12/12/20, Jonathan Shabtai

Final Project

Overview

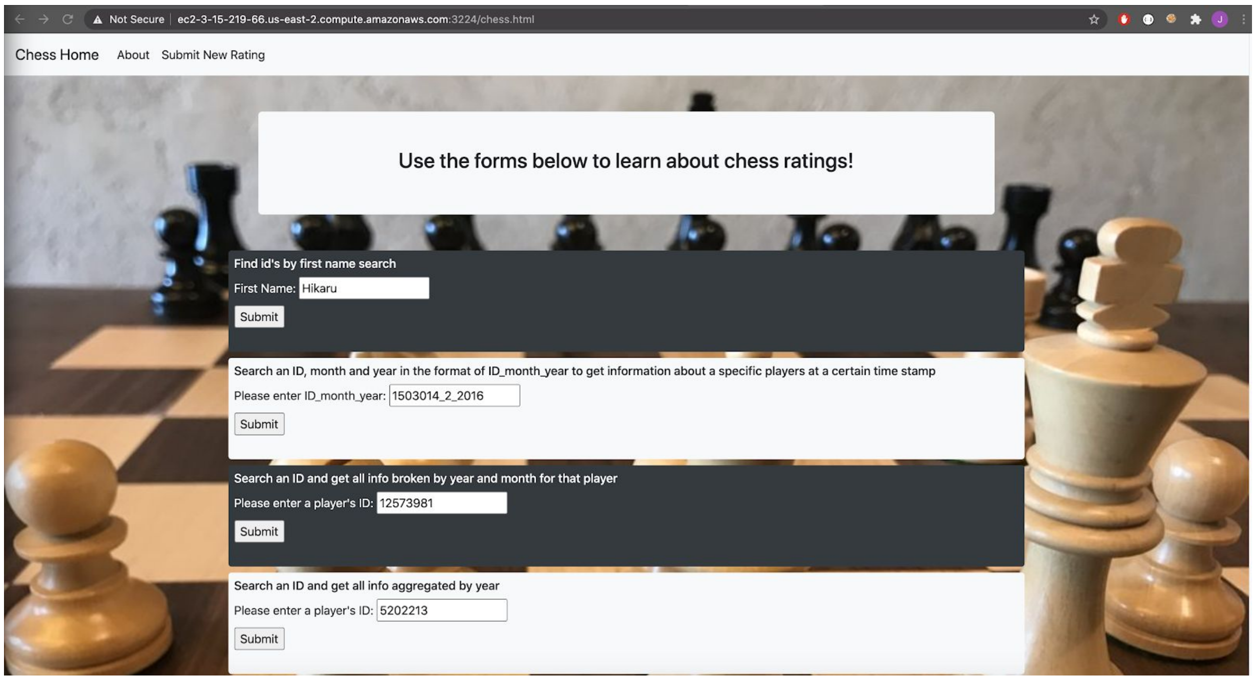
I included a YouTube video that goes over my webapp and its features. It includes a demonstration of my views, queries, and speed layer (running spark-submit).
Please see the following link: https://youtu.be/fVXS7ZGWUzY

My big data app provides a summary of chess ratings for 427,952 players. In this document I describe the tables I use for my master data, batch layer, serving layer, the views available in the webapp, and the speed layer. I used Bootstrap as a .css framework for the front end.

Dataset taken from kaggle:
https://www.kaggle.com/rohanrao/chess-fide-ratings

My port is 3224, I deployed both using QuickDeploy and the LoadBalancer:

QuickDeploy Link: http://ec2-3-15-219-66.us-east-2.compute.amazonaws.com:3224/chess.html
LoadBalancer Link:
http://mpcs53014-loadbalancer-217964685.us-east-2.elb.amazonaws.com:3224/chess.html


Not Secure | ec2-3-15-219-66.us-east-2.compute.amazonaws.com:3224/chess.html
Chess Home
About
Submit New Rating

Use the forms below to learn about chess ratings!

Find id's by first name search
First Name: Hikaru
Submit

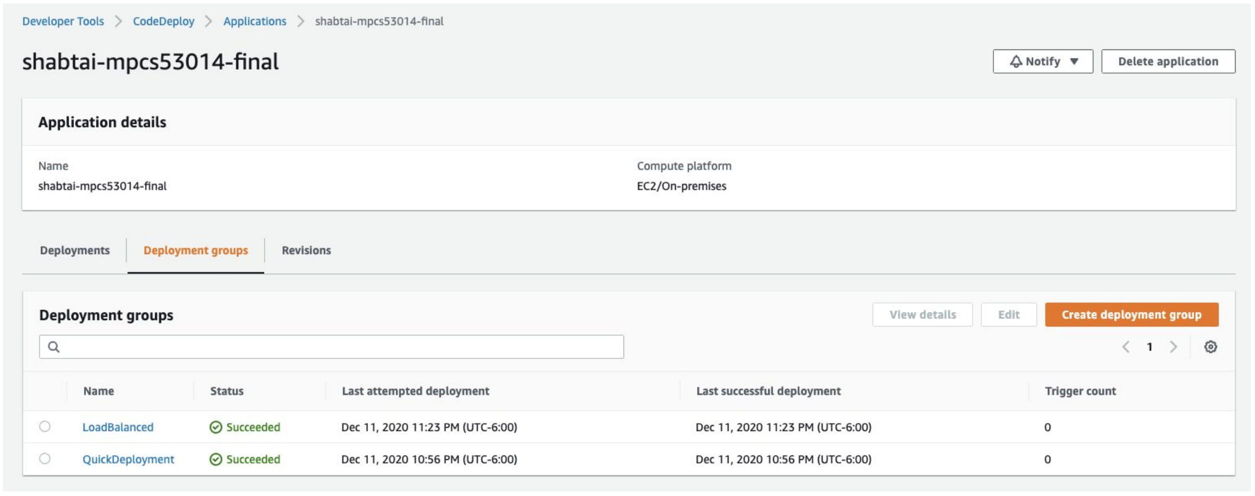
Search an ID, month and year in the format of ID_month_year to get information about a specific players at a certain time stamp
Please enter ID_month_year: 1503014_2_2016
Submit

Search an ID and get all info broken by year and month for that player
Please enter a player's ID: 12573981
Submit

Search an ID and get all info aggregated by year
Please enter a player's ID: 5202213
Submit

In addition to the project zip file which contains the app files (shabtai-final-project.zip, has app.js, all mustache and html files), I attached the scala code for my speed layer, a python script that cleaned my data, and all of the commands I ran in Hive and HBase (these will be available here and also in a commands_used.txt file, it also includes all the commands I found helpful in debugging).

My application name on AWS is ‘shabtai-mpcs53014-final’, and it is running both on QuickDeployment and LoadBalanced:



shabtai-mpcs53014-final

Application details

Name	Compute platform
shabtai-mpcs53014-final	EC2/On-premises

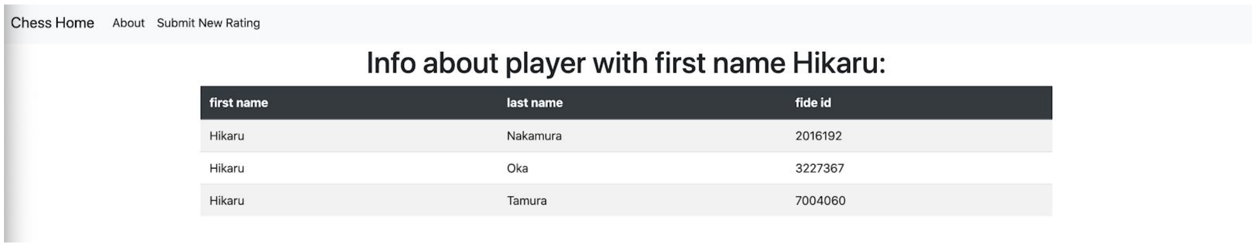
Deployment groups

Name	Status	Last attempted deployment	Last successful deployment	Trigger count
LoadBalanced	Succeeded	Dec 11, 2020 11:23 PM (UTC-6:00)	Dec 11, 2020 11:23 PM (UTC-6:00)	0
QuickDeployment	Succeeded	Dec 11, 2020 10:56 PM (UTC-6:00)	Dec 11, 2020 10:56 PM (UTC-6:00)	0

In the webapp, each submit form has a specialized HBase table behind it - provided as a serving layer per the lambda architecture. Recommended values to plug in are provided below, but you can also follow the format shown as the pre-provided text in the boxes. There is an error catcher coded in the app.js level, as the input for each form has to be in a specific format, this is described below in my serving layer section. On the top of the website you can find a nav-bar that would take you to a Submit New Rating form - this constitutes my speed layer which is also described below.

__Queries__

Running the first query with Hikaru will give:



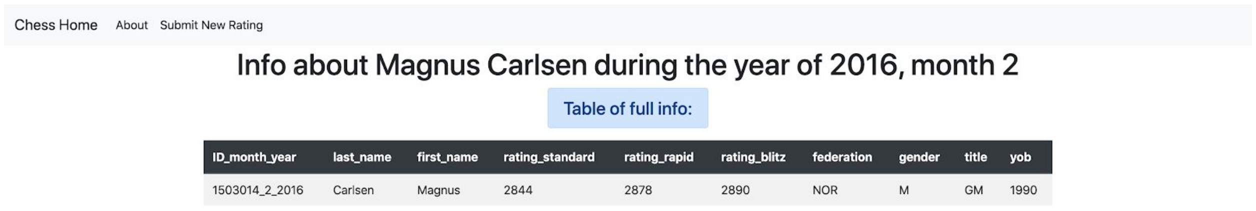
Info about player with first name Hikaru:

first name	last name	fide id
Hikaru	Nakamura	2016192
Hikaru	Oka	3227367
Hikaru	Tamura	7004060

This provides the user a convenient way to find the fide_id of their favorite player, and then use that fide_id for the subsequent forms.

In the next form, the user can input a fide_id_month_year to get a specific datapoint for a player. This will show the user the ratings for that player in a specific timestamp, along with other data provided in the master table.

Running 1503014_2_2016 will give:



Chess Home About Submit New Rating

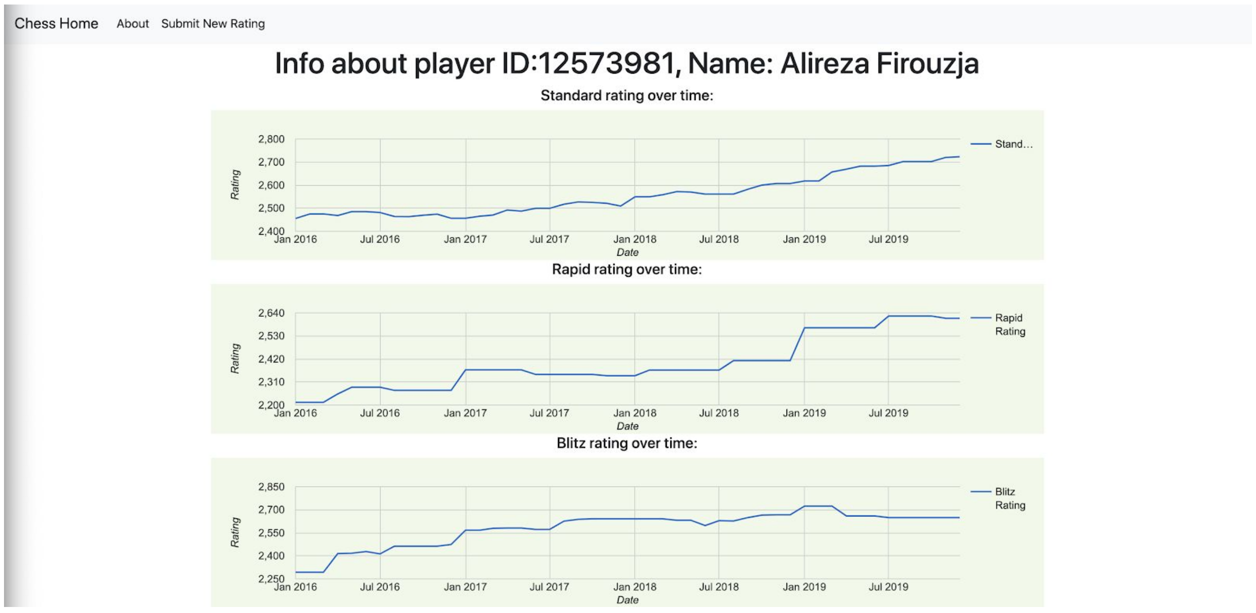
Info about Magnus Carlsen during the year of 2016, month 2

Table of full info:

ID_month_year	last_name	first_name	rating_standard	rating_rapid	rating_blitz	federation	gender	title	yob
1503014_2_2016	Carlsen	Magnus	2844	2878	2890	NOR	M	GM	1990

Next is the form that takes a fide_id as an input, and produces a view that showcases the players progress throughout 2016-2019. This is the most intricate web-page in my app as it includes javascript graphs, as well as information from three different HBase tables.

Running 12573981 gives:



Chess Home About Submit New Rating

Info about player ID:12573981, Name: Alireza Firouzja

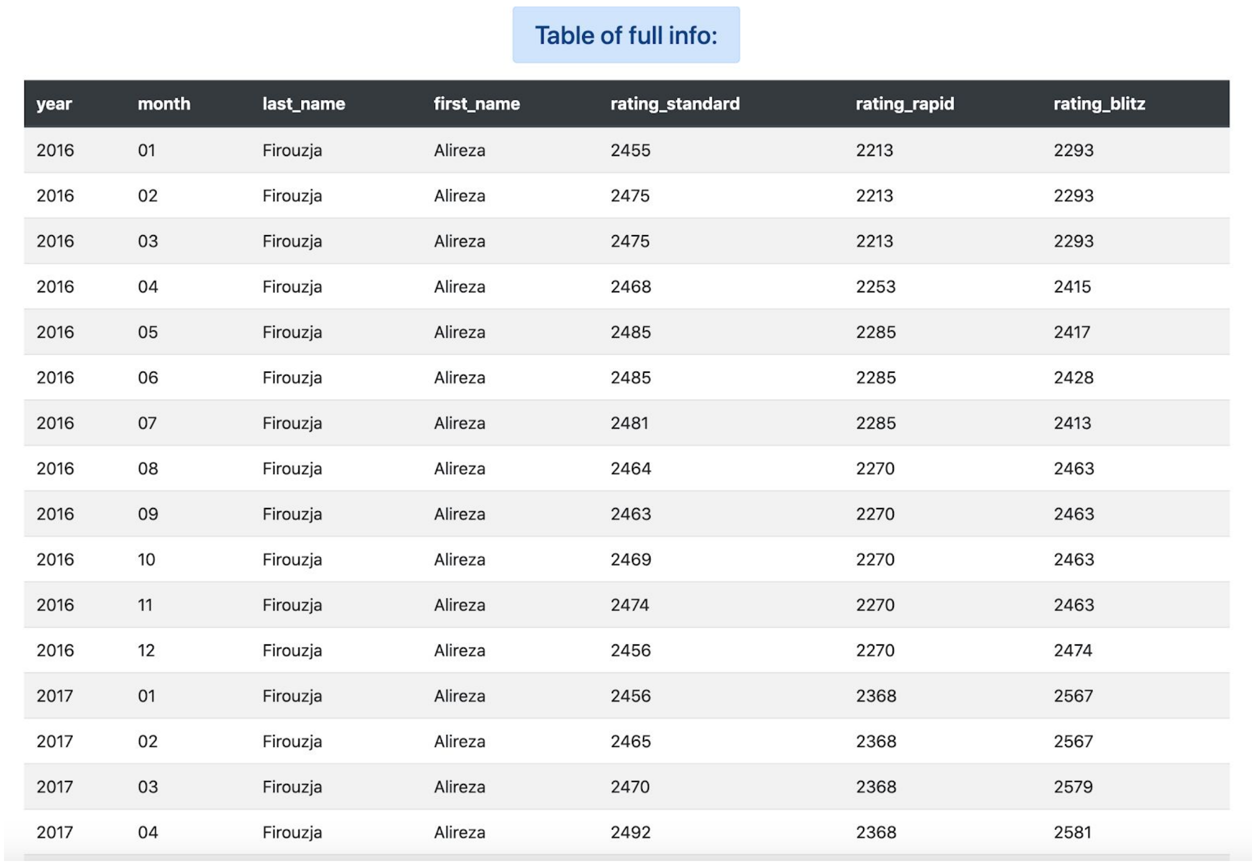
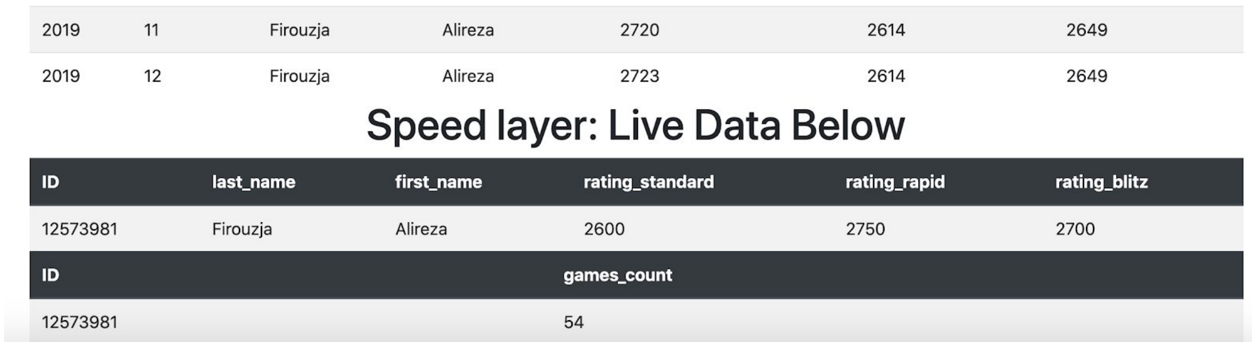


Table of full info:

year	month	last_name	first_name	rating_standard	rating_rapid	rating_blitz
2016	01	Firouzja	Alireza	2455	2213	2293
2016	02	Firouzja	Alireza	2475	2213	2293
2016	03	Firouzja	Alireza	2475	2213	2293
2016	04	Firouzja	Alireza	2468	2253	2415
2016	05	Firouzja	Alireza	2485	2285	2417
2016	06	Firouzja	Alireza	2485	2285	2428
2016	07	Firouzja	Alireza	2481	2285	2413
2016	08	Firouzja	Alireza	2464	2270	2463
2016	09	Firouzja	Alireza	2463	2270	2463
2016	10	Firouzja	Alireza	2469	2270	2463
2016	11	Firouzja	Alireza	2474	2270	2463
2016	12	Firouzja	Alireza	2456	2270	2474
2017	01	Firouzja	Alireza	2456	2368	2567
2017	02	Firouzja	Alireza	2465	2368	2567
2017	03	Firouzja	Alireza	2470	2368	2579
2017	04	Firouzja	Alireza	2492	2368	2581

At the bottom of this view, you can find the players current rating and game count. This info comes from the submit form as live data:



2019	11	Firouzja	Alireza	2720	2614	2649
2019	12	Firouzja	Alireza	2723	2614	2649

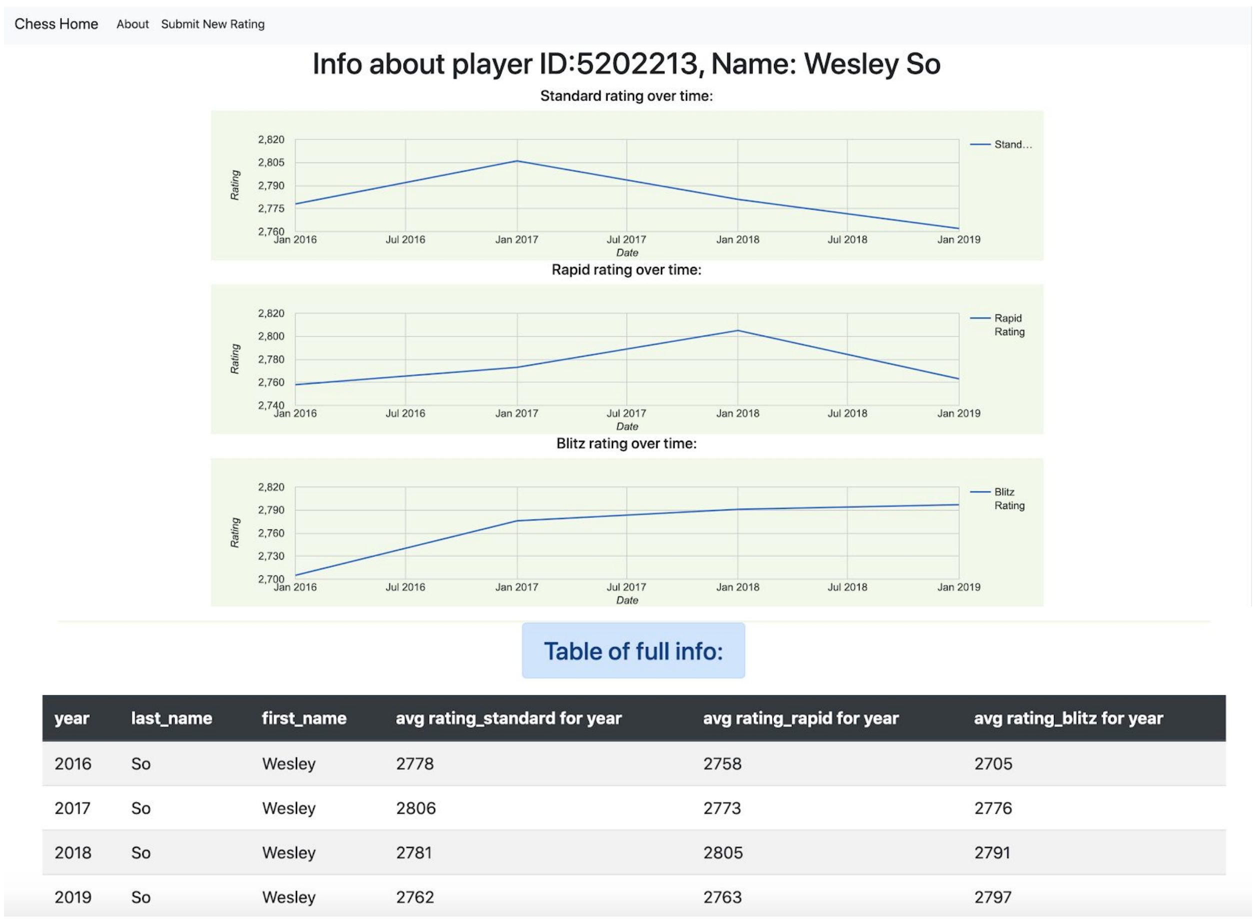
Speed layer: Live Data Below

ID	last_name	first_name	rating_standard	rating_rapid	rating_blitz
12573981	Firouzja	Alireza	2600	2750	2700

ID	games_count
12573981	54

This feature is further explained in the speed layer section. I also show how to use the submit form below in this section.

The next view takes a fide_id as an argument, and produces an annual aggregation table for the player through a specialized HBase table. Running 5202213 gives:



Info about player ID:5202213, Name: Wesley So

Standard rating over time:

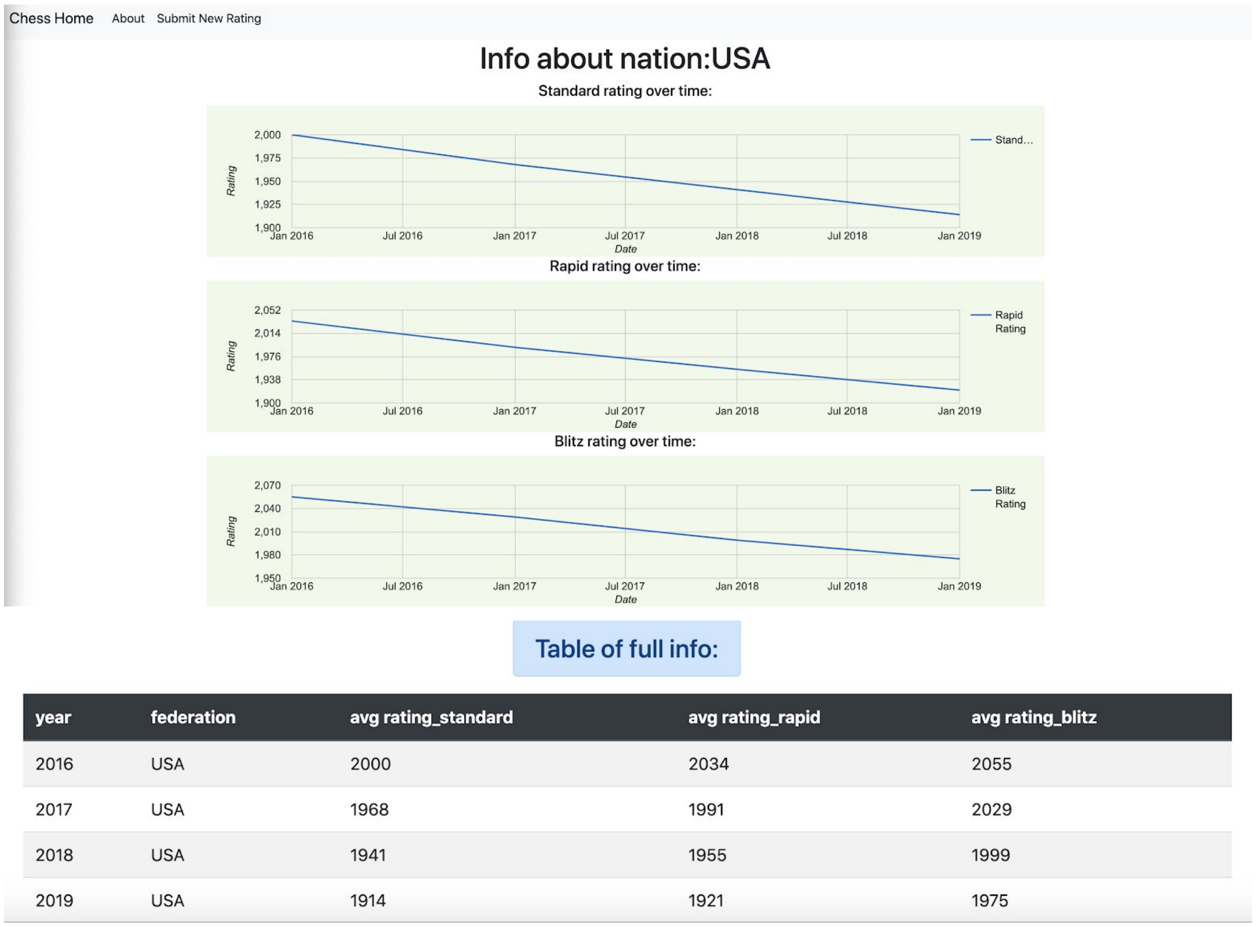
Rapid rating over time:

Blitz rating over time:

Table of full info:

year	last_name	first_name	avg rating_standard for year	avg rating_rapid for year	avg rating_blitz for year
2016	So	Wesley	2778	2758	2705
2017	So	Wesley	2806	2773	2776
2018	So	Wesley	2781	2805	2791
2019	So	Wesley	2762	2763	2797

The next view is similar, but takes as an argument a federation (USA, RUS, ISR, etc…) and produces an aggregated table by year for that federation from a specialized HBase table:

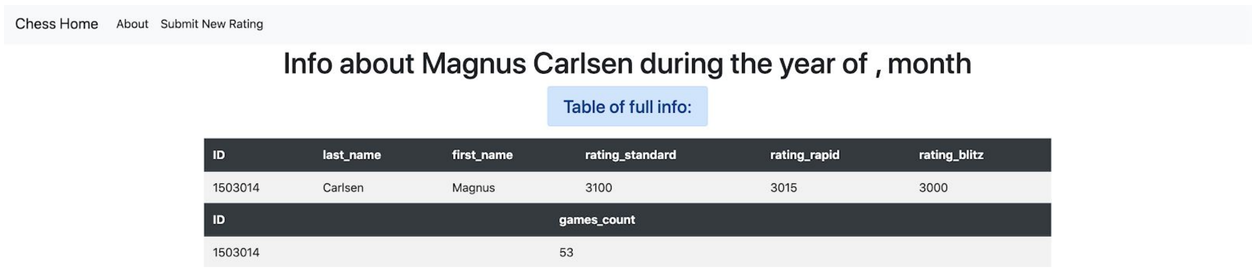


Info about nation:USA

Table of full info:

year	federation	avg rating_standard	avg rating_rapid	avg rating_blitz
2016	USA	2000	2034	2055
2017	USA	1968	1991	2029
2018	USA	1941	1955	1999
2019	USA	1914	1921	1975

Finally, the last submit provides a view for a fide_id latest data (this information is also shown at the bottom of one of the earlier view):



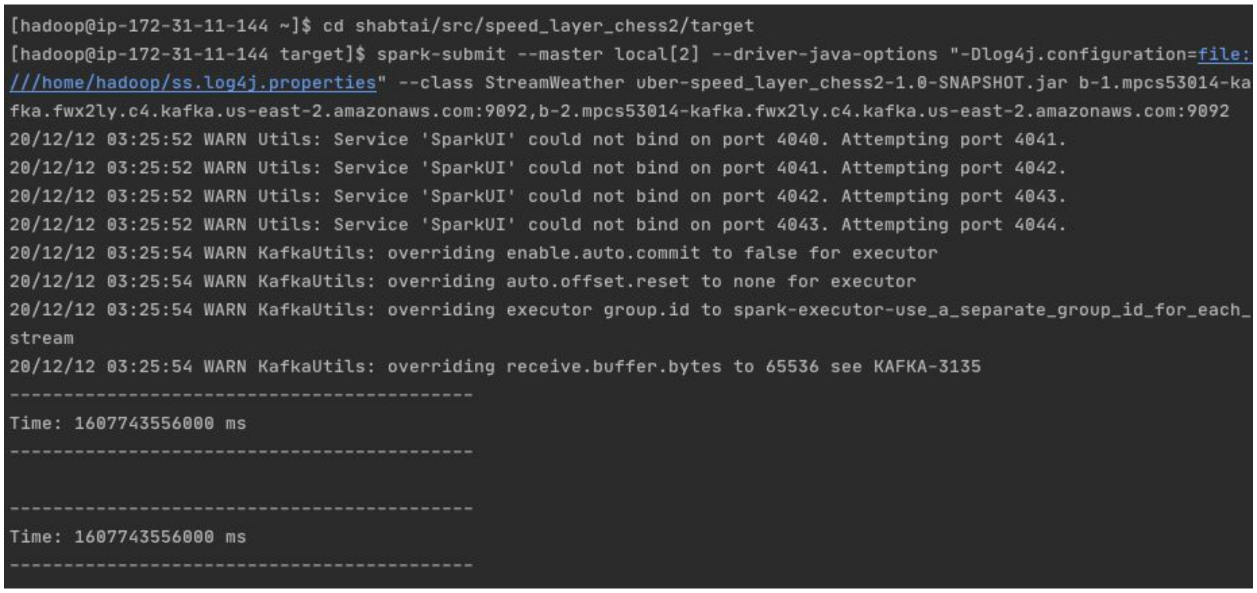
Info about Magnus Carlsen during the year of , month

Table of full info:

ID	last_name	first_name	rating_standard	rating_rapid	rating_blitz
1503014	Carlsen	Magnus	3100	3015	3000

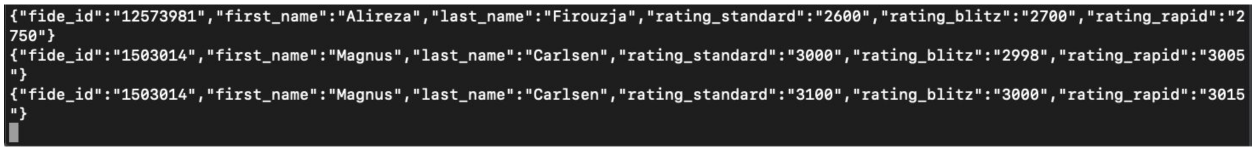
ID	games_count
1503014	53

To have the submit form working, we first run the following spark-submit command under my directory:



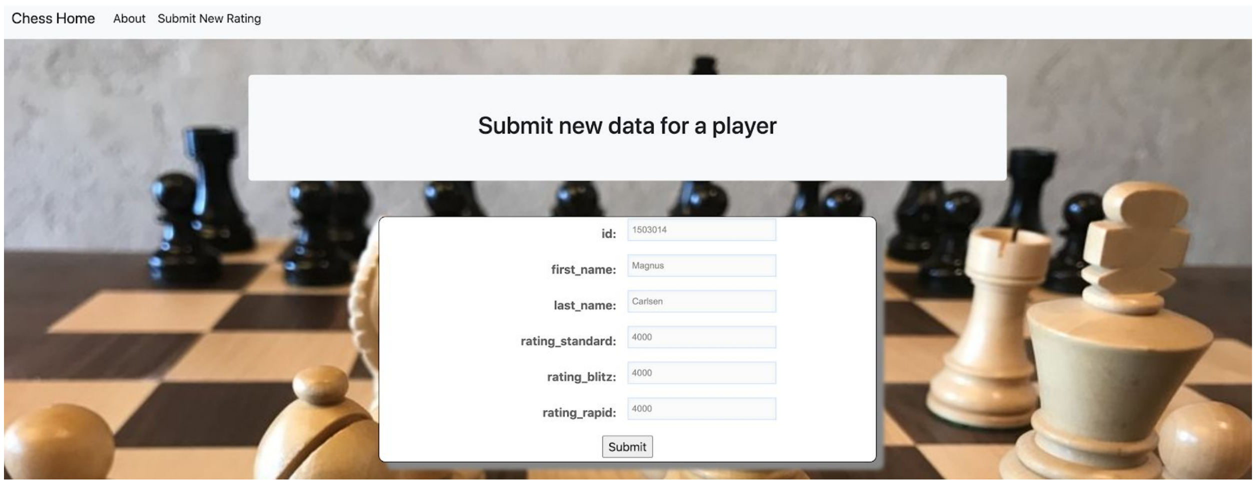
```
[hadoop@ip-172-31-11-144 ~]$ cd shabtai/src/speed_layer_chess2/target
[hadoop@ip-172-31-11-144 target]$ spark-submit --master local[2] --driver-java-options "-Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamWeather uber-speed_layer_chess2-1.0-SNAPSHOT.jar b-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092,b-2.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092
20/12/12 03:25:52 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
20/12/12 03:25:52 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
20/12/12 03:25:52 WARN Utils: Service 'SparkUI' could not bind on port 4042. Attempting port 4043.
20/12/12 03:25:52 WARN Utils: Service 'SparkUI' could not bind on port 4043. Attempting port 4044.
20/12/12 03:25:54 WARN KafkaUtils: overriding enable.auto.commit to false for executor
20/12/12 03:25:54 WARN KafkaUtils: overriding auto.offset.reset to none for executor
20/12/12 03:25:54 WARN KafkaUtils: overriding executor group.id to spark-executor-use_a_separate_group_id_for_each_stream
20/12/12 03:25:54 WARN KafkaUtils: overriding receive.buffer.bytes to 65536 see KAFKA-3135
-------------------------------------------
Time: 1607743556000 ms
-------------------------------------------

-------------------------------------------
Time: 1607743556000 ms
-------------------------------------------
```

And we can also look at my topic (shabtai_final_test):



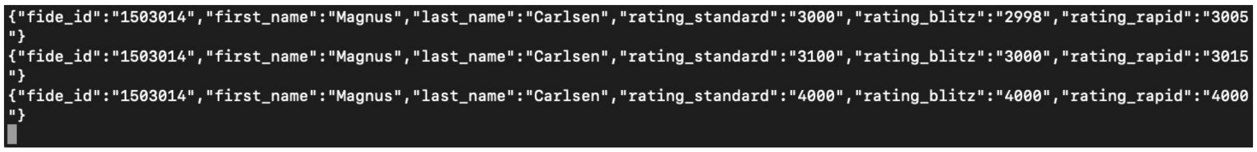
```
{"fide_id":"12573981","first_name":"Alireza","last_name":"Firouzja","rating_standard":"2600","rating_blitz":"2700","rating_rapid":"2750"}
{"fide_id":"1503014","first_name":"Magnus","last_name":"Carlsen","rating_standard":"3000","rating_blitz":"2998","rating_rapid":"3005"}
{"fide_id":"1503014","first_name":"Magnus","last_name":"Carlsen","rating_standard":"3100","rating_blitz":"3000","rating_rapid":"3015"}
```

After writing a submit form as such and running spark-submit, we can see that the topic updates and with it also the view:



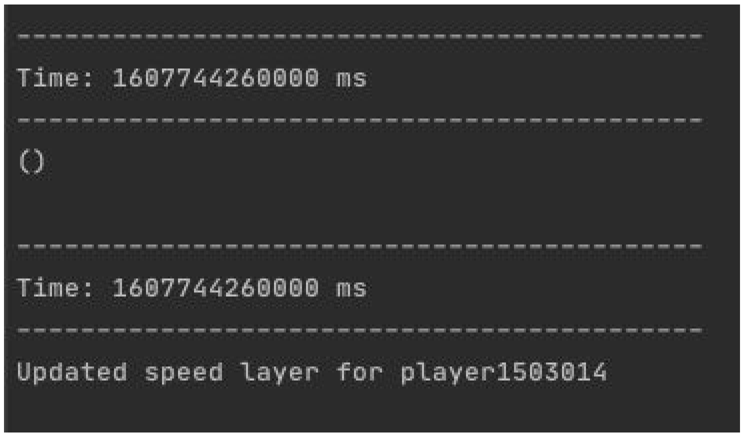
id:	1503014
first_name:	Magnus
last_name:	Carlsen
rating_standard:	4000
rating_blitz:	4000
rating_rapid:	4000

New Kafka message appears once we click submit:



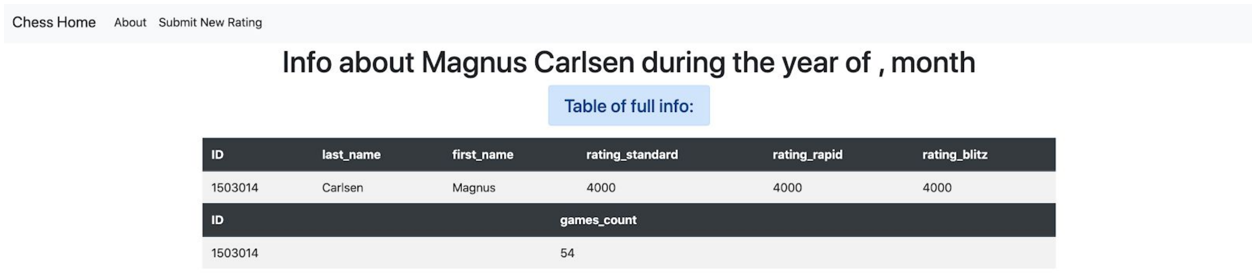
```
{"fide_id":"1503014","first_name":"Magnus","last_name":"Carlsen","rating_standard":"3000","rating_blitz":"2998","rating_rapid":"3005
"}
{"fide_id":"1503014","first_name":"Magnus","last_name":"Carlsen","rating_standard":"3100","rating_blitz":"3000","rating_rapid":"3015
"}
{"fide_id":"1503014","first_name":"Magnus","last_name":"Carlsen","rating_standard":"4000","rating_blitz":"4000","rating_rapid":"4000
"}
```

And the .scala script also shows an update:



```
-------------------------------------------
Time: 1607744260000 ms
-------------------------------------------
()

-------------------------------------------
Time: 1607744260000 ms
-------------------------------------------
Updated speed layer for player1503014
```

We confirm that the data was updated for Magnus:



Info about Magnus Carlsen during the year of , month

Table of full info:

ID	last_name	first_name	rating_standard	rating_rapid	rating_blitz
1503014	Carlsen	Magnus	4000	4000	4000

ID	games_count
1503014	54

We can note that the ratings are updated now, and the games_count has increased by 1.

Getting Data on Hive (Batch Layer)

From Kaggle, the data came as players.csv and rating<year>.csv. I combined the ratings from 2016 to 2019 locally and ran a python script to clean the players.csv file - getting rid of quotation marks and spaces.

I uploaded the data to our cluster from my local machine using the following code:

scp -i shabtai.pem ./final/shabtai_ratings_2016_2019.csv
hadoop@ec2-52-15-169-10.us-east-2.compute.amazonaws.com:/home/hadoop/shabtai/final
scp -i shabtai.pem ./final/players_new.csv
hadoop@ec2-52-15-169-10.us-east-2.compute.amazonaws.com:/home/hadoop/shabtai/final

The columns for the dataset are:
players table:
fide_id,name,federation,gender,title,yob
ratings table:
fide_id,year,month,rating_standard,rating_rapid,rating_blitz

My batch layer (and master data) is built in Hive and is a join of the players table and the shabtai_ratings_2016_2019 table on fide_id (fide_id is a unique primary key in the database for each player).

Commands I used to create the tables and join them:

In Hive:

```
CREATE TABLE shabtai_ratings_2016_2019
  (
    fide_id STRING,
    year BIGINT,
    month BIGINT,
    rating_standard BIGINT,
    rating_rapid BIGINT,
    rating_blitz BIGINT
 )
row format delimited fields terminated BY ',' lines terminated BY '\n' tblproperties("skip.header.line.count"="1");

load data local inpath '/home/hadoop/shabtai/final/shabtai_ratings_2016_2019.csv' into table
shabtai_ratings_2016_2019;

CREATE TABLE shabtai_players
  (
    fide_id STRING,
    last_name STRING,
    first_name STRING,
    federation STRING,
    gender STRING,
    title STRING,
    yob BIGINT
 )
row format delimited fields terminated BY ',' lines terminated BY '\n' tblproperties("skip.header.line.count"="1");

load data local inpath '/home/hadoop/shabtai/final/players_new.csv' into table shabtai_players;
```

Then I joined the tables in Hive:

```
create table shabtai_players_and_ratings as(
SELECT shabtai_players.fide_id,shabtai_players.last_name,
shabtai_players.first_name,shabtai_players.federation,shabtai_players.gender,shabtai_players.title,shabtai_players.y
ob,
shabtai_ratings_2016_2019.year,shabtai_ratings_2016_2019.month,
shabtai_ratings_2016_2019.rating_standard,shabtai_ratings_2016_2019.rating_rapid,
shabtai_ratings_2016_2019.rating_blitz
FROM shabtai_players JOIN shabtai_ratings_2016_2019
ON (shabtai_players.fide_id = shabtai_ratings_2016_2019.fide_id));
```

This provides me with the notion of batch layer and data source (master data).

When running describe shabtai_players_and_ratings we see:

```
+------------------+------------+----------+
|     col_name     | data_type  | comment  |
+------------------+------------+----------+
| fide_id          | string     |          |
| last_name        | string     |          |
| first_name       | string     |          |
| federation       | string     |          |
| gender           | string     |          |
| title            | string     |          |
| yob              | bigint     |          |
| year             | bigint     |          |
| month            | bigint     |          |
| rating_standard  | bigint     |          |
| rating_rapid     | bigint     |          |
| rating_blitz     | bigint     |          |
+------------------+------------+----------+
```

Then I inserted this table into an HBase table as I use it as one of my views in the serving layer. Note that the key values are fide_id_month_year. I use a similar remove prefix function presented in the flight_and_weather apps in order to parse my table values.

```
//In hbase:
create 'shabtai_players_and_ratings_hbase', 'stats'

//In hive:
create external table shabtai_players_and_ratings_hbase (
   fide_id_month_year STRING,
   last_name STRING,
   first_name STRING,
   federation STRING,
   gender STRING,
   title STRING,
   yob STRING,
    year BIGINT,
    month BIGINT,
    rating_standard BIGINT,
    rating_rapid BIGINT,
    rating_blitz BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES
('hbase.columns.mapping' = ':key,
stats:last_name,
stats:first_name,
stats:federation,
stats:gender,
stats:title,
stats:yob,
stats:year,
stats:month,
stats:rating_standard,
stats:rating_rapid,
stats:rating_blitz')
TBLPROPERTIES ('hbase.table.name' = 'shabtai_players_and_ratings_hbase');

insert overwrite table shabtai_players_and_ratings_hbase
select concat(fide_id, '_', year, '_', month), last_name, first_name, federation, gender, title, yob,
year, month, rating_standard, rating_rapid, rating_blitz from shabtai_players_and_ratings;
```

HBase Tables (Serving Layer)

My HBase tables represent precomputed views for queries that are highlighted in the webapp. Each HBase table is specialized to a certain query in order to have the webapp run quickly and efficiently.

My HBase tables are as follows:

- shabtai_id_name_hbase - provides lookup based on first name (key is first_name_last_name_fide_id). *Use just first_name to query on the webapp.*
  For example, run Mike and see all the Mikes in the database.

- shabtai_players_and_ratings_hbase - described in the previous section.
  In the second query on the site, provide fide_id_month_year.
  For example, run 2016192_3_2019 to see Hikaru's rating in March 2019.

  The third query uses the same HBase table, but queries on just fide id. Run 1503014 to see how Magnus Carlsen is doing.

- shabtai_players_and_ratings_by_year_hbase - aggregation by year (key is just fide_id). Specialized HBase table to query on annual average performance. Use just fide_id to query on the webapp.
  For example, Run 623539 to see Maxime Vachier-Lagrave's performance.

- shabtai_fed_and_ratings_by_year_hbase - aggregation by federation (key is just fed). Specialized HBase table to query on annual average performance of a federation. *Use just federation to query on the webapp.*

Code related to creating these:

```
// search for a player's name and find some id's for that name
//In hbase:
create 'shabtai_id_name_hbase', 'stats'

//In hive:
create external table shabtai_id_name_hbase (
    first_name_last_name_fide_id STRING,
    first_name STRING,
    last_name STRING,
    fide_id STRING)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES
('hbase.columns.mapping' = ':key,
stats:first_name,
stats:last_name,
stats:fide_id')
TBLPROPERTIES ('hbase.table.name' = 'shabtai_id_name_hbase');

insert overwrite table shabtai_id_name_hbase
select concat(first_name, '_', last_name, '_', fide_id), first_name, last_name, fide_id from shabtai_players;

// year agg
```

```
//In hbase:
create 'shabtai_players_and_ratings_by_year_hbase', 'stats'

//In hive:
create external table shabtai_players_and_ratings_by_year_hbase (
    fide_id_month_year STRING,
    last_name STRING,
    first_name STRING,
    federation STRING,
    gender STRING,
    title STRING,
    yob STRING,
    year BIGINT,
    rating_standard BIGINT,
    rating_rapid BIGINT,
    rating_blitz BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES
('hbase.columns.mapping' = ':key,
stats:last_name,
stats:first_name,
stats:federation,
stats:gender,
stats:title,
stats:yob,
stats:year,
stats:rating_standard,
stats:rating_rapid,
stats:rating_blitz')
TBLPROPERTIES ('hbase.table.name' = 'shabtai_players_and_ratings_by_year_hbase');

insert overwrite table shabtai_players_and_ratings_by_year_hbase
select concat(fide_id, '_', year), last_name, first_name, federation, gender, title, yob,
year, AVG(rating_standard), AVG(rating_rapid), AVG(rating_blitz) from shabtai_players_and_ratings GROUP BY
concat(fide_id, '_', year), last_name, first_name, federation, gender, title, yob, year;


// fed agg
//In hbase:
create 'shabtai_fed_and_ratings_by_year_hbase', 'stats'

//In hive:
create external table shabtai_fed_and_ratings_by_year_hbase (
    fed_year STRING,
    federation STRING,
    year BIGINT,
    rating_standard BIGINT,
    rating_rapid BIGINT,
    rating_blitz BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES
('hbase.columns.mapping' = ':key,
stats:federation,
stats:year,
stats:rating_standard,
stats:rating_rapid,
stats:rating_blitz')
TBLPROPERTIES ('hbase.table.name' = 'shabtai_fed_and_ratings_by_year_hbase');

insert overwrite table shabtai_fed_and_ratings_by_year_hbase
select concat(federation, '_', year), federation, year, AVG(rating_standard), AVG(rating_rapid), AVG(rating_blitz) from
shabtai_players_and_ratings GROUP BY concat(federation, '_', year), federation, year;
```

Speed Layer

The main HBase table I use for my speed layer is 'latest_chess_results'. It contains data that represent live ratings for players. The webapp user can navigate to a submission form though the nav bar. The data imputed in the form is written into my Kafka topic (shabtai_final_test). From the Kafka topic, a .scala file populates the 'latest_chess_results' HBase table. In addition, I created an HBase table that keeps count of each player's games ('shabtai_id_count'). This table will be incremented for a player if a form with their fide_id was submitted.

My HBase tables are as follows:

- shabtai_id_count - shows games_count for each fide_id (key is fide_id), this table is an incremental one. The count increases by 1 for a specific fide_id when a submit form is processed. This is a good example of an HBase table that updates as new data points are recorded.
- latest_chess_results - shows the latest chess ratings for a specific player (key is fide_id). This HBase table reads from my topic (shabtai_final_test), and updates through the spark-submit job mentioned below.

Code related to creating the HBase tables:

```
In hbase:
create 'latest_chess_results', 'stats'
// Number of games for each player
In hbase:
create 'shabtai_id_count', 'stats'

In hive:
create external table shabtai_id_count (
    fide_id STRING,
    games_count BIGINT
    )
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES
('hbase.columns.mapping' = ':key,
stats:games_count#b')
TBLPROPERTIES ('hbase.table.name' = 'shabtai_id_count');

insert overwrite table shabtai_id_count
select fide_id, count(*) from shabtai_players_and_ratings GROUP BY fide_id;
```

Data Streaming

The .scala code I used is attached. I have built it off of the StreamWeather.scala presented in class.

My Kafka topic is 'shabtai_final_test'
Code I used to create it:

```
// speed layer work
//in hadoop:
cd kafka_2.12-2.2.1/bin/

//to create my topic:
./kafka-topics.sh --create --zookeeper
z-2.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:2181,z-3.mpcs53014-kafka.fwx2ly.c4.kafka.us-east
-2.amazonaws.com:2181,z-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:2181 --replication-factor 2
--partitions 1 --topic shabtai_final_test

//make sure the topic is ready to go:
./kafka-topics.sh --list --zookeeper
z-2.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:2181,z-3.mpcs53014-kafka.fwx2ly.c4.kafka.us-east
-2.amazonaws.com:2181,z-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:2181

//and read what is in it:
./kafka-console-consumer.sh --bootstrap-server
b-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092,b-2.mpcs53014-kafka.fwx2ly.c4.kafka.us-east
-2.amazonaws.com:9092 --topic shabtai_final_test --from-beginning
```

My .jar file is located on the cluster at:
/home/hadoop/shabtai/src/speed_layer_chess2/target
It is saved as: *uber-speed_layer_chess2-1.0-SNAPSHOT.jar*

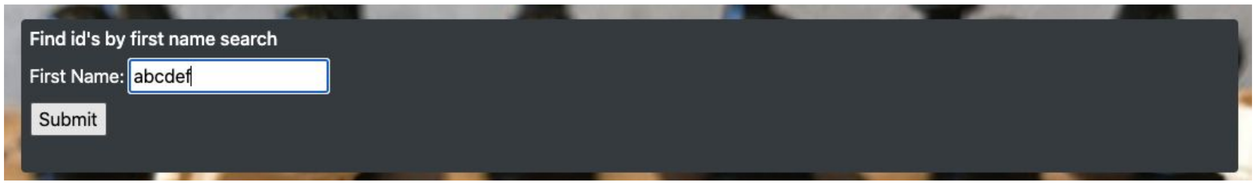
To run it I cd into the target directory and run (this is shown in my tech demo video):

```
cd shabtai/src/speed_layer_chess2/target

spark-submit --master local[2] --driver-java-options
"-Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamWeather
uber-speed_layer_chess2-1.0-SNAPSHOT.jar
b-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092,b-2.mpcs53014-kafka.fw
x2ly.c4.kafka.us-east-2.amazonaws.com:9092
```

<u>Error catcher</u>

This is just a feature I included through my app.js that gives an output for values that are not present as keys or key-prefixes in the HBase tables.

For example running a name that doesn't:

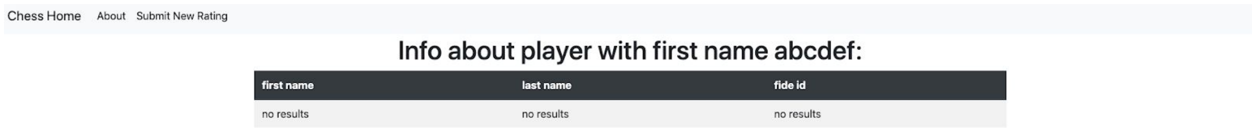


Find id's by first name search

First Name: abcdef

Submit

Will give:

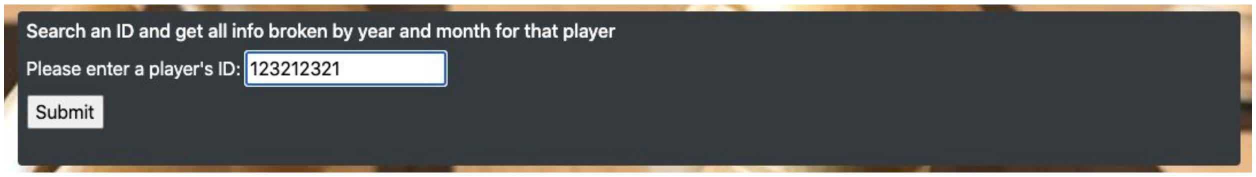


Chess Home About Submit New Rating

Info about player with first name abcdef:

first name	last name	fide id
no results	no results	no results

And running a fide_id that doesn't exist for the third query:

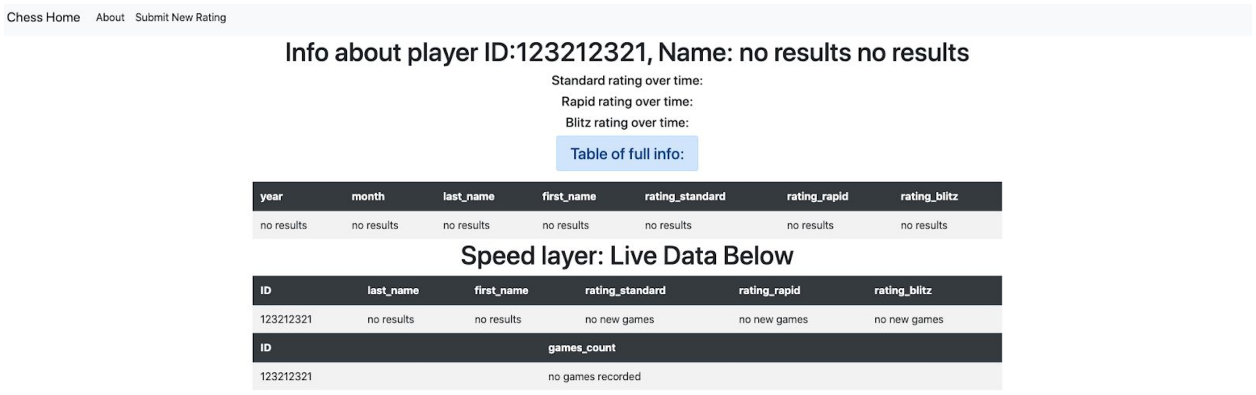


Search an ID and get all info broken by year and month for that player

Please enter a player's ID: 123212321

Submit

Will give:



Chess Home About Submit New Rating

Info about player ID:123212321, Name: no results no results

Standard rating over time:

Rapid rating over time:

Blitz rating over time:

Table of full info:

year	month	last_name	first_name	rating_standard	rating_rapid	rating_blitz
no results	no results	no results	no results	no results	no results	no results

Speed layer: Live Data Below

ID	last_name	first_name	rating_standard	rating_rapid	rating_blitz
123212321	no results	no results	no new games	no new games	no new games

ID	games_count
123212321	no games recorded

The same works for the rest of my queries.