Technology Upgrade for Forsa

Porting Forsa Application to
MEAN Stack

1. About the Application

Forsa is a single page applications(SPAs) which allows lenders and borrower to perform deals online. It presents best live deal to both of the parties. These business logics need up to date information to find out such a deal for getting best out of live market. We need to make loads of request to server to feed the market changes got from client(s) and get them back to the client(s) across the platform. Those requests are integral part of the application as they enable the data on screen to be updated instantly as the market changes.

2. Current Architecture and Problems

Current architecture comprises of majorly two technologies: CodeIgnitor(A PHP framework for backend) and AJAX for information updation to and from server. As mentioned earlier, as application needs real time data always on screen, we have to hit server and ask for the latest information using AJAX calls which is also known as Polling. Given that, say, we need 10 requests per user per second to check for change in data and if we have 10 simultaneous users, that would be 100 requests per second to the server. And which, undeniably, will increase exponentially with increase in number of concurrent users. And this increase in requests would need more resources and can affect the overall performance of the application. That’s one. Second is scalability. Though current application architecture can suffice the need of 10k users, but we would have to scale our resources accordingly. The whole problem is because of the “not dedicated” architecture as per the behaviour of the application. Simple SPAs with nominal updations on page can be implemented using the current architecture but complicated applications like and Forsa would increase number of complications in terms of implementation, performance, scalability as well as maintenance.

3. Solution

Thanks to vastly growing field of web technologies, we are introduced with great technologies for specific use-cases day-by-day. So, that said, we have two technologies for our use-case(i.e. Single Page Application or SPA).

a. WebSockets

This is the excerpt from the official website,

“WebSockets are an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.”

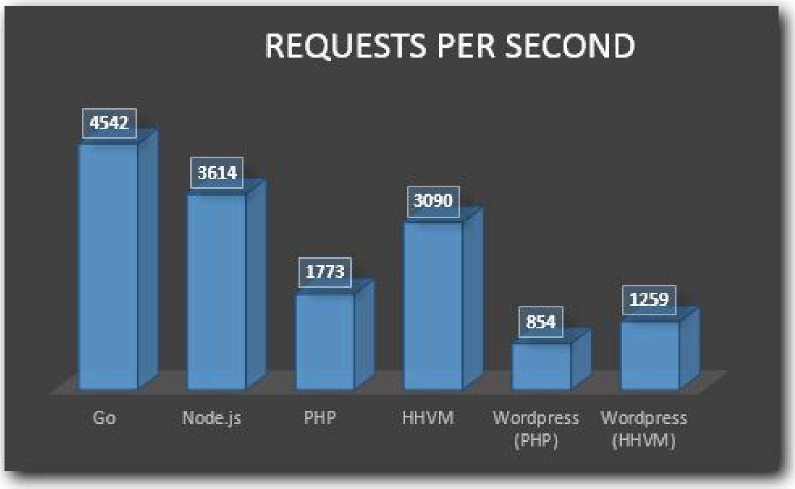
It sounds like a wonderful idea to choose this technology as we don’t have to poll for the data every seconds. It would greatly save us from overhead of polling which goes to the server with a heavy header and takes back information, if any, that too with a heavy header. Websockets can save us from that as it creates a connection with the client and stay with it until and unless he/she closes the application. Now this is the problem, giving undivided attention to a client would, of course, need resources allocated to him/her dedicatedly. So if number goes up, we have to scale out which would be very costly as compared to scaling up. Second, this is relatively new technology and we do not have enough community to reach out for any guidance or support.

b. MEAN

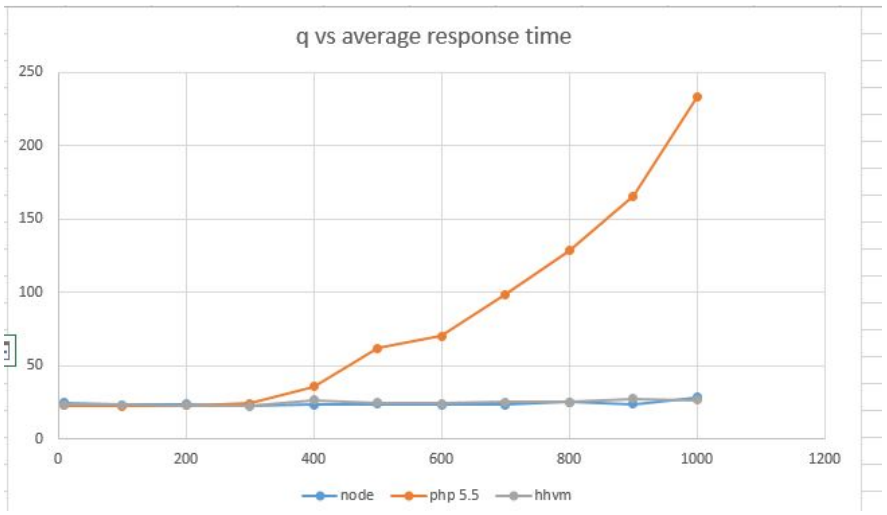
MEAN is a backronym stands for combination of 4 great technology put together which are: MongoDB, Express.js, AngularJS and Node.js. This technology stack perform fine for almost every use case but is acclaimed specially for SPAs. Node.js is runtime environment written with C++ on Chrome V8 engine. It allows developers to run JavaScript for server-side. Node is single thread but follows non-blocking I/O aka. Asynchronous I/O. Being single thread, it can handle way more requests than its counterparts(See the benchmarks below). Also scaling up will suffice the need of more power if needed. Express.js provides a way to develop APIs more effectively and efficiently. The data to and from API are glued to the Angular directives. Angular takes care for the data updation. Hence we need not to poll server manually as we did in the case of AJAX. Angular is way more efficient than AJAX requests. Coming to MongoDB, it is faster, easier and more scalable if we plan to scale out(scale horizontally).

4. Benchmark

a. Request Per Second



b. HTTP + CPU tasks



c. CombSort Strict CPU Test

	CPU time	System time	RAM
PHP 5.6.4	102.69s	104.20s	2497508 KB
HHVM 3.5.0	12.56s	14.83s	362488 KB
Node.js v0.10.35	2.64s	2.64s	92240 KB

So as the numbers speak for themself, node is clear winner here. Combining it with Angular, Mongo and Express, we can take great boost in terms of overall performance of the app. And for the longer term, it is the best architecture economically as well.

5. Deployment Pipeline

Now, as we are done talking about application technology stack, let’s talk about the deployment pipeline. We plan to use Git as our VCS. That way, we would have better track for what is being done and by whom. We will have multiple branches for multiple instances as DEV, TEST and LIVE. The features requests and/or bug reporting will be catered using JIRA. And the implementation will go first on DEV then to TEST for testing. Once the testers signalled green, the changes would be finally pushed to the LIVE. Each Git branch will be connected with Continuous Integration service such as CodeShip, GitLab CI etc. It will take care for final steps of testing to deployment automatically.