# Progetto Specifico

Gentile studente,

in qualità di referente del gruppo riceve questa mail per l'assegnamento del progetto di programmazione ad oggetti.

Al seguente indirizzo è disponibile il documento con i requisiti per lo sviluppo del progetto come richiesto:

<https://learn.univpm.it/mod/page/view.php?id=260183>

Di seguito i dettagli relativi al vostro progetto.

<MACRO TEMA>:

TICKETMASTER

<API REFERENCE>:

<https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/#search-events-v2>

<OBIETTIVO>:

Sviluppare un'applicazione Java che consenta all'utente di **analizzare gli eventi che avranno luogo in Europa** cercati attraverso delle **keyword** a scelta dello studente. Nota: se necessario, usare endpoint aggiuntivi o alternativi rispetto a quello suggerito.

<STATS E FILTRI>:

Per ogni stato (nazione), statistiche e filtri su numero minimo/massimo/medio di eventi raggruppati per "**city**" e per **giorno della settimana**.

Esempi di Stats e Filtri sono disponibile al seguente indirizzo: <https://learn.univpm.it/mod/page/view.php?id=260184>

Dettagli circa l'utilizzo delle API sono disponibili in learn.

Ricordiamo che la scadenza per la presentazione del progetto per l'appello di Marzo 2022 è giovedì 17 marzo, ore 23:59:59.

La consegna avviene unicamente online con le seguenti modalità:

- invio email ai docenti e.frontoni@staff.univpm.it e m.paolanti@staff.univpm.it indicando la commit ed il repository github utilizzato

- compilando il modulo online disponibile al seguente indirizzo:

<https://docs.google.com/forms/d/e/1FAIpQLSdmtgZIbpABtr5Ukwd58gS9NMlkYr5Zv-EYt6C6YE22wBuXBQ/viewform?usp=sf_link>

In caso di dubbi si invita a scrivere ai docenti e.frontoni@staff.univpm.it e [m.paolanti@staff.univpm.it](mailto:m.paolanti@staff.univpm.it).

Si ricorda inoltre che il progetto verrà sottoposto anche ad un controllo anti-plagio per verificare l'originalità dello sviluppo.

# Specifiche Progetto d'esame (aggiornato al 04/12/2021)

Le seguenti specifiche sono personalizzate per ogni gruppo con variazione delle parti in rosso che saranno comunicate ai singoli gruppi dopo la richiesta del progetto d'esame.

A partire dalla seguente API e dalla relativa documentazione: <API REFERENCE>

* Sviluppare in JAVA una applicazione che modelli il problema ed il dataset attraverso opportune classi che rispecchiano i casi d’uso e le strutture dati del progetto e implementi le seguenti funzionalità utilizzando al meglio i principi della OOP (incapsulamento, ereditarietà, polimorfismo, classi astratte, interfacce):
  + all’avvio: autenticarsi nell’API, gestendo, ove necessario, i parametri del progetto in un file JSON locale opportunamente strutturato;
  + a connessione completata e dopo aver ricevuto i dati: effettui il parsing dei dati creando delle strutture dati opportune sulla base delle classi di cui sopra (ogni record del data-set corrisponde ad un oggetto di una classe) con i seguenti obiettivi: <OBIETTIVO>
* Su richiesta, mediante API REST GET o POST (per evitare query string complesse) con rotte distinte:
  + restituire i metadati (formato JSON) ovvero elenco degli attributi e dei relativi tipi di dato
  + restituire i dati (formato JSON);
  + restituire statistiche sui dati (formato JSON) con possibilità di effettuare filtri e in particolare: <STATS E FILTRI>
  + gestire problemi durante il processo di importazione (es. numero di attributi errato in una data riga) e nella fase di richiesta (es. si richiedono statistiche per un attributo inesistente oppure i filtri non sono corretti)
* Documentare il codice sviluppato (attraverso commenti esaustivi e l’utilizzo di JavaDoc, da pubblicare su github)
* Creare almeno 2 Unit Test per la verifica della correttezza di uno dei metodi sviluppati e delle relative eccezioni gestite
* Usare github per versionare il codice da consegnare e consegnare la documentazione JavaDoc del progetto.
* Usare il ReadME.md per creare una pagina descrittiva del progetto, inserendo quanto necessario alla comprensione del funzionamento del vostro programma ed alla fase di verifica delle chiamate API realizzate; dovranno essere indicati anche i contributi allo sviluppo del progetto dei singoli membri del gruppo come esempio di suddivisione dei compiti di sviluppo dei vari membri del gruppo.

**Tutte le specifiche del progetto saranno validate in fase di correzione con particolare riferimento a: qualità del progetto e della modellazione, qualità del codice, completezza della soluzione, completezza della documentazione, design dei test, adeguatezza del ReadME.md, adeguato contributo dei singoli membri del gruppo anche in base alla gestione di GIT e della suddivisione dei compiti all’interno del gruppo.**

A scopo di chiarimento del processo sono a vostra disposizione su learn.univpm.it:

- alcuni progetti di qualità dello scorso anno;

- una pagina di guida alle chiamate API da usare nei progetti;

- una serie di esempi su statistiche e filtri.

# Specifiche Progetto d'esame - Esempi di Stats & Filtri (aggiornato al 04/12/2021)

## Esempi di Filtri & Statistiche

## Logical operators overview

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| $not | Negation logical operator | {"field" : {"$not" : val}} |
| $in | Match any value in array | {"field" : {"$in" : [value1, value2, ...]}} |
| $nin | Not match any value in array | {"field" : {"$nin" : [value1, value2, ...]}} |
| $or | Logical operator | {"$or": [{"status": "GOLD"}, {"status": "SILVER"}]} |
| $and | Logical operator | {"$and": [{"status": "GOLD"}, {"sales": 1000}]} |

## Conditional operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| $gt | > | {"salary": {"$gt": 10000}} |
| $gte | >= | {"salary": {"$gte": 10000}} |
| $lt | < | {"salary": {"$lt": 10000}} |
| $lte | <= | {"salary": {"$lte": 10000}} |
| $bt | >= value <= | {"salary": {"$bt": [5000, 7500]}} |

## Esempio

Data-set: <https://www.istat.it/storage/codici-unita-amministrative/Elenco-comuni-italiani.xls>

### GET /metadata/ (ottiene i metadati)

[

 {

  "alias": "CodReg",

  "sourceField": "Codice Regione", "type": "string"

 },

 {

  "alias": "CodUTS",

  "sourceField": "Codice dell'Unità territoriale sovracomunale \n(valida a fini statistici)", "type":  "string"

 },

 {

  "alias": "CodProv",

  "sourceField": "Codice Provincia (Storico)(1)", "type": "string"

 },

 {

  "alias": "Progressivo",

  "sourceField": "Progressivo del Comune (2)", "type": "string"

 },

 {

  "alias": "CodComAlfa",

  "sourceField": "Codice Comune formato alfanumerico", "type": "string"

 },

 {

  "alias": "Den",

  "sourceField": "Denominazione (Italiana e straniera)", "type": "string"

 },

 {

  "alias": "DenITA",

  "sourceField": "Denominazione in italiano", "type": "string"

 },

 {

  "alias": "L2",

  "sourceField": "Denominazione altra lingua", "type": "string"

 },

 {

  "alias": "CodRipGeo",

  "sourceField": "Codice Ripartizione Geografica", "type": "string"

 },

 {

  "alias": "RipGeo",

  "sourceField": "Ripartizione geografica", "type": "string"

 },

 {

  "alias": "DenReg",

  "sourceField": "Denominazione regione", "type": "string"

 },

 {

  "alias": "DenUTS",

  "sourceField": "Denominazione dell'Unità territoriale sovracomunale \n(valida a fini statistici)","type": "string"

 },

 {

  "alias": "Flag",

  "sourceField": "Flag Comune capoluogo di provincia/città metropolitana/libero consorzio", "type": "integer"

 },

 {

  "alias": "SiglaAuto",

  "sourceField": "Sigla automobilistica", "type": "string"

 },

 {

  "alias": "CodComNum",

  "sourceField": "Codice Comune formato numerico", "type": "integer"

 },

 {

  "alias": "CodComNum110",

  "sourceField": "Codice Comune numerico con 110 province (dal 2010 al 2016)", "type": "integer"

 },

 {

  "alias": "CodComNum107",

  "sourceField": "Codice Comune numerico con 107 province (dal 2006 al 2009)", "type": "integer"

 },

 {

  "alias": "CodComNum103",

  "sourceField": "Codice Comune numerico con 103 province (dal 1995 al 2005)", "type": "integer"

 },

 {

  "alias": "CodCatastale",

  "sourceField": "Codice Catastale del comune", "type": "string"

 },

 {

  "alias": "Popo2011",

  "sourceField": "Popolazione legale 2011 (09/10/2011)", "type": "integer"

 },

 {

  "alias": "NUTS1",

  "sourceField": "NUTS1", "type": "string"

 },

 {

  "alias": "NUTS2",

  "sourceField": "NUTS2(3)", "type": "string"

 },

 {

  "alias": "NUTS3",

  "sourceField": "NUTS3", "type": "string"

 }

]

### GET /data (tutti i dati)

### GET /data?filter={"Popo2011": {"$bt": [1000, 10000]}} (solo quelli tra 1000 e 10000)

[

 {

  "CodReg": 11,

  "CodUTS": "042",

  "CodProv": "042",

  "Progressivo": "002",

  "CodComAlfa": "042002",

  "Den": "Ancona",

  "DenITA": "Ancona",

  "CodRipGeo": 3, "RipGeo": "Centro",

  "DenReg": "Marche",

  "DenUTS": "Ancona", "Flag": 1, "SiglaAuto": "AN", "CodComNum": 42002,

  "CodComNum110": 42002,

  "CodComNum107": 42002,

  "CodComNum103": 42002,

  "CodCatastale": "A271", "Popo2011": 100.497, "NUTS1": "ITI",

  "NUTS2": "ITI3",

  "NUTS3": "ITI32"

 },

 {

  "CodReg": 11,

  "CodUTS": "042",

  "CodProv": "042",

  "Progressivo": "003",

  "CodComAlfa": "042003", "Den": "Arcevia",

  "DenITA": "Arcevia",

  "CodRipGeo": 3, "RipGeo": "Centro",

  "DenReg": "Marche",

  "DenUTS": "Ancona", "Flag": 0, "SiglaAuto": "AN", "CodComNum": 42003,

  "CodComNum110": 42003,

  "CodComNum107": 42003,

  "CodComNum103": 42003,

  "CodCatastale": "A366", "Popo2011": 4.914, "NUTS1": "ITI",

  "NUTS2": "ITI3",

  "NUTS3": "ITI32"

 }

….

]

### GET /stats?field=Popo2011

[

 {

  "field": "Popo2011", "avg": 7509,

  "min": 30,

  "max": 2617175,

  "std": 40177,

  "sum": 59433744,

  "count": 7915

 }

]

### GET /stats?field=Popo2011&filter={"Popo2011": {"$bt": [1000, 10000]}}

[

 {

  "field": "Popo2011", "avg": 3610,

  "min": 1001,

  "max": 10000,

  "std": 2300,

  "sum": 17522288,

  "count": 4855

 }

]

# TicketMaster

Giovanni Patriarca, UNIVPM

We registered a user through the TicketMaster registration procedure, available in the TicketMaster website.

Username: giovapatri

Email: [patriarcagiova@gmail.com](mailto:patriarcagiova@gmail.com)

## User credentials in TicketMaster

Username: giovapatri

|  |  |
| --- | --- |
| Consumer Key | 55VBfAWOGW4lK8A28ZUjV7VAYmQYX9pK |
| Consumer Secret | jkMkjHcnHo8p5w2U |
| Key Issued | Fri, 05/27/2022 - 16:16 |
| Key Expires | Never |

# API documentation

We retrieved and studied the API documentation provided by Ticket Master.

In particular, the API specific for our project: Discovery API.

<https://developer.ticketmaster.com/products-and-docs/apis/getting-started/>

# API code from GitHub

We searched for sample Java projects related to the Ticket Master Discovery API.

Specifically, in GitHub.com, we searched for projects with the keyword “TicketMaster”.

<https://github.com/search?q=ticketmaster>

## Repository

We found a Software Development Kit (SDK) implemented in Java for the TicketMaster API.

<https://github.com/ticketmaster-api/sdk-java>

Project for TicketMaster API’s in Java.

Release 0.1.5. Date 2017/03/01.

## Sample code

DiscoveryApi discoveryApi = new DiscoveryApi(sApiKey);

PagedResponse<Events> response =

discoveryApi.searchEvents(new SearchEventsOperation().countryCode(sCountry).pageSize(100));

The usage of the TicketMaster Java SDK is based on two classes:

* DiscoveryApi
* SearchEventOperation

In the first call, we create an object of class **DiscoveryApi**, which is the entry point for the Discovery API library (package).

When the DiscoveryApi object is created, we pass as parameter the string for the ApiKey, which is then used internally for the **authentication** of the user with the TicketMaster Discovery service.

In the second call, the method **searchEvents** is called, taking as parameter an object of class **SearchEventsOperation**. The SearchEventOperation object is instantiated, in our specific case, passing as parameters the string of **“country codes”** and the number of events to be returned per page.

The SearchEventOperation can be instantiated passing other types of parameters, such as keyword, attraction id, venue id …

# IntelliJ Gradle project

## Gradle settings

Gradle is the equivalent of a C makefile, specifying the source code and the external packages in use by the project.

**setting.gradle**

File containing the project settings, in our case only the project name.

**build.gradle**

File containing the dependencies for the project, that is the external packages used by the application, that need to be “linked”

For example, the package to “read and write” JSON files is an external dependency specified as:

implementation 'org.json:json:20220320'

Simple Main class to use the TicketMaster Discovery API in Java.

Classes implemented:

* **Main** (application)
* **RepEvent** (event as Java object)
* **RepStat** (statistical table entry as Java object)

# Java Doc

## Examples from Oracle

<https://www.oracle.com/it/technical-resources/articles/java/javadoc-tool.html#examples>

The documentation introduced following the JavaDoc templates was added for the three classes implemented

* **Main** (application)
* **RepEvent** (event as Java object)
* **RepStat** (statistical table entry as Java object)

For each class, also the documentation for the main methods was introduced.

IntelliJ includes a tool for JavaDoc generation:

Tools >> Generate JavaDoc

JavaDoc generates a set of HTML files, with the hierarchical description of: packages, classes, methods.

# Java Unit Tests

We implemented, using the tools available in the IntelliJ IDE, a test class to implement unit tests directly for the main class of the project **org.example.Main**.

Selecting the class name, using the combination of keys Alt+Enter, the contextual menu allows to create a test class associated with the source class.

The class is implemented as **org.example.MainTest**.

The method implemented in the class MainTest cover 5 use cases for the application:

GET /data

GET /data?filter={"city":"Hamburg"}

GET /data?filter={"weekday":"Friday"}

GET /stats?field="city"

GET /stats?field="weekday"