id: int + api_id: String + name: String + description: String + url: String + organizer_id: String + startdate: Date + enddate: Date + city: String + adress: String + capacity: int + logoURL: String + Is_free: boolean + status: String + tagList: List<Tag>

3.1.9. Tag

Class Name	Tag
Class Description	Stores the tag information.
Package	Model
Attributes	+ id: String + name: String + relevance: int

3.1.10. Client Controller

Class Name	ClientController
Class Description	Handles client side data management
Package	Controller
Attributes	- singletonObject: ClientController - user: User
Operations	+ getInstance(): ClientController + isUser(id: int): boolean + createUser() + changeLanguage(lang: String) + changeLocation(loc: String) + toggleNotification() + followEvent(event: Event, value: boolean) + search(param: Map<String, String>): List<Event> + getRecommendation(List<Tag>, List<int>): List<Event>

3.1.11. Server Handler

Class Name	ServerHandler
Class Description	Uses REST API to interact with server and fetch event data.
Package	Controller
Attributes	+ JSONObject obj
Operations	+ POSTJSONObject(JSONObject: obj): List<Event> + GETJSONObject(): Response

3.2. Application Layer

3.2.1. API Client

Class Name	<<interface>> APIClient
Class Description	Interface of the API services provided to clients
Package	RESTAPI
Operations	+ createWebService(InputStream):Response + verifyWebService(iInputStream): Response + getSearchResults(Map<String,String>):EventListResource + getRecommendation(List<Tag>, List<int>, int) :EventListResource

3.2.2. Event List Resource

Class Name	EventListResource
Class Description	Representation of the EventList resource
Package	RESTAPI
Attributes	- eventList: List<EventResource> - tagList: List<TagResource>
Operations	+ getEvent(Map<String,String>):List<EventResource> + getRecommendation(List<Tag>, int):List<EventResource>

3.2.3. Event Resource

Class Name	EventResource
Class Description	Representation of the Event resource
Package	RESTAPI
Operations	+ getEventID(): int + getTagList(): List<TagResource>

3.2.4. Tag Resource

Class Name	TagResource
Class Description	Representation of the Tag resource
Package	RESTAPI
Operations	+ getTagID(): int

3.2.5. Recommender

Class Name	Recommender
Class Description	Recommends number of events according to their relevance. Search will start optimistic and it will become more pessimistic as the search goes unsuccessful.
Package	Recommender
Operations	+ run() + recommend(List<Tag>, List<int>, int): List<Event>

3.2.6. Event Collector

Class Name	EventCollector
Class Description	Fetches and tags events periodically and updates database.
Package	EventCollector
Attributes	- eventFetcher: EventFetcher - eventUtility: EventUtility
Operations	+ run()

3.2.7. Event Fetcher

Class Name	<<interface>> EventFetcher
Class Description	Communicates with different APIs and collects event data.
Package	EventCollector
Operations	+ fetchEventList(): List<Event>

3.2.8. Event Utility

Class Name	EventUtility
Class Description	Responsible for processing the collected event data.
Package	EventCollector
Attributes	- event: Event
Operations	+ detectLanguage(): String + lemmatize(): List<String> + classify(): List<String> + parser(): Map<String, String>

3.2.9. Tag

Class Name	Tag
Class Description	Stores the tag information
Package	DatabaseHandler
Attributes	+ id: int + tagName: Sting
Operations	+ Tag(int, Node) + Tag(tagName: String)

3.2.10. Event

Class Name	Event
Class Description	Stores the event information
Package	DatabaseHandler
Attributes	+ id: int + api_id: String + name: String + description: String + url: String + organizer_id: String + startdate: Date + enddate: Date + city: String + adress: String + capacity: int + logoURL: String + Is_free: boolean + status: String + tagList: List<Tag>
Operations	+ Event(id: int, node: Node) + Event(api_id: String, name: String, description: String, url: String, organizer_id: String, startdate: Date, enddate: Date, city: String, adress: String, capacity: int, logoURL: String, Is_free: boolean, status: String)

3.2.11. Database Handler

Class Name	DatabaseHandler
Class Description	Handles all database transactions.
Package	DatabaseHandler
Attributes	+ singletonObject: DatabaseHandler + eventList: List<Event> + tagList: List<Tag>
Operations	+ getInstance(): DatabaseHandler + createEvent(Event): boolean + createTag(String): Tag + createEventTagRelation(Event, Tag): boolean + createTagRelation(Tag, Tag): boolean + getEvent(int): Event + getEventList(Tag): List<Event> + getEventList(Map<String, String>): List<Event> + getEventList(Map<String, String>, Tag): List<Event> + getTag(int): Tag + getTagList(Tag, int): List<Tag> + getTagList(Event): List<Tag> + setTagRelationValue(Tag, Tag, int): boolean + isTag(String): boolean + isEvent(Map<String, String>): boolean

3.2.12. Tag List

Class Name	TagList
Class Description	Stores a list of Tag objects
Package	DatabaseHandler
Attributes	+ tagList: List<Tag> + relevanceList: List<int>

3.2.13. Event List

Class Name	EventList
Class Description	Stores a list of Event objects
Package	DatabaseHandler
Attributes	+ eventList: List<Event>

4. References

[1] "Android - 4.4 KitKat", Android, (n.d.). [Online]. Available: https://www.android.com/intl/tr_tr/versions/kit-kat-4-4/. [Accessed:12-02-2017].

[2] "Android - Play", Android, (n.d.). [Online]. Available: https://www.android.com/play/. [Accessed: 12-02-2017].

[3] "The Neo4j Graph Platform – The #1 Platform for Connected Data", Neo4j Graph Database Platform, 2017. [Online]. Available: https://neo4j.com/. [Accessed: 12-02-2017].

[4] "Dashboards | Android Developers", Android, (n.d.). [Online]. Available: https://developer.android.com/about/dashboards/index.html. [Accessed: 12-02-2017].

[5] "IEEE Editorial Style Manual (Online)", ieee, (n.d.). Available: https://www.ieee.org/conferences_events/conferences/publishing/style_references_manual.pdf. [Accessed: 12-02-2017].

[6] "About the Unified Modeling Language Specification Version 2.5.1", Object Management Group, 2018. Available: http://www.omg.org/spec/UML/2.5.1. [Accessed: 12-02-2017].