|  |  |  |
| --- | --- | --- |
|  |  | Software development Projects |

Working in an agile environment (TDD, BDD, DDD Pair Programming etc.)

I hope this gives you a good outline on some of the major contributions I have been able to provide as part of my roles in each company.

All images shown are publically available to anyone that has purchased these products and shown here for illustrative purposes only.

# TRUFLA TECHNOLOGY

Senior Software Developer/Architect

## REACT INSIDE EMBER.JS

**Problem:** After the first month or so of planning and the team coming up with a grandiose vision there came the time to put theory to practice.

The first obstacle in our way was working against a fairly old version of Ember.js (2.13.1) which had zero tests without compromising the current functionality.

Of course, I was quite biased towards going down the React route, but I feel it was the only obvious practical solution without having to start from scratch which was a non-starter.

A screenshot of a cell phone

Description automatically generated

**Major issues solved with this approach**

* Ember version was extremely out of date
  + Getting Ember.js upgraded to 3.3.x was difficult but going directly onto 3.4.x was risking the whole project
* Working within the constraints of Ember.js proved to be a development nightmare due to their overall lack of documentation and questionable design decisions.
  + Unable to upgrade fully to 3.4.x due to the number of breaking changes and risk it posed
  + Cannot compete with its competitors for community libraries and accompanying libraries
  + Ember.js are going through major changes in an effort to catch up which brings with it even more risk.
  + React is the only technology that could make this possible due to its flexibility by design to work inside any HTML

**Major findings**

* Turned out to be a hard sell due to so many unknowns & lack of frontend knowledge within our team of three core developers.
* The biggest delays by far were compilation issues when attempting to add new libraries into the mix. Ember.js uses their own build tool namely broccoli which did not play well at all without significant effort.

**Outcomes**

* Helped make the seemingly impossible possible in securing a massive contract.
* Updated the old look and feel to better match the new as exemplified in the image above.
* We turned zero tests into hundreds
* Released new functionality without causing any obvious issues with the old functionality.
* We continue to attract more talent to organization.

DYNAMIC FORMS

**Problem:** Create forms as dynamically as possible in order to keep up with future needs, improve scalability and reduce maintenance cost.

After another month or so of research and prototyping we came up with a working example of creating forms directly based off of JSON Schemas.

In the end we went with RJSF (React Json Schema Form) on top of our own Ant design components.

**Major issues solved with this approach**

* Once we built the first set of forms in order to add more you simply needed to declare a schema for each form.
* Gave us more time overall to focus on full end-to-end testing via Cypress

**Major findings**

* We ended up swapping out most of RJSF components with their Ant design counterparts mainly due to the added maintenance cost of customizing their components to our needs.
* Ant Design turned out to be another unexpected coup which was made easier when RJSF introduced themes.
* The biggest pain point by far has been dealing with re-renders and conditionals but again with teamwork and effort we overcame these.

**Outcomes**

* We now are able to create our forms solely based upon JSON Schemas
* We had a lot less to test so we could focus more on the full end-to-end scenarios

# HRand/Medi-Direct Inc.

Senior Software Developer/Architect

As part of architecting and developing the system I have been a ferocious learner and tried out many different technologies and techniques before coming up with a framework that is truly pleasing to the eye and is extremely easy to work with.

I work hard to ensure developers have excellent tools and helpers to ensure they can develop in a concise and consistent manner by hiding complexities under intuitive interfaces that are well documented and fully tested.

Because of this undertaken we as a team find it very easy to refactor major pieces of our codebase without having to worry about breaking anything due to how much API and maybe more importantly UI testing that hits every piece of functionality and makes things really scalable for bringing on new developers and will allows us to grow fast.

## ADVANCE BDD TESTING FRAMEWORK

**Background:** Early on in the project it was decided that we would develop with the Behavior Driven Development process. The idea was to have tests to ensure we are building the product in a way that is transparent and understandable to our stakeholders.

**Motivation:** I was finding that there was a massive amount of boilerplate and lack of maintenance when we initially started creating API and UI tests against static Behaviors. Early on I recognized this as a risk and collaborated with the team and it was agreed that I would come up with an automation process to speed up development overall and address the risk of having separate processes for the API and UI.

[Sample Feature File](https://github.com/desoleary/BBD-Testing-Infrastructure/blob/master/README.md)

I have been able to take the idea of Behavior Driven Development to the next level of automation by building a framework that does some of the following:

* All steps are implemented in a generic way in order to promote re-use across any number of resources.
* Converts background steps into fixture data which can be referenced in any other background/scenario step for convenience.
* Detailed instruction on the syntax for each step so that developers/business can come up with clean scenarios.
* A request is built up based on given input from each step which is then sent to the API endpoint and there is an automated process to verify that the data sent was actually saved correctly and audited.
* The same request is used based on changes made and is then used to control the UI and again there is a check to ensure everything is saved and audited as expected.

[Sample Step Implementation](https://github.com/desoleary/BBD-Testing-Infrastructure/blob/master/HasRequestBuilderBehaviourSteps.php)

**Some outcomes:**

* Provides developers with a clear and concise interface that hides the complexities so they can focus a lot more the task at hand
* Extremely pluggable and adaptive to handling new resources and UI pages with minimal effort
* Provided helper methods that interact with the UI automatically based on the step e.g. if the step includes a reference it knows it will be updating an existing field.
* Simply fun to use and unbelievably well received.

# Hubstaff Inc.

Senior Software Architect

My responsibilities as a software architect was to take high level requirements and come up with functional & technical requirements for each team of developers to undertake. I was working directly with the CTO and Lead architect and was able to get through an intensive trial.

Both projects (timesheets & time off features) were successfully completed within the quarter and at the same time I was working on my own side projects.

## TIME OFF

The high-level requirements here were to implement a system to allow for the creation of public holidays (paid & unpaid) and time off policies including unpaid, hourly & yearly accruals. In order to come up with a comprehensive document I did a lot of research on the problem at hand and also when through many competitors that with successful products and after multiple iterations landed on a final design. This included mockups & workflow diagrams.

## TIMESHEETS

The high-level requirements here were to create timesheets based off of activity recorded via Hubstaff client applications. Timesheets would display details of the activity including any prolonged idle time with each member with timesheets enabled expected to submit each based on pay schedule. What made this more complex was the interactions with other systems including invoicing, billing and scheduling. With a lot of effort, collaboration and iteration I was able to come up with a detailed and approved design.

## SSO (FRONT AND BACK CHANNEL LOGOUT)

One of the first projects I worked on as part of my trial period was to close up potential security holes with the pre-existing SSO system. This included implementing front channel logout based on the official Open ID specification. This was used when a normal logout is performed to ensure that all sessions under the same browser was invalidated.

The second piece was to implement the back-channel logout to invalidate all sessions regardless of browser by issuing a request for the identity provider to request each relying party to sign out completely.

# Cisco SYSTEMS

Senior Software / Product developer

*All of our major features were featured flagged to allow for continuous integration and quick rollback if a significant bug is found on production especially due to the sensitive nature of the features we were developing.*

**MODULARIZE AUTHORIZATION**

One of the senior developers on extracting our login process into a separate engine and now it serves as a micro service for use across multiple applications. As part of this work we also introduced an SSO option and our own Identity Provider.

**INTRODUCED PARTNER ACCOUNTS**

Lead developer & team lead on a brand-new feature that now allows partner companies to service their clients and produce accurate reporting like never before. This is one of the highest priority projects of the year and is expected to bring in a lot of revenue

* Introduced Single Table Inheritance (STI) to differentiate major account types. Provided good encapsulation and separation of concerns.
* Devised a daily report that records summary information on all accounts sourced from MySQL, Cassandra and MongoDB
* Used the daily report to produce aggregate reports on individual partners including breakdown per customer
* Came up with a Partner Dashboard that gives them a quick look into the health of their customers
* In order to manage individual accounts, we offered the impersonation feature to allow them to manage customer accounts on behalf of the customer

**Partner Customer Dashboard**

A screenshot of a cell phone

Description automatically generated

**Individual Partner Customer Overview**

**A screenshot of a social media post

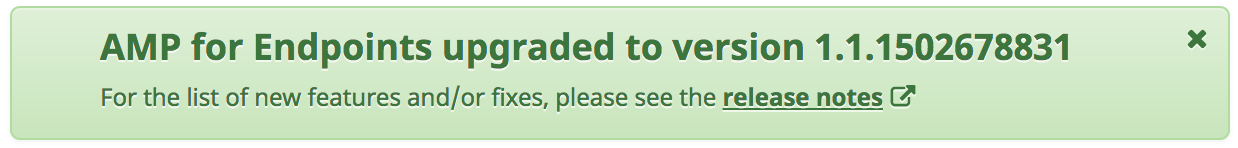
Description automatically generated**

**GLOBAL ANNOUNCEMENTS**

**Backend Details**

* Implemented on top of Starburst Gem Foundation
* Includes announcement views table in order to keep track of items read by user
* On click of /announcements all announcements are viewable and marked as read

**Frontend Details**

****

* Announcements (General, Maintenance [**very rare**] and Upgrades as shown above)
* Includes preview as you type

**BUSINESS CREATION REVAMP**

Project to revamp how we provision user accounts

**Backend Details**

Biggest issue here we've found is DB locks

- Figure out where the dead locks are coming from

- For us it was the provisioning code that was causing major issues

    - Went from old attributes tables to having one table with many columns

    - This was done in stages

    - Removed redundant items based on info on prod, many became default values that never change so no need for DB entries

    - Done piece by piece until complete

    - Moved provisioning to Delayed Job and ensured it worked much the same as before

 - Improved the experience for the user and allowed them to do a lot more work instead of creating multiple tabs

**Frontend Details**

**A screenshot of a cell phone

Description automatically generated**

* Unified error handling / client and server side validation looks the same whether on the form or alert

**A screenshot of a cell phone

Description automatically generated**

Instead of just coming up with the standard fare I went a little further and still managed to deliver on time and customers loved it

* including email syntax highlighting
* validation is done using email validator that can handle multiple fields
* one click of item it goes into editing mode immediately
* x icon is highlighted on hover
* Inspired by Gmail’s version

**IMPERSONATION**

Allow our support staff to impersonate customer accounts

**Backend Details**

* Found out how this is done in the wild and tested many implementations until I landed with pretender gem and implemented changes on top of this
* Audit log all actions taken by the impersonating user
* All actions by the impersonating user displayed for the customer to see
* In order for a customer to opt into this feature they had to accept an agreement

**Frontend**

* Support admin only
* User has to provide reason
* Came up with widget to make it obvious we are impersonating another user

A screenshot of a cell phone

Description automatically generated

**A screenshot of a cell phone

Description automatically generated**

**RAILS 4 & RUBY UPGRADE**

Lead developer on Rail 4 Upgrade. Had to be done very carefully to ensure there was no customer impact.

Ruby upgrade was done with the use of feature flag and only globally enabled once all specs were passing for both Ruby versions. Rails 4 upgrade was done in a similar fashion. Ended up with no customer issues.

* Beefed up tests to ensure functionality works the same before and after the upgrade
* Replaced deprecated features
* Upgrade gems
* Fix broken specs

**BOOTSTRAP 3 & SITE WIDE LOCALIZATION**

Lead developer on our new multi-language offering which involves localizing text across the application including handling of constants stored in the database.

Heavily involved in migrated from Bootstrap 2.x to 3.x without any production issues raised. Genuine massive undertaken that was achieved due to excellent collaboration between multiple teams.

This gave us the opportunity to clean-up the overall UI to use the newer look and feel that we introduced throughout the application. This was done with extensive collaboration with multiple team members and we came up with FAQ wiki and task list to complete the work on time and without any customer issues.

* Created a FAQ wiki in order to keep learnings / tips in the one place (this made it easier to bring others on to the team)
* Used resources that helped pinpoint necessary changes from bootstrap 2 to 3
* Feature flagged
* Formulated another FAQ for tips on adding text to localization files, this helped tremendously in keeping the workload down to a minimum for the translators