Nova code challenge

# Intro

Welcome to my write up of the Node.js backend challenge. I would like to start out by saying that I personally love interviews like this as they allow to me to not only test my skills against a fun and challenging mini project but also provides me with an opportunity for learning and technical growth.

Just as an FYI, I have little experience implementing full stack solutions for web projects. Working in the front end isn’t something I have a lot of practice with and I find that the lack of good debugging tools as well as sometimes unpredictable behavior turned me off years ago. I have also never worked with node.js and honestly have done zero javascript coding over the last few years so it was a lengthy process to get my development cycle moving. All that being said, I have enjoyed working on this project.

I’ll be splitting up the write up into sections that I feel represent my own thought process as well as the intended objective you wanted from assigning a code challenge

Contents

[Intro 1](#_Toc465200674)

[Architecture 2](#_Toc465200675)

[Front end 2](#_Toc465200676)

[Back end 2](#_Toc465200677)

[Client page (Iframe) 2](#_Toc465200678)

[Server 3](#_Toc465200679)

[Logging DB 3](#_Toc465200680)

[Deployment 3](#_Toc465200681)

[Testing 4](#_Toc465200682)

[Final Thoughts 4](#_Toc465200683)

[Challenges 4](#_Toc465200684)

[Cross domain communication 4](#_Toc465200685)

[Javascript debugging 4](#_Toc465200686)

[Security 4](#_Toc465200687)

# Architecture

I tried to follow spec as close as possible including everything described and keeping it as simple as possible. I avoided heavy include solutions because I wanted to keep size of project down as low as possible. Below is back workflow

* Form data hits JonnyLender.html and gets processed for PostMessage
* Inner Iframe receives Postmessage on event listener and loads data into its div
* Inner Iframe makes ajax request to server putting Customer info in querystring
* Server receives ajax request and processes querystring.
* Server compiles customer ‘credit score’ and uses that as response to ajax request
* Iframe receives response from ajax request and sends it back to parent window through postmessage.
* Jonnylender.html receives postmessage from iframe via event listener and displays credit info in div.

## Front end

Being a backend developer by day, I tried to minimize the front end as much as possible. Here I simply created 1 static HTML page hosted on my local machine through no domain. I called this file JonnyLender.html as it is the bank side of the system. This main front end page contains an empty iframe pointing to the the node.js server, a form to include customer data, and a div to display information returned by the iframe

[INCLUDE PICTURE OF PAGE HERE]

* Iframe
* Form
* Div1

## Back end

I found this to be the most enjoyable part of the project for me. I felt more at home here as I could easily debug as well as use non browser console printing. I split the backend into 3 sections, the client page (which is the iframe used by the front end, the node.js server, and the logging DB

### Client page (Iframe)

This is the page that contains part of the magic that is the backend. It is the main point of communication between the frontend and the server to ensure any data flowing between the two is what we want and expect. It contains the following

* Window event listener: Place to receive message from the parent window. This is crucial from a cross domain communication stand point. I struggled to find a good solution but after many attempts, found that window.postmessage plus window.addeventlistener was the best way to go.
* Ajax request sender and receiver: Use this to communicate with the node.js server so that I didn’t have to send round trip call backs. This was relatively straight forward as it is a well-documented and solved problem.
* Divs to display data: Use this to display customer data received from parent window and node.js server

### Server

The node.js server in this case creates 2 paths for requests to go through: forwarding request to the indexpage.html which displays the client page, and a path for ajax requests to come through.

* Indexpage route: Basically reads in the indexpage.html and writes it out to the response. I did it this way just to ease my own development and readability of the indexpage.html. Not sure if this is how it’s done other places though
* Ajax route: I check to determine whether or not a page is attempting an ajax request by checking the path of the request. If the path contains “/getCustomerData”, then I send through the request through to get a random reply from a list of ‘choices’ and add the reply to the response.

### Logging DB

Here I opted to make a simple text file that does the logging. Adding in a sql database is a pretty trivial task and ended up complicating the design quite a bit. That being said, I’ll offer some info on why text databases can offer better storage for some systems.

Pros

* Ease or readability: Anyone can open and read
* Easy to export to excel: Useful for people who want to do quick sorting using pivot tables without sql knowledge
* Reads and writes are fast
* Sorted writes make reads faster (something sql inherently does)

Cons

* Multiple process may encounter conflicts (can always log to separate files)
* Hard to do joins (not impossible though)

# Deployment

I chose to go with the visual studio version of the node.js server add-in and it made the deployment and transfer of the project super easy. (There’s probably an easy way to deploy without Visual studio using just the files but I didn’t fully explore that). Steps to deploy

* Download project from GitHub (<https://github.com/jonnyhay/NodejsWebApp1/archive/master.zip>)
* Running with Visual studio is easiest so if you don’t have VS2015, you can download the community version of the IDE
* Install node.js visual studio plugin (https://www.visualstudio.com/vs/node-js/)
* Open .sln file with visual studio

Once running, load up the server. You’ll first want to open up JonnyLender.html. This is a static html file that shouldn’t require any server running. If the internal iframe doesn’t load, that means your server isn’t running. Once the server is up and running, enter in the information requested in JonnyLender.html. Upon hitting submit, this should start the system workflow described in architecture section

# Testing

I performed mostly manual testing throughout the development phase but here’s a brief explanation of what my testing plan would be.

* End to end testing
  + Write a tool to achieve end to end testing meaning, from JonnyLender.html, pass in some set of inputs, and compare with predetermined outputs that should come out.
* Black box testing
  + At various stages of the process workflow, write tests that check end to end functionality of that component.
* Unit testing
  + Every function will also need to contain unit tests that get validated against various inputs
* Tests like these will include a variety of (bare minimum).
  + Border cases
  + Different languages
  + Large inputs
  + Blank inputs
  + Improperly formatted inputs
  + Nonescaped strings
* These tests will be run on every check-in as well as periodically throughout a given week/day (depending on scope)
* Previous runs will need to persist to compare current runs with previous test runs

Once verified that the app works, we’ll test against reliability and scalability. This is out of scope for this current code challenge but I thought it would be important to note

# Final Thoughts

## Challenges

### Cross domain communication

Getting cross domain communication took me a little while to figure out. There are so many resources available online that I ended up trying solutions from back in 2005. After trying many solutions (none that worked ), I stumbled on PostMessage and that solved it for me. Since I had very little knowledge of how browser dom components work together, I had to do a lot of learning of basic browser security and capabilities

### Javascript debugging

I had a hard time getting a good development cycle formed. I couldn’t figure out an easy way to step through the javascript code I had written and so I settled on extensive file and console logging. I found eventually that you can debug in most browsers as well as in visual studio.

Another issue was that I couldn’t figure out how to debug inner iframe in browser. I eventually resorted to file logging but late in the project I discovered I could load the inner iframe directly through a test page I wrote that would mimic parent window but directly.

## Security

I left some of the domain specific sections empty/commented out. This would obviously need to be included if this project shipped.

* Every Lender would have to include Nova domain in their list of approved domains
* Nova iframe would need to have a whitelist of approved users. This would be best stored in some off site config file (sql, separate service, etc) and requested direct from the iframe.