Senior Python Test

Time to implement: 3-5 hours

Motivation:

Betbright is a sports betting platform. The main functional areas are:

- Manage data about sporting events to allow users to place bets.
- Provide API to receive data from external providers and update our system with the latest data about events in real time.
- Provide access to support team to allow the to see the most recent data for each event and to query data.

Implementation Details:

- Data:
  - Markets are unique per Sport
  - Selections are unique per Market
- Receiving data:
  - For our purposes we can assume this API will be used by a single provider, so no need to keep track of which provider is sending the message.
- Receiving request from the support team:

Retrieve match by id:

request: http://example.com/api/match/994839351740

```
output format:
 {
        "id": 994839351740,
        "url": "http://example.com/api/match/994839351740",
        "name": "Real Madrid vs Barcelona",
        "startTime": "2018-06-20 10:30:00",
        "sport": {
            "id": 221,
            "name": "Football"
        },
        "markets": [
            {
                "id": 385086549360973392,
                "name": "Winner",
                "selections": [
                    {
                        "id": 8243901714083343527,
                        "name": "Real Madrid",
                        "odds": 1.01
                    },
                    {
                        "id": 5737666888266680774,
                        "name": "Barcelona",
                        "odds": 1.01
                    }
                ]
            }
        ]
    }
```

Retrieve football matches ordered by start_time:

request: http://example.com/api/match/football?ordering=startTime

```
output format:
   [
 {
        "id": 994839351740,
        "url": "http://example.com/api/match/994839351740",
        "name": "Real Madrid vs Barcelona",
        "startTime": "2018-06-20 10:30:00"
    },
 {
        "id": 994839351788,
        "url": "http://example.com/api/match/994839351788",
        "name": "Cavaliers vs Lakers",
        "startTime": "2018-01-15 22:00:00"
    }
   ]
```

Retrieve matches filtered by name:

request: http://example.com/api/match/?name=Real%20Madrid%20vs%20Barcelona

```
output format:
   [
 {
        "id": 994839351740,
        "url": "http://example.com/api/match/994839351740",
        "name": "Real Madrid vs Barcelona",
        "startTime": "2018-06-20 10:30:00"
    }
   ]
```

Specification for sports data sent by external providers:

The external providers will send the data in a specific format:

Message Types

message types have a common structure

- **NewEvent**:
  - A complete new sporting event is being created. Once the event is created successfully the only field that will updated is the selection odds.
- **UpdateOdds**:
  - There is an update for the odds field (all the other fields remain unchanged)

NewEvent

```
{
    "id": 8661032861909884224,
    "message_type": "NewEvent",
    "event": {
        "id": 994839351740,
        "name": "Real Madrid vs Barcelona",
        "startTime": "2018-06-20 10:30:00",
        "sport": {
            "id": 221,
            "name": "Football"
        },
        "markets": [
            {
                "id": 385086549360973392,
                "name": "Winner",
                "selections": [
                    {
                        "id": 8243901714083343527,
                        "name": "Real Madrid",
                        "odds": 1.01
                    },
                    {
                        "id": 5737666888266680774,
                        "name": "Barcelona",
                        "odds": 1.01
                    }
                ]
            }
        ]
    }
}
```


UpdateOdds

```
{
    "id": 8661032861909884224,
    "message_type": "UpdateOdds",
    "event": {
        "id": 994839351740,
        "name": "Real Madrid vs Barcelona",
        "startTime": "2018-06-20 10:30:00",
        "sport": {
            "id": 221,
            "name": "Football"
        },
        "markets": [
            {
                "id": 385086549360973392,
                "name": "Winner",
                "selections": [
                    {
                        "id": 8243901714083343527,
                        "name": "Real Madrid",
                        "odds": 10.00
                    },
                    {
                        "id": 5737666888266680774,
                        "name": "Barcelona",
                        "odds": 5.55
                    }
                ]
            }
        ]
    }
}
```


Message Definition:

Each message contains the full data for that event (match)

Data definition:

- id: INTEGER the unique id for a message
- message_type: STRING it defines the what data is going to be created/updated
- event: the full event data

Event definition:

The event represents a match being played

Data definition:

- id: INTEGER the unique id for a event
- name: STRING name for that event
- markets: LIST contains a list of markets (for our purposes it will be always a list containing a single Market)

Market definition:

This contains data related to the market of that match. Markets define the kind of bet a customer can bet on.
For our purposes we use a single market called: Winner, which means that market is about betting on which player/teams

the customer guess it's going to win the match

Data definition:

- id: INTEGER the unique id for a market
- name: STRING the name for that market
- selection: LIST contains a list of selections

Selection definition:

Selections are the players/teams playing a certain match.
For example, a football match for "Barcelona vs Real Madrid" would have 2 selections: "Real Madrid", "Barcelona".
For Golf matches, you may have 3 players, so you also need to handle that.

Data definition:

- id: INTEGER the unique id for a selection
- name: STRING the name that selection
- odds: FLOAT the current odds for that selection

API Requirements:

- It must be a **REST API**
- It must have unit tests
- The code should be documented

Keep in mind:

- You should use Python
- Assume that your API is going to be deployed to a production environment.
- There is no restriction regarding which dependencies you can use.
- It's up to you how you store the data