OVIA Incentive Demo

What You Need

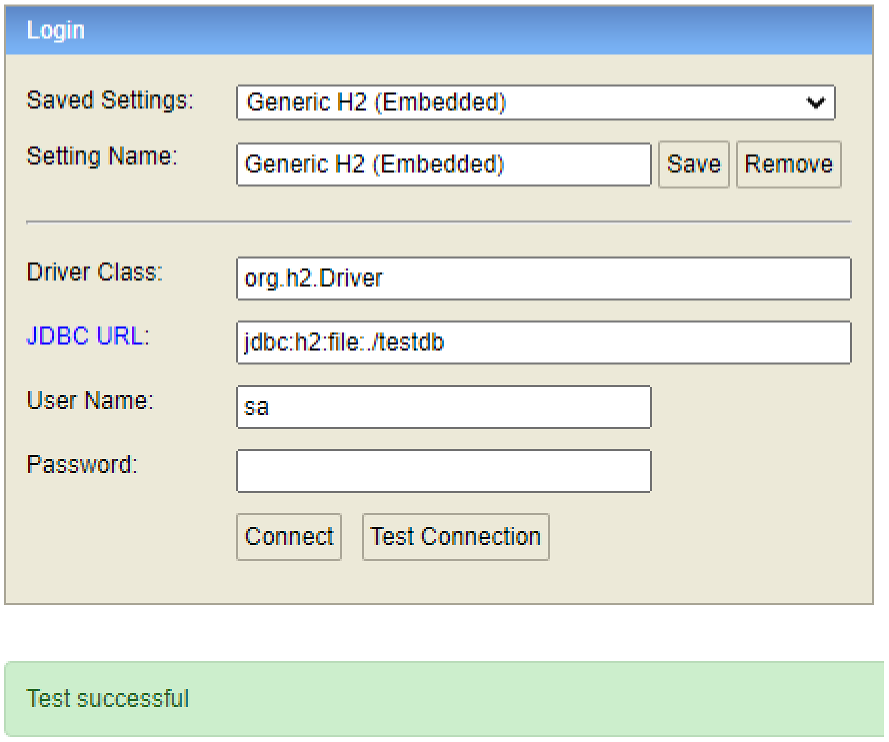
- JDK 1.8 or later
- Maven 3.2+
- Favorite IDE

Download

- git clone https://github.com/t2r/ovia.git

How to run the demo

- cd ovia-incentive-demo
- mvn clean install
- mvn spring-boot:run (server)
- ./client-test.sh (client requests to trigger awards)
    - This is just using *curl* to spend get/post requests to the REST API endpoints.
- You may also query the database tables directly by hitting http://localhost:9000/h2



Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:file:./testdb

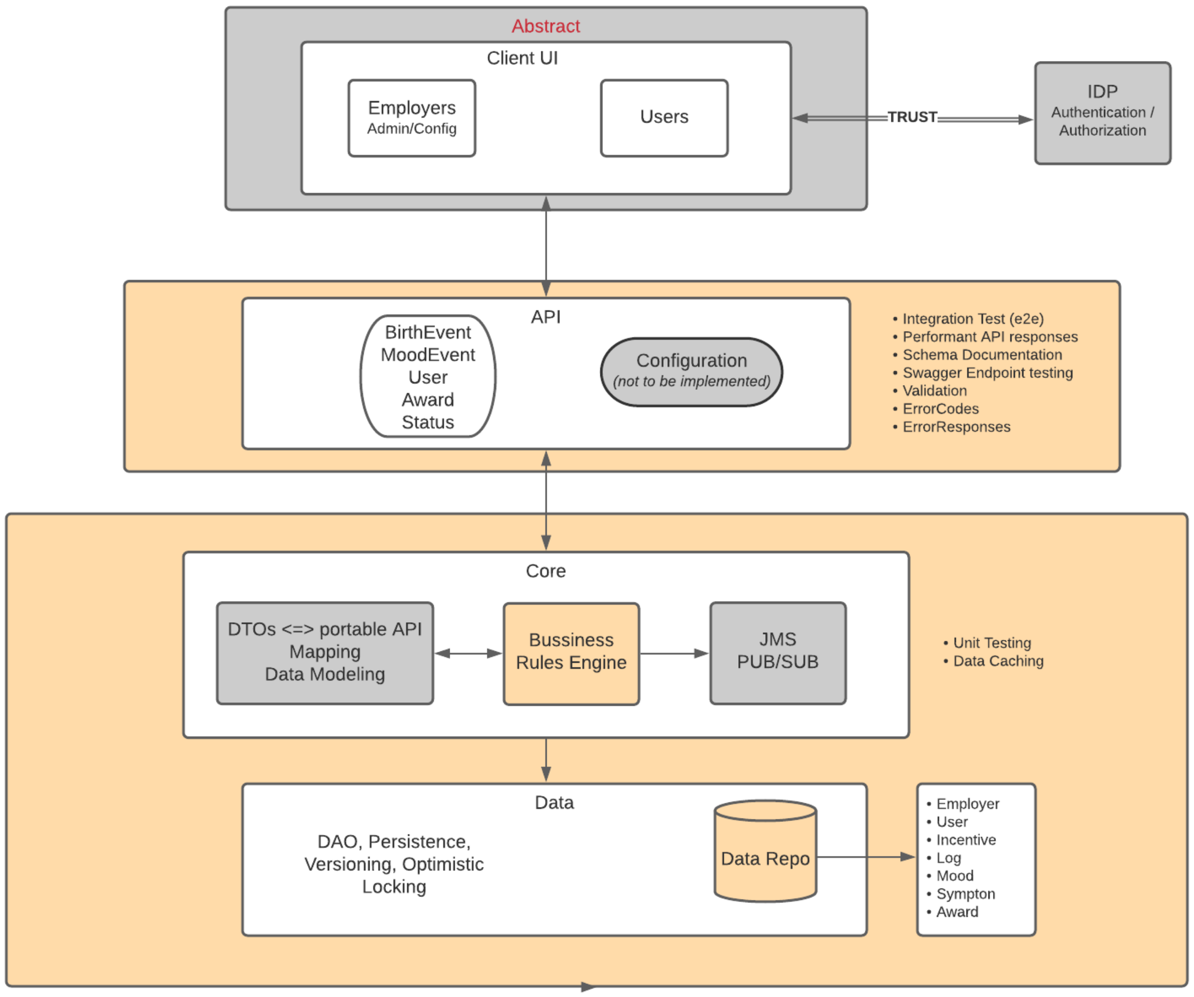
User Name: sa

Password:

Test successful

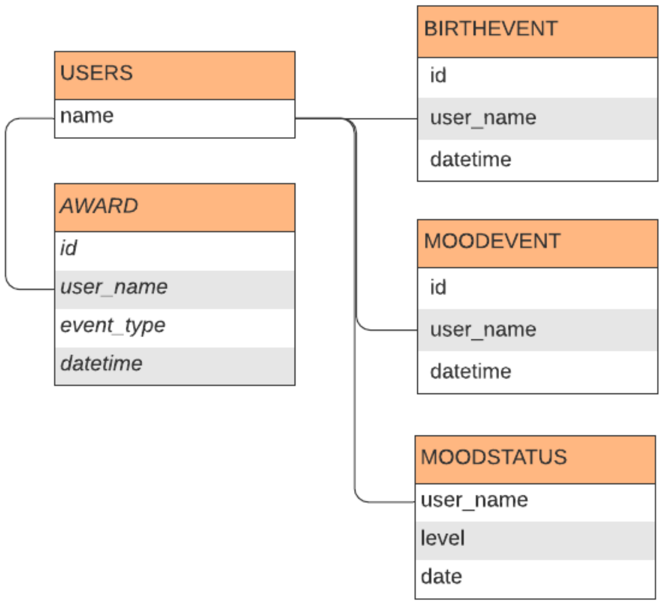
Design Overview

- I first started with an overview software component architecture design to determine how I was going to break this task down into a manageable demo and to help identify the components/frameworks I would need to accomplish this task as well as identify areas that that would not be able to focus on.



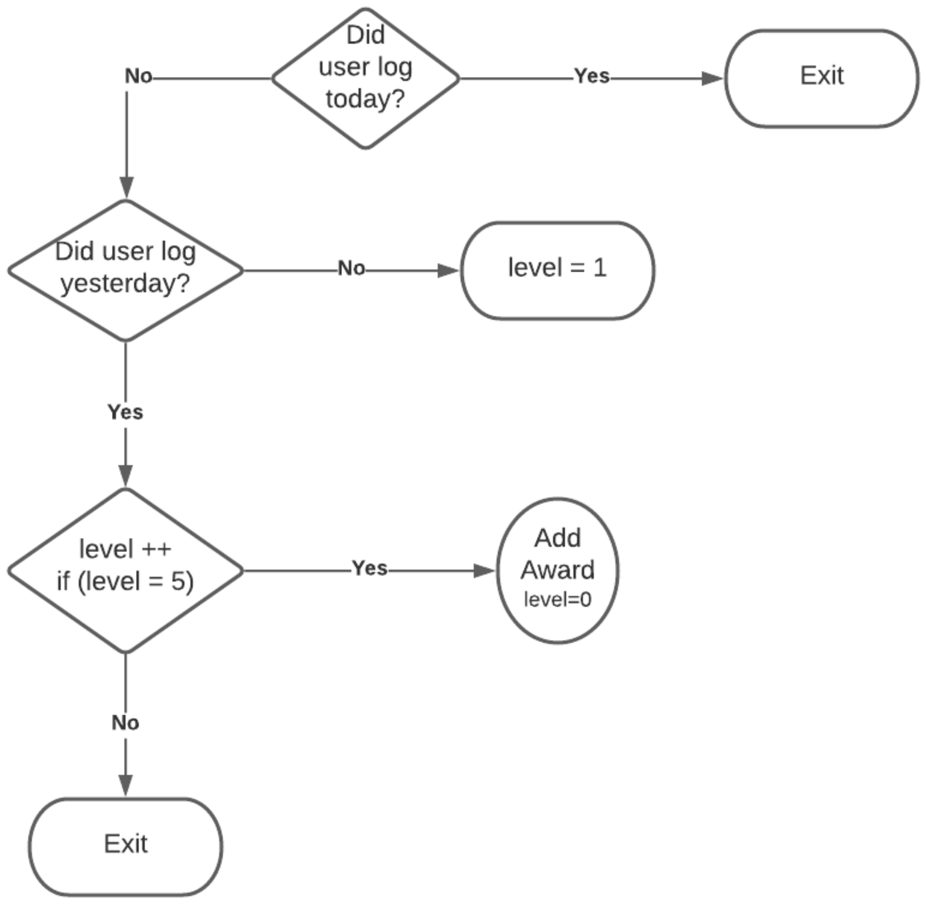
1. I knew I would not get to get documentation, testing, mapping from DTO to portable JSON API objects, flexible dynamic configuration, a *real* BRE to trigger events, etc. Of course, BRE, is one of the most significant performance improvements that could be considered here. The demo is simplistic in that it is single-threaded and executes all logic before returning response back to client. My thoughts immediately shifted to concurrency, data storage size where perhaps a background scanner that is scheduled to run off-peak hours/etc to collect statistics and determine awards without blocking the API response/client and send push notifications instead would be far much better approach.

- My next step, since I had no real system integration requirements, was to determine the data layer needed to help conceptualize the api layer that would be needed.



- I then knew I had two algorithm requirements to meet:
    1. Reported BIRTH event, which is just a direct relationship with earning an award.

2. User logged data five days in a row. For this, I created a state transition diagram to help me decide how I would accomplish this code wise:



- I knew that that algorithm implementation would be extremely rigid meaning that I wasn't going to be able dive into a flexible, dynamic configuration based on customer configuration/requirements that would drive the behavior for how this functions.

Improvements

- Documentation - code, API schema
- Unit Testing
    - Something like swagger to allow for interactive API testing and viewing documentation.
    - End-to-End integration tests that test the entire work-flow.
    - Additional unit tests that provide more code coverage with min, max, edge-case, and bad data scenarios.
- A BRE system that allows event triggering/scheduling to decouple heavy algo processing from user API response time.
    - Something like a PUB/SUB model came to mind where notifications could be pushed.
- Concurrency, optimistic locking, versioning

- Configuration, business rules, business policy were surrounding impacts that did not contain enough requirements to properly address.

Data Sharing

- This needs a lot more surrounding requirements to give it proper attention. But here are my initial thoughts:
    - IDP and trust
    - Reporting Subsystem. Rather than data access, are the customer's simply look for metrics and data reporting charts/views that can viewed and/or exported as PDF's, for example.