Internship Report

Jingkang Zhang
University of California, Berkeley

May - August 2019
At
ByteDance

During the summer of 2019, I worked as a Software Engineering Intern at ByteDance, with a focus on front-end web development. Through this report, I intend to illustrate my experiences, describe my responsibilities, summarize what I have learned during this internship, and compare them to knowledge learned in school.

To start, I was on one of the front-end teams in *Toutiao* tech devision. Our team's responsibilities include working with other dev teams to ship products and new features both for clients and for internal platforms, and building tools/wheels to facilitate future developments. I mainly worked on the front-end side of management platforms, both for users and for operation managers, and briefly got my hands on some server-side programming, creating APIs and writing web crawlers.

Chronologically, I started off working on a Client Relationship Management System for managing authors (who writes articles which in turn feed our readers). After the project was configured, I started contributing to some of its pages. I implemented a table that displays entries of data according to filtering conditions specified by input fields above it. Before taking on the task, I first had to brush up my HTML, CSS, and JavaScript skills while acquainting myself with my team's standards on code styles, editor configs, project structures, and git workflow. Then, in conformance to my team's tech stack, I learned TypeScript, LESS, and React Hooks. Omitting the apparent technological materials learned in each of these common languages/skills, I'd like to jump into the project-specifics. While implementing the UI, I made use of one of the bootstrap-like UI libraries called AntDesign. With ready-made UI components, I was able to quickly construct the page. When it comes to managing the

states and operation logic of the page, I experimented with the new React Hooks and attempted to seal logics specific to the filtered table, including event-handling, state storing/updating, and data fetching, into a single hook. This way, the logic is easily re-used among other filtered tables in the form of a hook, which was named in the same sense as it can be simply "hooked onto" other components. Designing and implementing this hook constituted one of the major challenges during my whole internship: What should the hook take care of and what not? What does the hook need to take in, and what should it return? How should the inner state of the hook be managed? etc. These were all questions with multiple answers, and required me to decide on one that not only works, but does so elegantly. All these challenges were even further complicated with the adoption of TypeScript. While TS offers convenience when *using* an API, it demands equally more work from developers who *write* the API. I had to carefully design the generics pattern in order for TS to work properly with the hook.

After the first version of the CRM system was deployed, I turned to work on another platform. Rather than CRM's purpose of serving the operation managers (so that they can manage the authors), this platform is more of a user dashboard on which the authors manage their accounts and articles. I'd like to introduce two main features I implemented on this platform, a level-displaying dashboard and a form with validations. In working on the platform for our users instead of for internal use, I felt the pressure to implement pixel-level-accurate UI (given by designers) and handle edge cases so that the page displays consistently in all conditions. As such, both my CSS and JS skills were tested in writing industry-level web pages. For example, while implementing a progress bar which illustrates the current author level on the dashboard, aside from what one would expect on a normal progress bar, it was also segmented by several nodes as level thresholds. Nodes that were surpassed, current, or ahead have different styles as well as the segments in between. After trials and errors, I managed the class names of elements with some arithmetics with JS and implemented the styles with great help from the ::before and ::after virtual elements.

Next, I moved on to the form with validations. For contexts, the platform was trying to enable the authors to give out customized coupons to readers who'd purchase their articles, and it was through this form that the author specifies various properties of the coupon, including the name of the coupon, valid date range, applicability, types (and properties specific to each

type), etc. While implementing the form logic, again, I looked to use some new libraries. With the adoption of the "rc-form" library which manages the form state, I didn't have to explicitly keep the form states, handle updates, or perform validations. Apart from skills gained the same as what were described in previous paragraphs, I have had valuable experiences communicating and collaborating with other teams. Though these interactions were abound throughout the whole internship, they are even more so during my work on this part: while implementing the validations rules, whose subtle cases hadn't been specified, yet crucial both for operation and security, I discussed extensively with product managers and together determined the rules; due to the interrelated nature of the field validations, some fields may contain errors as a joint effect of other fields, leading me to communicate accordingly to the QA and contribute to their test cases; regarding the APIs provided by my server-side colleagues, I confirmed with them what should the type and unit be for each parameter to prevent unwanted errors.

In addition to the front-end work mentioned above, I had the chance to write a part of the back-end APIs in Go and worked on a crawler project in Node. Though hadn't the time to dive deep into the development, I benefited from the experiences. On one hand, creating APIs myself added to my understandings to how the APIs are designed, and therefore, can be better used. The crawler project, on the other hand, familiarized myself with the HTTP protocol, added to my knowledge on proxy, and trained me to quickly analyze APIs by other websites, with examples but not docs.

Overall, the knowledge I learned throughout this internship is relevant from that in school yet different. I'll first expand on how they are relevant. Needless to say, basic programming concepts from CS101 (well, in my case, CS61A and CS61B) were ubiquitous. Solid understandings on variables, data structures, control flows, loops, functions, abstractions, and classes are required on a day-to-day basis in order to carried out project development in JavaScript. Still, I'd like to highlight two examples where the knowledge learned in school, seemingly unimportant, is indeed indispensable in my course of development.

First is generics. In CS61B, I briefly learned generics in Java. The confusing syntax and abstractness of the concept made it seem all too costly to put into time and make sense of it. "Why not just write proper functions/classes for each of the cases?" Now, after I worked on

my internship projects hands-on, the question was answered to my satisfactions: with TypeScript, React components takes in the types of its props and states in the form of generics. Also, when I was working on the React hook mentioned earlier, I had to use generics in my definition to allow clients to pass in arbitrarily shaped data and still enjoy the advantages of TypeScript. Second, CS61A really tried hard to hammer home the concepts of abstractions and Object Oriented Programming. Yet I didn't understand the significance of the whole lot of practices around higher order functions: "Who would want some functions that take or return other functions in real life? Why not just import and call whichever function wherever I deem proper?" Soon to my realization, in JavaScript, higher order functions are almost a canonical solution to handling asynchrony (barring the new usage of async/await). One would pass in call back functions (probably nested) to asynchronous functions or, with the new *promises*, to the "then()"s of the chained "thenables".

As to how the internship learning experiences differ from that in school, I would analyze in three aspects: the means, focus, and purpose of learning. Discrepancies in the means of learning are apparent: in school, we have lectures, discussions, office hours, homework, exams, whereas in company I would mostly self-study, with help from internal docs and experienced colleagues. The focus also shifts more or less from understanding the principles and ideas in tools/programs to learning how to use the tools. In the company, we value the practical usage of tools and are less interested in knowing their implementations (well, to some extent we might do, but not so much as in school where we'd thoroughly analyze, say, how is a hash map implemented.) Finally, the purpose of learning while in the company becomes more practical, e.g., to ship the product, while in school the purpose varies, from training thinking skills, to obtaining a good GPA for PhD, or to getting ready to go into the industry.

Above shall constitute my report on my experiences, responsibilities, knowledge learned, and how it compares to knowledge learned in school, while I was interning at ByteDance as a Software Engineering Intern during the summer of 2019.