Technical report

Travel Buddy

Problem presentation

As a tourist abroad or foreigner, interaction with the people around you might be difficult because of social barriers or cultural differences. Especially when travelling alone, some people may experience a certain anxiety and fear of the unknown. Jumping out of your comfort zone is not an easy deed and without outside help, integrating in a new environment is even harder.

In our time and age there are a lot of sources of information, a lot of unfiltered content and this huge amount of resources is overwhelming to some of us. Here we are talking about media reports, a lot of suggestions regarding friends, travel destinations and ads over ads overflowing our news feed on every social media channel existent.

With our project we are aiming to eliminate at least parts of the enumerated issues.

State-of-the-art

- What did the others, techniques and methodologies used, evaluation methods, results:

  We will take as reference the most popular travel application available on the market, **TripAdvisor**. This platform offers popularity rankings across vacation destinations all over the world, using a popularity algorithm.

  The Popularity Ranking is based on the quality, recency and quantity of reviews that a business receives from users — and the consistency of those reviews over time.

  As of 2011, (the most recent report we found) the web application has 40 million monthly unique visitors, a weekly release cycle and daily patches. With only around 100 engineers this does sound impressive.

When talking about the general architecture, the technology stack includes: Open Source Linux, Apache, Tomcat, Java, Postgres, Lucene, Velocity, Memcached, JGroups. The main idea is to keep everything really simple in order to make every software process easier (build, debug, deploy, maintain and operate). A bank of stateless Java web servers running simple Java and Velocity templates, a bank of services, as each functional area is packaged as a stand-alone piece of software, exposing high level API optimized for over the wire performance.

Moreover the architecture is fail-safe, assuming things fail. Either have plenty nodes in a cluster (web servers, service machines), or have true N+1 redundancy (databases and the datacenter itself).

- Important names in the field, research teams

Constraint programming can be expressed in the form of constraint logic programming, which embeds constraints into a logic program. This variant of logic programming is due to **Jaffar** and **Lassez**, who extended in 1987 a specific class of constraints that were introduced in **Prolog II**. The first implementations of constraint logic programming were Prolog III, CLP(R), and CHIP.

- Relevant links

http://www.jot.fm/contents/issue_2014_09/article1.html

https://en.wikipedia.org/wiki/Constraint_programming

http://highscalability.com/blog/2011/6/27/tripadvisor-architecture-40m-visitors-200m-dynamic-page-view.html

- Resources and tools available

The resources are vast, any open source tool or framework can be integrated in our work. We want to deliver a web application, so this is the area we will be the most interested in. Narrowing even more, we will focus on the Java ecosystem. Spring framework is a powerful open-source tool that will ease our web development work. It also comes with a generous documentation and a lot of tutorials on their official site.

The presentation layer can be created using any front-end technology, from JSF if we want to remain in the Java scope, to plain html/css mixed with Javascript.

Other tools necessary for development will be Apache Tomcat, JUnit, Maven, MySql, all of which are well documented, with plugins available for IDE integration (Eclipse/IntelliJ).

Solution

Our implementation is a web platform that offers the possibility to organize travel plans. Based on what the user is interested in, attractions and points of interest will be suggested to them. The suggestions are user friendly, not intrusive. The user can choose his own destinations, but also take into account our proposals. In this way, when preparing a vacation, the people who are using our software will be assisted with hints. This can enrich someone's horizon and knowledge about different cultures and give them the chance to explore places which maybe they would not know about.

Moreover, our services include a user matching functionality. This is done taking into consideration common elements between users. These include interests, travel destinations which were visited in the past, which the person is currently visiting or planned to visit in the near future. Everything is summed up and calculated as a score.

Taking into account that some users might be popular than other, we want to deliver a fair experience to everyone using our application. This is why we implemented a penalty system which favors the new users, or the ones with less matches. With this initiative we hope that people in our community will not feel left out, by contrary, will feel part of the whole. In the initial plan, after being matched, the users could initiate a conversation. This feature would have emphasized our goal: helping users connect with the people around them during their trips.

We aimed for a scalable architecture and low coupling. Our main module exposes a number of services for any presentation layer. We built a front-end web application, but any kind of consumer which supports HTTP protocol is possible (mobile, desktop, smartwatch etc). The main module is connected to a MySQL database hosted on another server. The motivation behind this decision is that

during development to have one local environment for each developer to work, but also a database shared by everyone, populated with real life data.

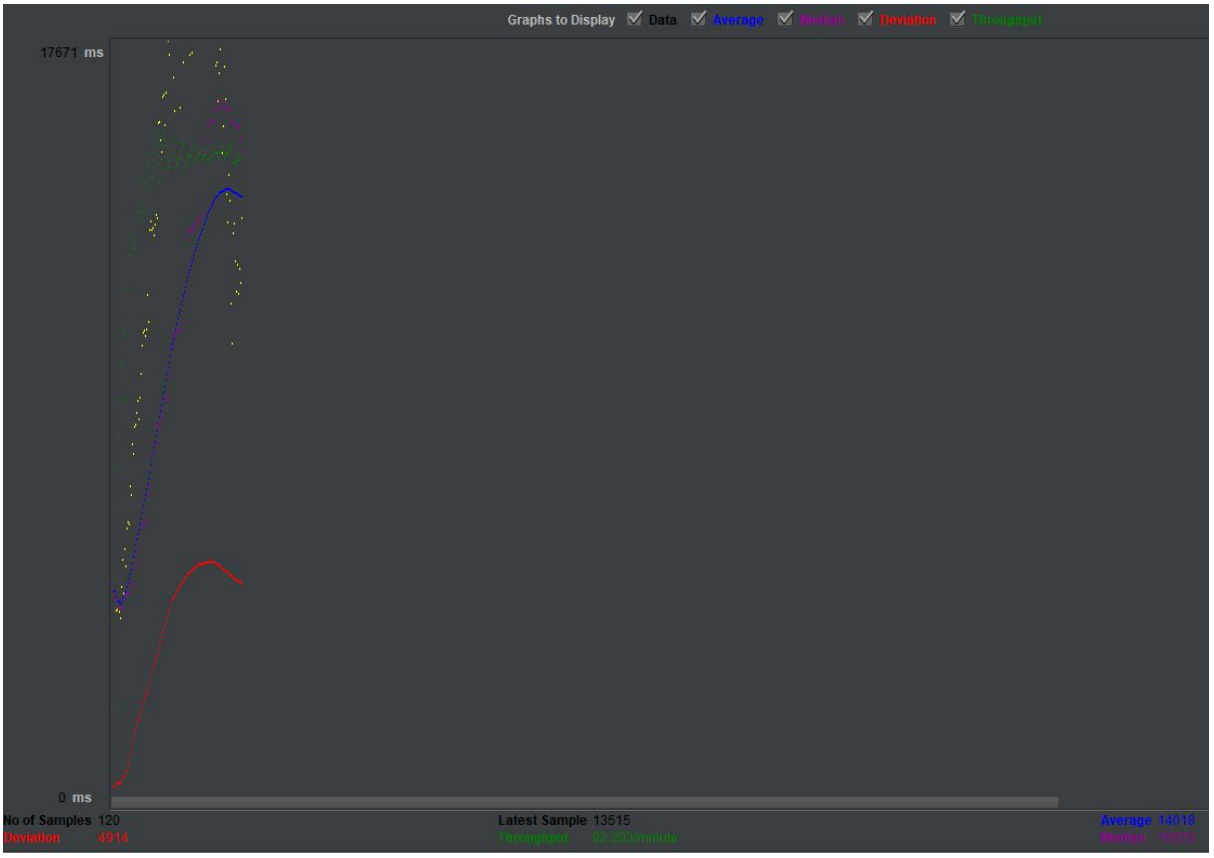
In addition to this, the core module was developed using Spring Boot and Hibernate for data persistence. The design was tiered, as follows: controllers (exposing the services), services (processing information + business logic), repository (data accessing layer, bridge to the database). AOP was used to measure the execution time of services methods and logged for future statistics and analysis. Wrapped as a war, the module can be easily deployed on almost any application server, enterprise or not.

Results

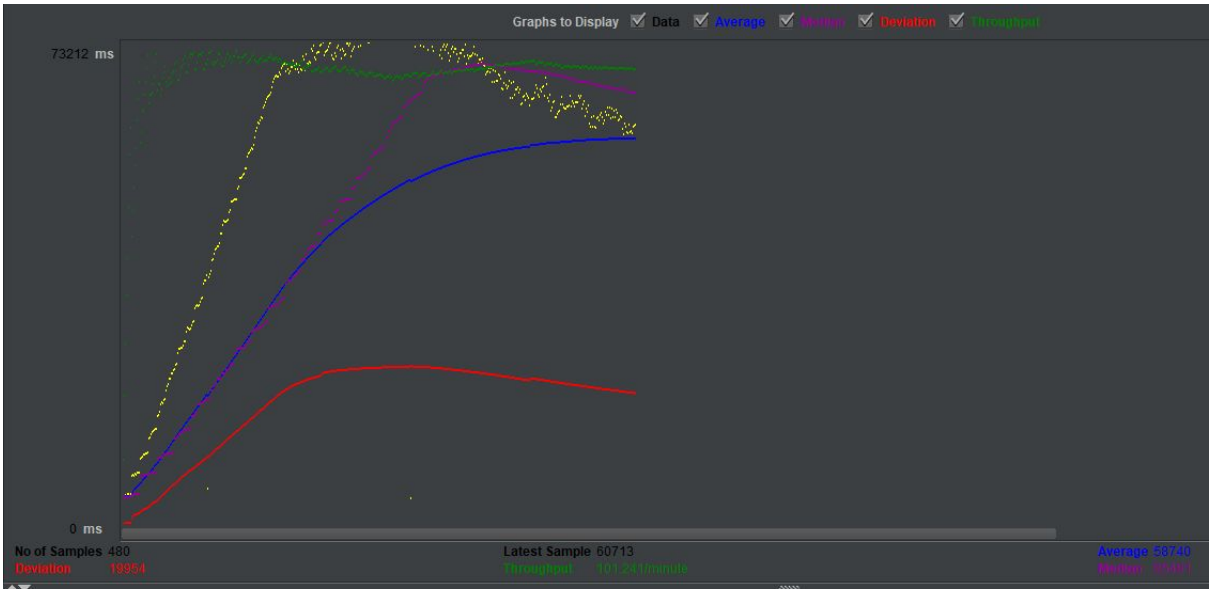
After finishing all the sprints, the application is ready for deployment. The developed modules were integrated and functional. We successfully developed two modules, back-end and front-end, communicating with each other and delivering the travel functionalities presented at the beginning of the course.

If the server is under great traffic it will fail to answer all the requests. Testing with jMeter and analyzing the logs for execution times of the service methods revealed a need for optimization when fetching data from database. In order to enhance the performance a little bit we added second level caching on some of the entities which are updated rarely, like City or Country. This is a feature of Hibernate and configured by our team.

1. Testing our server with the default memory (256m heap size) and with 30 user sending request to two of our endpoints (retrieving the travel plans for a logged user and also all the cities persisted in the database), iterating twice for each (120 samples), the server handled all the requests. The number of requests processed were 92 per minute.
2. For diversity purposes we increased the number of users to 120 and also increased the JVM memory size to 4096m. Again, all the requests were successfully handled with a throughput of 101 requests per minute.
3. A third test was made with data from a real life scenario. We assumed that 10.000 users accessed a couple of our endpoints with a ramp-up time of 60 seconds. This proved to be way too much for our machine to handle.



The graph for the first test plan



The graph for the second test plan

The third test plan was stopped after 10.641 requests, with a throughput of 1.364 requests served per minute. After 8 minutes the 10.000 users were calling our API, but the error rate was too high. The final results were not concludent.

Comparison with other solutions

We will take TripAdvisor as a reference. The comparison is far fetched, given the fact that we lack the experience and budget of such a giant, but some of the functionality is similar.

The differences between the two application are:

- Unlike on the TripAdvisor platform, on our application the user can plan his trips, not just see suggestions
- Our application does not offer the review functionality, but we use the reviews already created by Google, by calling their Google Places API.
- On our platform, besides connecting with places, the users can also find people that match their profile (interests, or future, past or present travel destinations)
- Lacking the hardware capabilities is our main disadvantage. Our infrastructure cannot compete with the greatest player on the market for this niche.
- The news feed system, which we planned implementing, but did not achieve, is present in TripAdvisor. Accommodation offers, restaurants and posts from other people visiting different points of interest can be seen on the feed of users.

Future work

As for improvements, a logging mechanism could be implemented. This would improve debugging and general maintenance of the software. Log4j offers this functionality and can be programmed on levels of tracing. Combining this with AOP with result a powerful tool.

If the number of users exceeds our server's quota, a load balancing mechanism may be needed, combined with horizontal or vertical scaling, depending on the budget. This will also improve the response time. By doing this we could also upgrade our architecture in order to apply for the microservices design, breaking the monolith and offering a completely decoupled approach, This could greatly improve scaling and user experience, and also clearly separating the concerns in the application.

Chat functionality would be nice to have in the future and also the news feed presenting different offers from airline companies, restaurants or accomodations. These can be fetched using different APIs, TripAdvisor being one of the providers.

Conclusions

Even though the project did not reach its initial expectations, which were from the start really high, the final product is functional and every member of the team could see his contribution materialized in some part of the application.

Team:

Chilaboc Ecaterina-Mihaela
Herdes Andrei
Joldescu Catalin-Petronel
Oana Daniel
Podaru Rares-Ioan
Stolnicu Simona-Tereza

Bibliography

- http://www.jot.fm/contents/issue_2014_09/article1.html
- https://en.wikipedia.org/wiki/Constraint_programming
- http://highscalability.com/blog/2011/6/27/tripadvisor-architecture-40m-visitors-200m-dynamic-page-view.html
- https://www.researchgate.net/publication/288204978_Service_oriented_architecture_A_research_review_from_the_software_and_applications_perspective
- https://www.baeldung.com/java-collaborative-filtering-recommendations