비전공자를 위한

개발자 취업 실전 가이드

목차

이 전자책을 쓴 이유

(편의상 평어체로 작성했습니다.)

체육을 전공하고 29 살에 개발 공부를 시작해서, 연봉 4 천만원 받는 개발자가 되었다. 이 과정에서 진하게 쌓은 노하우를 그냥 흘려보내기는 아쉬웠다. 누군가는 애타게 찾을 내용이고, 누군가에게는 새로운 삶을 시작할 용기를 줄 내용이라고 자신했기 때문이다.

기억에서 희미해지기 전에 하루빨리 기록으로 남기고, 다시 전진하기 위해 이 전자책을 썼다. 지인들이 개발자가 되는 방법에 관해 물어온다면, 나는 이 전자책을 보내주고 내 할 일에 집중할 것이다. 그냥 **이 전자책만 넘기면, 더 이상 신경 쓰지 않아도 되는 것을 목표로 내가 가진 노하우를 전부 담았다.**

노하우를 전달하는 전자책들을 몇 번 구매한 적이 있다. 구매 충동을 일으키는 자극적인 제목의 전자책들도 막상 읽고 나면, '겨우 이 내용이야?'라는 말이 절로 나왔다. 그런 평가를 받고 싶지 않아서, 계획했던 것보다 더 자세하게 썼다.

'개발자 되는 방법'을 검색하던 2017 년 11 월의 나 자신을 잊지 않고 있다. 29 살에 백수가 된 유부남이 개발자가 되려면, 어디서 어떻게 시작해야 할지 막막했기 때문이다. 개발자가 되는 방법을 알려주는 가이드북 같은 것은 고사하고, 개발자가 된 비전공자의 이야기도 드물어서 개발자의 길에 도전하는 것이 두려웠다.

이 전자책은 2017 년 11 월의 나와 비슷한 상황에 계신 분들께 보내는 **가이드북 + 경험담** 느낌으로 작성했다. 과거의 나를 위한 글이라는 생각으로 작성하다 보니, 나의 견해를 강요하듯 전달하는 부분도 있다. 객관적인 시각을 유지하며 읽어야 한다.

이 전자책의 목적은 **비전공자가 빠르고 안정적으로 개발자 커리어를 시작한 노하우 전달**이다. 때문에 전공자와 2 년 이상의 경력자는 예상 독자로 고려되지 않았다. 이 전자책의 모든 내용은 직접 경험한 사실에 의한 것이며, 정답이 아닌 개인적인 의견임을 미리 밝혀둔다.

(이 전자책이 값어치를 한다면, 리뷰 한 줄 부탁드리겠습니다. ☺)

1. 준비하기

1.1. 개발자가 되기로 결심하기

개발자가 되기로 결심하는 것은 중요하다. 개발을 직업으로 선택한 이상, 조기 축구 선수가 아니라 프로 선수의 길을 걷겠다는 것이다. 프로 선수들이 오랫동안 꾸준히 훈련하듯이, 개발자도 실제 업무를 수행하기 위해서는 꾸준한 학습과 경험이 필요하다. 때문에 이 과정을 견디고 즐기기 위해서 결심이 필요하다.

결심한다는 것은 개발에 재미를 느끼고 개발자가 되고 싶은 의지를 갖는 것이다.
개발에 재미를 느껴 보지 못하고 외부요인(유망한 직업이라서, 연봉이 높아서 등등)에 의해 개발자의 길을 선택한다면, 이 직업을 견뎌내기 힘들 가능성이 크다.

- 개발 배우기가 정말 어려운 이유 [번역] - 개발자들이 흔히 겪는 각 단계별 성장통과 극복 방법을 엿볼 수 있다.

1.1.1. 개발의 재미에 빠질 계기 만들기

하지만 아직은 개발의 재미에 빠진 계기가 없어도 괜찮다. 계기가 반드시 우연일 필요도 없다. 필연적인 계기를 만들면 된다. 만들고 싶은 서비스가 있어서 그것을 만들다가 개발의 재미에 빠지면 좋지만, 그런 것이 없더라도 좋다. 아래 무료/소액 강의를 수강하고 따라 해보며 개발의 재미에 빠질 계기를 만들자.

a. 생활 코딩 - WEBn 시리즈
b. 노마드코더 - 클론 코딩 시리즈
c. Edwith - 부스트 코스

a. 생활코딩 WEBn 시리즈
생활코딩 WEBn 시리즈는 개발을 처음 접하는 비전공자들을 대상으로 만든 강의다. 때문에 개념 설명의 눈높이가 비전공자에게 맞추어져 있어서 가장 처음 접하는 개발 강의로는 WEBn 시리즈가 좋은 선택이다. 하지만 동작하는 어플리케이션을 만들어가는 방식이 아니어서 조금 지루해질 수 있다. 모든 강의를 다 들을 필요는 없다. HTML, CSS 와 언어 한 가지 정도만 수강해보는 것을 추천한다. 개발자가 된 이후에는 생활코딩 강의를 들을 일이 거의 없었다. (더 좋은 대체재들이 많다)

b. 노마드코더 클론 코딩 시리즈

노마드코더 클론 코딩 시리즈는 실존하는 서비스들(인스타그램, 카톡, 유튜브 등)을 따라 만들어보는 강의다. 클론 코딩을 통해 웹 혹은 앱 개발에 대해 개략적으로 이해할 수 있다. 개발이 무엇인지 모른 채 Java 문법부터 공부하는 것보다, 가벼운 프로그램을 따라 만들어 보는 것이 개발의 재미를 느끼기에 수월하다. 하지만 클론 코딩의 결과를 본인의 실력으로 착각하면 안 된다. 당신의 클론코딩은 왜 실패하는가?를 읽어 보고 시작하는 것을 추천한다.

c. Edwith 부스트코스

Edwith 부스트코스는 Naver 에서 신입 개발자를 교육할 때 사용되는 코스이며, 누구나 무료(코드 리뷰는 유료)로 수강할 수 있다. 웹 프로그래밍 코스의 경우 HTML, CSS, Java 기본 문법은 알아야 하기 때문에 조금 어려울 수 있다. 하지만 다른 강의들을 병행하면서 이 코스를 이해할 수 있는 수준까지만 천천히 진행하면, 웹 개발의 전반적인 흐름을 익힐 수 있어서 좋다. 만약 너무 어렵게 느껴진다면, 시간을 가지고 천천히 도전해도 좋은 강의다.

1.1.2. 첫 강의 선택 기준

위의 강의들은 2020 년 초를 기준으로 작성되었는데, 이 3 가지 코스 외에도 유익한 강의들이 많이 생기고 있다. 혹시나 좋은 강의가 많이 보이는데 본인 시간은 부족해서 선택 장애가 온다면, 아래의 기준을 가지고 선택하면 실패하지 않는다.

> **기준 1.** 운영체제, 자료구조 등 CS 과목부터 시작하지 않는다.
> **기준 2.** 실무를 위한(Spring, JPA 등 프레임워크) 강의부터 시작하지 않는다.
> **기준 3.** 하기 싫은 숙제 같은 강의는 피하고, 즐겁게 따라 할 강의를 선택한다.

기준 1. 운영체제, 자료구조 등 CS 과목부터 시작하지 않는다.

이 단계에서 듣는 강의는 어디까지나 개발의 재미를 느껴보기 위한 것이다. 기초를 중요시하는 마음가짐은 좋지만, 처음부터 CS 과목들부터 시작한다면 개발의 재미를 느껴보기도 전에 포기할 수 있다. 축구를 해보기 전에 축구공의 원리부터 공부할 필요는 없다. 우선 축구를 해보고 축구 선수가 될지 판단하는 것이 좋다. 축구공의 원리는 축구선수가 된 이후에 학습해도 늦지 않다.

기준 2. 실무를 위한(Spring, JPA 등 프레임워크) 강의부터 시작하지 않는다.

최근 들어 인프런(https://www.inflearn.com/)과 여러 오프라인 강의 사이트에 양질의 실무 기술 강의들이 많아졌다. 그런데 실무 기술 강의들은 실무자가 되고 난 이후에 들어도 절대 늦지 않는다. 토이 프로젝트를 만들면서 듣는 것이 아니라면 아직은 쳐다볼 필요가 없다.

기준 3. 하기 싫은 숙제 같은 강의는 피하고, 즐겁게 따라할 강의를 선택한다.

보고 따라하면서 개발의 재미를 느끼려고 듣는 강의다. 강의 자체가 재미가 없으면 개발이 재미있는지 판단할 수 없다. 지루한 대학교 교양수업 같은 강의는 피하고, 보고 따라하면서 무언가 만들어가는 강의를 선택하자.

Q. 개발자가 되기로 결심한 계기는?

- 한국 화장품을 미국 소비자들에게 소개하고 판매하는 사업 기획.
- 프리랜서 개발자에게 제작비를 내고 웹사이트 제작.
- 소액으로 제작했기 때문에 수정할 부분이 많았음.
- 직접 수정하기 위해 구글 및 유튜브 검색 시작.
- 검색해서 코드를 하나씩 수정해가는 재미에 빠짐.
- 사업보다 개발이 더 재미있어서 개발자가 되기로 결심.

Q. 처음 들었던 개발 강의는?

- 생활코딩 - 웹 어플리케이션 만들기 (당시에는 WEBn 강의가 없었음)

Q. 다시 이 시기로 돌아간다면?

- 강의를 통한 기초 다지기 보다, 만들던 웹사이트 개발에 더 집중할 것이다.
    - 서비스를 만들면서 자연스럽게 마주하는 문제들을 해결하는 과정에서 배우는 것들은 기억 오래 남지만, 강의에서 배운 내용은 금새 잊혀진다.
- 강의를 듣는다면, 눈으로만 듣지 않고 모든 코드를 따라 칠 것이다.
    - 눈으로만 들은 내용은 코딩할 상황이 닥쳤을 때, 절대 기억나지 않는다.
- 시력 보호를 위해 블루라이트 차단 안경을 사용할 것이다.

1.2. 개발 분야 알기

비전공자들은 '개발 = 코딩'이라고 단순하게 생각하는 경우가 많다. 개발 분야가 다양하다는 사실은 간과하고, 코딩을 한다는 사실에 집중하는 것이다. 하지만 코딩은 코드를 작성하는 행위일 뿐이다. 개발 분야에 따라 사용하는 기술과 학습 방법이 다르다. 이 차이를 조금이라도 알고 개발 공부를 시작해야 시행착오를 줄일 수 있다.

1.2.1. 소프트웨어 개발 분야

소프트웨어 개발 분야를 크게 나누면 다음과 같다.

1. 웹 개발
2. 모바일 앱 개발
3. 게임 개발
4. 시스템/인프라 개발
5. 임베디드 개발
6. AI / 머신러닝 개발
7. 데이터 사이언스
8. 보안

1.2.2. 비전공자가 진입하기 좋은 분야

이 중, 비전공자가 가장 수월하게 진입할 수 있는 분야는 웹 개발과 모바일 앱 개발이다. 그 이유는 다음과 같다.

1. 웹/앱 개발자의 채용 수요가 가장 많다.
2. 스스로 학습할 수 있는 자료(책, 인강, 블로그 글)가 가장 많다.
3. 웹/앱 개발자들의 이직이 활발해서 비전공자에게도 취업 기회가 많다.
4. 회사들과 국비지원학원의 직무교육 코스가 대부분 웹/앱 개발자 양성 코스다.
5. 다른 분야에 비해 그룹 스터디를 구하기가 수월하다.
6. 직접 토이 프로젝트를 구현하거나, 혼자서 서비스를 만들기에 수월하다.

무조건 웹/앱 개발자가 되어야 한다는 것이 아니다. 비전공자가 개발자가 되기에 가장 유리한 분야가 웹/앱 개발자라는 것이다. 물론, 그만큼 경쟁도 치열해서 남들보다 더 나은 점들을 어필할 수 있어야 한다.

이쯤에서 국비지원학원에서 웹 개발을 배우고 취업 시장에서 1 년 동안 탈락만 해온 비전공자가 2020 년 2 월 18 일에 쓴 글 (비전공자분들께 말씀드리는 제 경험과 조언)을 읽어보는 것을 추천한다. 웹/앱 개발 분야가 진입장벽이 낮은 만큼, 경쟁에서 살아남아야 한다는 사실을 알 수 있다.

1.2.3. 프론트엔드? 백엔드?

웹 개발자는 크게 **프론트엔드(front-end)와 백엔드(back-end)** 개발자로 나뉜다.
자동차 운전석에 비유하자면, 프론트엔드 개발자는 운전자에게 보이는 모든 부분(핸들, 계기판, 엑셀, 브레이크, 변속기 등)을 구성하고 개발하는 역할이다.

반면 백엔드 개발자는 운전자가 엑셀을 밟거나 기어를 변속했을 때, 차량 내부에서 이루어지는 동작들을 개발하고, 그 결과 데이터(속도 변화, 현재 기어 등)를 운전석 계기판과 기어 변속기에 보내주며, 효율적인 연료 소모를 위해 차량 내부 장치들과 엔진을 구성하고, 동작을 개발한다.

비유 없이 간단히 설명하자면, 웹 서비스 화면을 구성하고 사용자의 요청(행위)에 따른 동작을 개발하는 프론트엔드 개발자와, 사용자의 요청에 대한 데이터를 가공, 관리 및 제공하며 서버를 구성하고 개발하는 백엔드 개발자로 나뉜다.

이렇게 나뉘는 사실이 중요한 이유는, 사용하는 언어와 기술이 다르기 때문이다.
프론트엔드 개발자는 Javascript 를 기반으로 프론트엔드 프레임워크들(React.js, Vue.js 등)을 사용하는 반면, 백엔드 개발자는 Java, Python, Node.js 등의 언어와 프레임워크들(Spring, Django, Express 등)을 사용한다.

이 차이를 알고 있어야 하는 이유는, 선택에 따라 학습의 우선순위가 바뀌기 때문이다.

프론트엔드 개발자가 되고자 한다면 Javascript 를 우선적으로 익히는 것이 유리하고, 백엔드 개발자가 되고자 한다면, Java, Python, Node.js 중 선택하는 단계가 필요하다. (2020 년 한국 기준이며, 국가/지역별로 많이 사용되는 언어와 기술이 조금씩 다르다)

물론, 백엔드 개발자도 프론트엔드 기술을 알아야 하고, 프론트엔드 개발자도 백엔드/서버 언어와 기술을 알수록 업무에 도움이 많이 된다. 만약 컴퓨터공학과 1 학년 학생이라면, 분야를 가리지 않고 학습하는 것이 맞다. 하지만 나이 30 세에 가까운 비전공자라면, 한쪽을 선택하고 시작하는 것이 효율적이다.

개인적으로는 백엔드 개발자로 시작하고, 추후에 회사 업무 또는 토이 프로젝트를 통해 프론트엔드 개발에 도전해보고 넘어가는 방향을 추천한다. 이유는 아래와 같다.

1. **현재의 프론트엔드 기술은 진화가 너무 빠르다.**
    a. 진화가 빠르다는 것은, 개발을 처음 배우는 비전공자에게 큰 부담이 된다. Javascript 의 기초적인 것들도 제대로 사용하지 못하는 상황에서, 새로운 버전의 Javascript 와 프레임워크를 계속 학습해야 한다.
    b. 2016 년에 자바스크립트를 배우는 기분을 읽어보면 프론트엔드 기술이 얼마나 빠르게 변했는지 알 수 있고, 지금은 2016 년보다 변하는 속도가 더디지만, 비전공자가 느끼는 속도는 2016 년에 자바스크립트를 배우는 기분과 비슷할 것이다.

2. **웹 서비스의 전체적인 동작과 흐름은 백엔드/서버 개발자에 의해 개발된다.**
    a. 백엔드 개발자는 프론트엔드 개발자에 비해 커버해야 하는 범위가 넓다. 물론 백엔드 개발자가 더 우월하다는 것은 절대 아니다. 커버해야 하는 범위가 넓은 만큼, 더 넓은 시야로 다양한 것들을 고려하기 때문에 개발자로서 성장하기에 유리하다. (프론트엔드 개발자가 개발을 총괄하는 CTO 가 되는 경우는 드물다.)

3. **국비지원학원의 경우, 프론트엔드 기술들을 가르치는 코스가 없다.**
    a. 현실적으로 이 사실이 가장 큰 걸림돌이다. 프론트엔드 개발자는 채용 시장에서도 부족한 상황이고, 기술 변화가 빠르기 때문에 아직 프론트엔드 개발을 가르치는 국비지원학원 코스가 없다.

하지만 처음부터 프론트엔드 개발자의 길을 선택하는 것도 장점이 있다.

1. **현재 프론트엔드 개발자가 너무 부족하다.**
   a.

2020년 유망직종 상위 10개		
미국		
순위	직업명	만족도
1	프런트 엔드 엔지니어	3.9
2	자바 개발자	3.9
3	데이터 과학자	4
4	제품 매니저	3.8
5	데브옵스 엔지니어	3.9
6	데이터 엔지니어	3.9
7	소프트웨어 엔지니어	3.6
8	언어치료사	3.8
9	전략 매니저	4.3
10	비즈니스 개발 매니저	4

   b. GlassDoor 의 설문 조사 결과, 2020 년 유망직종 1 위가 프론트엔드 개발자로 뽑혔을 정도로 채용 시장에서 귀한 대접을 받고 있다.

2. **백엔드 개발자에 비해 학습 범위가 좁다.**
   a. 백엔드 개발자는 비즈니스 로직 개발과 서버와 데이터베이스 구성 등 담당하는 역할의 범위가 넓지만(회사마다 다름), 프론트엔드 개발자는 화면을 구성하고 동작하는 것에 집중할 수 있다.

웹 개발자 이외의 다른 분야는 직접 경험해보지 못했기 때문에 다루지 않았다. 본인이 임베디드 개발이나 시스템 개발 등, 다른 분야에 관심과 흥미가 있다면 도전해볼 것을 추천한다. 개발자의 길은 여러 갈래이지만, 정답은 없다. 각자의 관심과 흥미에 맞는 길을 가면 그 길이 옳은 길이 될 것이다.

Q. 다시 처음으로 돌아가서 개발 분야를 선택해야 한다면?

- 그래도 웹 백엔드 개발자로 도전할 것이다. 다른 분야에 비해서 All-round player 라고 생각하기 때문이다. 첫 시작을 웹 백엔드 개발자로 하는 것이 추후에 특정 분야에 진입하기에도 다른 분야의 개발자들보다 수월하다고 생각한다.

1.3. 올바른 개발 학습법 알기

개발자가 되기 위해 정해진 학습 방법은 없지만 올바른 방법은 있다. 올바른 개발 학습법을 설명하기 전에, 학습 방법과 관련하여 특히 문과 출신 비전공자가 반드시 알아야 할 것이 있다. 회사에서 전공자 출신 개발자분들이 문제를 해결하는 방식을 지켜보면서, 문과 출신에 체육을 전공한 나와는 조금 다른 점들을 발견했기 때문이다.

1.3.1. 사고방식 바꾸기

우선, 문과생들의 사고방식은 엔지니어의 사고방식과 다르다. 우리가 배운 학문은 사실 논술에 가깝다. 코에 걸면 코걸이, 귀에 걸면 귀걸이가 되는 내용이며 정해진 답이 없는 편이다. 때문에 어떤 문제를 해결함에 있어서, 제시되는 해결책들은 유일한 답이 아니기 때문에 대부분 두루뭉술하게 표현된다. (나쁘다는 것이 아니다. 이 세상 대부분 문제의 해결책은 두루뭉술하다)

그런데 이 '두루뭉술한' 문제 해결 방식이, 개발하며 마주하는 문제를 해결하는 데에는 아주 치명적인 단점이 된다. 나의 사례로 설명하자면, 개발하다가 문제가 발생했을 때 그 문제를 하나의 큰 덩어리로 보는 습관이 있었다. 1~2 시간을 구글링해보고 고민해봐도 해결이 안 되어 선배 개발자들에게 질문하면, 그분들은 우선 내가 맞이한 문제를 하나의 덩어리로 보는 것이 아니라, 여러 단계로 쪼갰다. 그리고 단계를 쪼개는 것만으로도 해결책이 발견된 경우가 70% 이상이었고, 단계를 나누어 하나씩 해결하다 보면 결국 단순하고 쉬운 문제였던 경우가 대부분이었다.

이와 같은 민폐(?)가 개발자가 된 이후 약 6 개월간 지속되었다. '다음에는 꼭 문제를 쪼개서 해결해야지' 다짐해봐도, 습관처럼 굳어진 두루뭉실한 사고방식 때문에 쉽게 고쳐지지 않았다. 그래서 문과 출신 비전공자들은 애초에 개발 학습을 문과 전공 공부하듯이 하면 안 된다.

개발 공부는 수학 공부하듯이 해야 한다. 문과 과목 공부는 주로 이론을 이해하고 암기하는 것인 반면, 수학 공부는 공식 암기보다 문제 풀이와 오답 분석이 중요하다.

개발도 그렇게 학습해야 한다. Java 문법을 배웠으면 직접 코드를 짜보고, 문제가 발생하면 문제의 원인을 분석하는 방식으로 학습해야 한다.

수학에서 답이 틀린 경우, 오답 분석을 통해 풀이 과정 중 어느 단계에서 풀이가 틀렸는지 찾는데, 이것이 개발에서의 디버깅과 같다. 개발을 하다 보면 업무 시간의 절반 정도는 이미 작성된 코드에서 발생한 문제를 찾고 해결하는 일(디버깅)이다.

개발자가 되고 난 후 느낀 것이지만, 개발자의 실력 차이는 디버깅 능력 차이에서 확연히 드러난다. 때문에 시작할 때부터 수학 공부하듯이 개발을 공부해서 엔지니어처럼 문제를 잘게 쪼개서 해결하는 연습을 해야 한다.

하지만 수학 공부하듯이 개발을 공부하라는 것은 나의 주관이 많이 개입된 방법이다. 사람마다 수학을 공부한 방법이 다를 수 있고, 고등학생 때 수학을 공부하지 않은 경우도 있기 때문이다. 개발자들의 다양한 견해를 참고하되, 본인에게 맞는 개발 학습법을 스스로 만들어가야 한다. 학습법을 배울 수 있었던 글들을 간추려보았다.

1.3.2. 반드시 읽어봐야 할 개발 학습법들

- 프로그래밍 학습 방법 – 박재성 님 영상
- 개발자가 공부로 살아남는 방법 – Evan Moon 님 블로그 글
- 학습에 실패한 이야기 – 남규진 님 블로그 글
- 나는 어떻게 공부했는가? – 조은 님 블로그 글
- 개발자의 평생공부 – 임백준 님 칼럼
- 프로그래머로서의 성장을 도왔던 태도들 – 안희종 님 블로그 글
- 프레임워크 공부를 멈춰라 – Jayce Park 님 번역 글
- 개발자가 실력을 향상시킬 방법은 OO 뿐이에요 – 김민태 님 인터뷰 영상
- 회사 밖에서 성장하기 – 권민재 님 발표 자료
- 개발자는 어떻게 성장해야 할까 – 강대명 님 발표 자료
- 어려운 것을 쉽게 배우는 방법 – 정도현 님 번역 글
- 내게 실용적이었던 프로그래밍 공부 방법들 – city7310 님 블로그 글
- 더 나은 개발자가 되는 8 가지 방법 – 박민우 님 번역 글

2. 학습하기

개발자가 되기로 결심하고 개발 분야도 정했으며, 올바른 개발 학습법도 알았다면, 이제 학습에 사용할 도구들을 알아야 한다. 개발자가 되기 위해 정해진 학습 도구는 없다. 책, 인강, 블로그 글, 학원, 그룹 스터디, 오픈 소스 등 모든 방법이 유효하다.

모든 방법이 유효하지만, 시간이 부족한 비전공자가 모든 방법을 시도하기는 쉽지 않다. 그렇다고 게임 캐릭터의 테크 트리를 정해주듯이 학습 도구를 정해줄 수는 없다. 개인마다 잘 맞는 도구와 방법이 다르기 때문이다. 학습 도구 선택과 집중에 참고할 수 있도록, 내가 경험한 학습 도구들과 각 도구의 장, 단점을 설명해본다.

2.1. 책

가장 먼저 접한 학습 도구는 책이었다. 개발 기술을 책으로 처음 접한 것은 아니었고, <프로그래머의 길, 멘토에게 묻다> 라는 개발자 성장 방법에 관한 책이었다. 이 글을 쓰면서 책의 목차를 다시 살펴보았는데, 깜짝 놀랐다. 내 스스로의 의지로 했다고 생각했고, 하길 잘했다고 생각했던 일들(무지를 드러내기, 배운 것들 기록하고 공유하기, 독서 목록 작성 등등)이 모두 이 책에 있는 내용이었기 때문이다.

개발 공부를 본격적으로 시작하기에 앞서 읽었던 이 책으로부터 영향을 받은 것이다. 그리고 그 영향은 개발자로 성장하는 데에 정말 큰 도움이 되었다. 이 책은 반드시 읽어보기를 추천한다. 개발을 전혀 모르는 상황에서는 임팩트가 없을 수 있지만, 개발자가 된 후 다시 살펴보니, 유익한 개발자 성장 방법들로 가득 찬 책이다.

2.1.1. 책의 장점

a. 책의 목차

책은 2 가지 장점이 있다. 첫 번째 장점은 책의 목차다. 비전공자가 개발자의 길에 도전하기 힘든 이유는, 무엇을 모르는지 모른다는 것이다. 그런데 기술 서적의 목차는 이 기술에서 중요한 내용을 목차를 통해 어림잡을 수 있게 해준다. 개발 기술은 저마다의 너비와 깊이가 있다. 혼자 공부하는 비전공자가 어떤 기술의 너비(범위)를 파악하기는 쉽지 않다. 그런데 책의 목차는 너비를 알려주기 때문에 우리는 얼마나 깊게 파고들지만 선택하면 된다.

b. 책은 정제된 문장으로 기술을 정의

어떤 기술을 학습할 때, 가장 중요한 것은 이 기술을 왜 사용하는지 아는 것이다. 이 기술을 왜 사용하고, 어떤 역할을 하는지 인지하지 못한 채, 단순히 문법을 익히고 예제를 따라 하는 것은 잘못된 방법이다.

내가 스프링 프레임워크를 공부할 때 그랬다. 이 기술을 왜 쓰는지 궁금해하지도 않았고, 그냥 남들이 다 하니까 따라 했었다. 그런데 면접에서 스프링 프레임워크를 왜 쓰는지 설명해달라는 질문을 받았다. 나는 말문이 막혔고 결국 그 면접에서 불합격했다. 면접에서만 중요한 것은 아니다. 개발자라는 직업을 선택한 순간부터 언제나 기술 사용의 이유를 스스로 되묻는 습관을 지녀야 한다.

개발 기술이 한 가지 역할만 하는 경우는 드물지만, 대표적인 용도는 한 가지로 정의될 수 있다. 그런데 책에는 기술의 용도가 정제된 문장으로 잘 정리되어있다. 모든 것이 낯선 초보자에게 이 정제된 문장은 등산로의 길 안내판과 같은 역할을 한다. 가는 길이 얼마나 험할지 모르겠지만, 이 길이 어디로 통하는지는 알 수 있는 것이다.

개발 기술 세계는 (다른 분야에 비해) 자유도가 너무나 높아서 무엇을 학습해야 할 지, 얼마나 학습해야 할지 감을 잡기 어려웠다. 칠흑같이 어두운 바다 한가운데 떠 있는 뗏목 같은 느낌이었다. 그래서 나는 책의 목차를 등대 삼아 학습했고, 낯선 개념들은 책에 설명된 문장을 여러 번 곱씹으면서 이해했다. 이 방법이 가장 효율적인 방법은 아니었다. 하지만 언제나 시간이 촉박했고, 주변에 아는 개발자가 없었던 나에게는 큰 도움이 된 학습 방법이었다.

2.1.2. 기술 서적 선택 순서

1. (개발을 처음 접하는 경우) 개발 언어 한 가지
2. 본인이 만들고 싶은 프로그램에 핵심이 되는 기술
3. 많은 개발자들이 추천하는 개발자 필독서
    a. 개발자 필독서 50 선
4. 원하는 회사에서 사용하는 기술, 혹은 그 회사에서 추천하는 기술 서적
5. 서점에서 펼쳐봤는데, 구미가 당기는 개발 서적
6. (번외) 내가 읽은+읽을 개발 서적 목록

2.2. 인터넷 강의

생활코딩과 인프런 덕분에 개발을 인터넷 강의로 학습하기 정말 수월해졌다. 앞서 언급했듯이 나는 생활코딩의 웹 어플리케이션 만들기 강의로 본격적인 개발 공부를 시작했다. 개발자가 된 이후에는 주로 인프런에서 필요한 강의를 구매해서 수강했다. 또한 유튜브에서 해외 개발자들의 강의 영상이나 MOOC 강의들을 수강하기도 했는데, 대부분 도움 되었다.

2.2.1. 인터넷 강의의 장단점

비전공자에게 인터넷 강의(이하 인강)가 좋은 점은 명확하다. **낯선 행위들**(개발 환경을 세팅, 코드 작성, 서버 구성, 배포 등등)**을 눈으로 확인할 수 있다는 점**이다.

개발을 글로 학습하는 것과 직접 보는 것은 큰 차이가 있다. 개발은 실제 행위이기 때문이다. 우리는 축구를 글로 배워서 시작하지 않았고, 키스를 글로 배워서 시도하지 않았다. 우리가 하는 실제 행위들은 어디선가 보고 무의식적으로 배운 것들이다.

비전공자에게 개발은 낯설고, 어색한 행위다. 개발을 처음 접할 때, 가장 쉬운 것을 하는데도 어렵게 느껴지는 이유는 개발이 낯설기 때문이다. 그런데 인강은 강사의 실제 행위를 목격할 수 있어서 무의식적으로, 혹은 의식적으로 배우는 것들이 생긴다.

내가 개발을 가장 많이 배웠다고 생각하는 때는 회사 선임 개발자가 내 컴퓨터를 잡고 문제 해결을 시도하는 것을 옆에서 지켜봤을 때다. 개발은 실제 행위이기 때문에, 잘하는 개발자가 어떻게 하는지 지켜보는 것만으로도 배우는 것들이 많다. 아직 회사에서 일하는 상황이 아니라면, 인강을 보고 따라하는 것이 큰 도움이 된다.

단점도 있다. 인강 강사가 진행한 내용을 눈으로만 이해하고 넘어가기 쉽다. 인강은 강사의 진행이 매끄럽기 때문에 쉬워 보이는 경향이 있다. 하지만 직접 해보기 전까지는 본인이 제대로 이해했고, 스스로 할 수 있는지 판단할 수 없다. 이 점을 유념하고 인강의 진도를 신경 쓰기보다, 느리더라도 배운 내용을 직접 해보면서 수강하는 것이 중요하다.

2.2.2. 추천하는 해외 강의

영어 듣기가 한국어 듣기의 80% 이상만큼 편하다면, 강의 선택폭이 훨씬 넓어진다. 하지만 그 정도 수준이 아니라면 과감하게 한국어 강의부터 듣는 것을 추천한다. 자칫하면 개발 공부가 아니라 영어 공부가 될 수 있기 때문이다. 만약 생소한 분야의 영어 듣기에 문제가 없다면, 아래의 학습 사이트를 활용하자.

- Udemy
- Udacity
- Coursera
- Edx
- FreeCodeCamp
- w3schools

위의 학습 사이트들을 둘러보면, 당장 결제하고 싶은 강의가 많이 보일 것이다. 욕심부리지 말고, 본인 수준에 맞는 강의 하나를 제대로 끝내는 것이 중요하다. 관심있는 언어, 분야의 강의를 검색해보고 리뷰까지 최대한 읽어본 후 강의를 수강하는 것이 좋다. 개발을 아예 처음 접하고, 다른 개발 강의를 들어본 적이 없다면 Udemy 의 The Web Developer Bootcamp 라는 강의를 추천한다. (수강한 지인들이 추천해줬다.)

나는 국비지원학원에 다니던 때에, 집에서 학원까지 왕복 3 시간이 소요되었기 때문에 버스와 지하철에서 휴대폰으로 CS50 라는 강의를 수강했다. 이 강의는 미국 하버드 대학교의 컴퓨터공학 수업 중 가장 쉬운 단계의 강의다. (보통 가장 첫 단계의 수업이 CS101 등 숫자 101 이 붙는데, 개발을 처음 접하는 학생들을 대상으로 하여 CS50 이다.)

이 강의는 교수의 말이 꽤 빠르지만, 강의 방식이 다채로워서 지루하지 않고, 컴퓨터 공학의 기초적인 부분들을 이해하기 쉽게 설명해줘서 하버드에서도 인기가 많은 강의다. 하지만 개발을 처음 접하는 비전공자가 이 강의에서 말하는 것들이 왜 중요한 것들인지 이해하기는 쉽지 않기 때문에, 가벼운 마음으로 수강하고 추후에 다시 수강하는 것을 추천한다. (나는 개발자가 된 후 다시 수강했을 때 더 많이 배웠다.)

현업의 개발자들도 인터넷 강의를 수강한다. 때문에 지금 단계에서는 듣고 싶은 강의가 많이 보인다고 조급해할 필요가 없다. 지금은 어떤 강의를 들어도 모두 새롭기 때문에 실력을 쌓아가는 기분에 취할 가능성이 높다. 많은 강의를 빠르게 수강하고는, 그 기분에 취해 개발 실력이 향상되었다고 착각하는 것이다. 강의 하나를 수강하더라도 제대로 이해하고, 코드를 직접 따라치면서 학습하기를 추천한다.

2.3. 블로그 글

개발 블로그를 운영하는 개발자들이 많다. 아니, 개발자라면 블로그를 운영해야 한다. 그 이유는 취업하기 파트에서 다룰 것이고, 학습 도구로써 블로그 글에 대해 논한다.

개발자는 구글링을 통해 기술의 사용법을 찾거나, 문제 해결 방법을 찾는다. 신입 개발자만 구글링하는 것이 아니다. 20 년 경력의 개발자도 검색하며 개발한다. 새로운 기술이 계속 나오고, 기존 기술도 계속 발전하기 때문이다. 구글링을 통해 개발자가 찾아보는 것은 크게 세 가지다.

1. 사용하려는 기술의 **공식 문서**
2. 문제 해결을 위한 **질문과 답변**
3. 개발자가 작성한 **개발 블로그 글**

2.3.1. 공식 문서

공식 문서는 해당 기술을 만든 주체가 제공하는 기술에 대한 공식적인 사용설명서다. 그런데 공식 문서는 해당 기술의 스펙과 기능을 나열하는 것이 목적인 문서다. 때문에 초보 개발자에게 친절하게 사용법을 설명하지 않으며, 주로 영어로 되어 있어 비전공자가 처음부터 접근하기는 쉽지 않다(특히 Java 공식 문서). 하지만 최근에는 Vue.js 공식 문서처럼 깔끔한 디자인의 친절한 사용 가이드를 제공하는 경우도 있다.

2.3.2. 질문과 답변

질문과 답변은 주로 stackoverflow.com 에 올라온 질문과 답변을 말한다. 개발을 하다 보면 문제가 발생한다. 이때, 문제의 원인을 알면 공식 문서를 찾아보면 되겠지만, 보통은 무엇이 문제인지 모르는 경우가 많다. 또는 문제의 원인을 알더라도 사용한 다른 기술들과 연관된 문제일 경우, 공식 문서만 봐서는 해결할 수 없다. 이때 가장 효율적인 해결 방법은, 나와 똑같은 문제를 이미 겪은 개발자가 올린 질문과 그에 대한 다른 개발자들의 답변을 보는 것이다.

본인이 맞이한 문제 상황 또는 에러 메시지로 구글에 검색하면, 자연스럽게 StackOverflow 에 올라온 질문과 답변이 검색 결과 상위에 뜰 것이다. 영어가 조금

부담스러울 수 있지만, 대부분 문장이 길지 않고 코드를 곁들이기 때문에 고등학생 수준의 독해 수준이면 문제없다.

개발자로 일하면 하루에도 수십 번 StackOverflow 에 방문하거나 직접 질문을 올리게 될 것이다. 내가 StackOverflow 를 사용하는 방법 [번역]이라는 블로그 글을 보면, StackOverflow 가 단순히 문제 해결 플랫폼이 아니라, 개발자로서 성장하기 위한 도구로써 사용된다는 것을 알 수 있다. 심지어 저자의 모국어는 영어가 아니다.

앞서 언급한 공식 문서와 StackOverflow 는 개발자로 살아가는 한 계속 찾게 될 것이다. 하지만 개발 공부를 처음 시작하는 비전공자가 공식 문서와 Stackoverflow 를 볼 일은 많지 않을 것이다. 공식 문서와 StackOverflow 는 개발을 하며 문제를 해결하는 과정에서 보는 경우가 많다. 개발 학습에는 블로그 글을 활용하는 것을 추천한다.

2.3.3. 개발 블로그 글

개발 블로그 글은 개발자가 블로그에 개발 기술을 정리하거나, 경험을 기록한 글들을 말한다. 이는 개발을 처음 접하는 비전공자에게 특히 유용하다. 이유는 아래와 같다.

a. 학습 주제를 제공해준다.
b. 이해하기 쉬운 설명 또는 시각화된 자료를 볼 수 있다.
c. 개발자들이 어느 수준까지 학습하는지 보며 자극받을 수 있다.
d. 회사 기술 블로그의 경우, 회사에서 사용하는 기술과 개발 문화를 알 수 있다.

a. 학습 주제를 제공해준다.

개발 학습을 처음 시작하면, 본인의 주된 학습은 잘 이어간다. 예를 들어 Java 를 주 언어로 선택했다면, Java 기본서 혹은 인터넷 강의를 수강하며 커리큘럼을 따라 열심히 학습할 것이다. 하지만 개발은 언어 하나, 프레임워크 하나로만 이루어진 것이 아니다. 개발자로 취업하려면 핵심 기술들은 물론이고, 개발 방법론이나 협업 도구, 학습 방법 등 다양한 주제의 학습을 병행해야 한다. 그런데 비전공자들은 다양한 학습 주제를 생각할 겨를도 없이 Java 하나에만 목을 맨다.

Java 하나에 집중하는 것은 좋다. 그런데 비전공자들이 개발 기술 학습을 RPG 게임 캐릭터가 스킬 배우는 것 같이 학습하는 것이 문제다. 게임 캐릭터가 스킬 배우듯, Java 기본서 한 권 보면 Java 는 자신이 습득한 스킬이 되고, 이제 다음 단계의 스킬을 배우면 된다고 생각하는 것이다.

물론 Java 같은 주 언어는 제대로 학습해야 한다. 하지만 Java 는 실전 경험 없이 마스터 할 수 있는 기술이 아니며, 실력에 따라 사용할 수 있는 수준도 달라지는 기술이다. 그러므로 비전공자는 Java 같은 주 언어 학습은 대략 3~5 년간 꾸준히 한다고 생각하고, 우선 이해 가능한 범위까지만 학습하면서 그 이외의 학습도 병행해야 한다.

이때, 개발 블로그가 큰 도움이 된다. 블로그 글은 주제가 있어서, 많이 접할수록 다양한 학습 주제를 알게 된다. 체육을 전공한 나는, 주변에 아는 개발자가 단 한 명도 없었다. 그래서 무엇을 공부해야 하는지 물어볼 사람이 내 주변에는 없었다. 그렇다고 마구잡이로 공부하고 싶지 않았고, 29 살의 결혼한 백수인 만큼 최대한 효율적으로 학습하고 싶었다.

결국 개발 블로그에서 그 답을 찾을 수 있었다. 좋은 회사에 다니는 신입 개발자들은 어떤 것들을 공부해왔는지, 유명한 개발자들은 어떤 개발 방법론을 이야기하는지, 개발자들이 쓰는 협업 도구는 어떤 것들이 있으며 어떻게 사용하는지 등등, 이 모든 정보가 개발자들의 블로그에 고스란히 담겨있었다.

b. 이해하기 쉬운 설명 또는 시각화된 자료를 볼 수 있다.

블로그 글은 개발자가 학습한 내용을 정리하기 위해서, 작성하면서 더 제대로 학습하기 위해서, 취업할 때 어필하기 위해서 작성한다. 또한, 독자들이 읽어주기를 기대하는 글이다. 때문에 독자가 이해하기 쉽게 설명하고, 이해를 돕는 시각 자료를 제공하는 경우가 많다. 이해력이 부족한 비전공자들에게 블로그 글은 좋은 학습 도구가 된다.

c. 개발자들이 어느 수준까지 학습하는지 보며 자극받을 수 있다.

전공자들은 학교에서 같이 수업을 듣는 친구들과 자연스럽게 비교된다. 더 잘하는 학생으로부터, 혹은 교수님이나 조교로부터 자극을 받을 수 있는 환경인 것이다. 하지만 비전공자들은 혼자 공부하는 경우가 많다. 때문에 본인이 어떤 수준인지

가늠하기 쉽지 않고, 더 잘하는 사람들로부터 자극받을 기회가 드물다. 이때 유용한 것이 개발 블로그 글이다.

개발자에게 자극은 중요하다. 성장 원동력이 되기 때문이다. 개발 세계는 넓고 깊은 바다와 같아서, 개인의 의지에 따라 원하는 만큼 멀리 나아가고, 원하는 만큼 깊이 파고들 수 있다. 블로그 글은 나보다 멀리 헤엄치는 개발자, 나보다 깊은 곳에서 헤엄치는 개발자를 볼 수 있는 기회다.

개발 학습에 있어 가장 많은 영감과 자극을 주신 분은 이동욱 님과 이종립 님이다. 이 두 분에 대한 이야기는 부록에서 다룰 예정이다. 이분들은 비전공자 출신이지만 실력으로 인정받는 분들이다. 이분들의 블로그 글을 보면 내가 무능력하게 느껴지기도 했지만, 지속적으로 자극을 받을 수 있어서 개발자로서의 성장에 정말 큰 도움이 되었다.

논외로, 내가 가장 처음으로 자극을 받았던 블로그 글은 Joshua 님의 2017 년 회고였다. 이 글을 읽고 개발자들이 연말 회고를 쓰는 문화가 있다는 사실을 알게 되었다. 그리고 2018 년부터는 나도 연말 회고를 쓰기 시작했다.

나는 개발 공부를 시작할 때부터 개발 블로그 글을 학습 도구로써 적극적으로 활용했다. 그래서 페이스북에 기존에 팔로우했던 페이지들을 모두 팔로우 취소하고, 개발 관련 페이지와 그룹들, 개발자들만 팔로우했다. 그 결과 페이스북에 접속하면 무조건 개발 관련 컨텐츠만 볼 수 있었고, 매일 타는 버스와 지하철에서 페이스북에 공유된 개발 블로그 글들을 읽었다.

개발 블로그 글 중에, 정말 유익했던 글들은 카테고리를 나누어 크롬 북마크에 저장했었다. 그런데 계속 유익한 글들을 쌓다 보니, 더 이상 크롬 북마크에서는 관리가 되지 않아 { 고퀄리티 개발 컨텐츠 모음 }이라는 깃헙 저장소를 만들었다. 이 저장소는 지금까지도 관리하고 있으며, 2 천 개 이상의 별을 받았다.

이 저장소의 운영 목적은 나의 학습에 있다. 개발 블로그 글들을 매일 읽고 그 중 유익했던 글들만 저장하고 있는데, 지금까지 저장된 글들만 약 1300 개다. 보통 찾아 읽은 블로그 글 중 60~70% 정도를 저장하기 때문에 찾아 읽은 블로그 글은 약 2100 개로

추정되고, 개발 중에 검색하면서 읽은 블로그 글까지 합치면 약 2 년간 최소 2500 개의 블로그 글을 읽은 것이다. 읽은 글의 개수가 중요하다는 것은 아니다. 그보다 매일 개발 블로그를 읽는 습관을 강조하고 싶다. 이 습관 덕분에 개발자 지인이 한 명도 없었지만, 스스로 다양하고 유익한 개발 지식을 꾸준히 쌓을 수 있었다.

d. 회사 기술 블로그의 경우, 회사에서 사용하는 기술과 개발 문화를 알 수 있다.
많은 회사들이 기술 블로그를 운영하고 있다. 기술 블로그를 통해 사내 개발자들이 문제를 해결한 경험이나 개발 문화 등을 공유하고, 채용 정보를 제공하기도 한다. 한마디로 회사 밖 개발자들을 대놓고 유혹하는 글들이다. 주요 회사들의 기술 블로그는 아래와 같다. 한 번씩 방문해보자.

1. LINE 기술 블로그
2. 카카오 기술 블로그
3. 네이버 기술 블로그
4. 우아한형제들 기술 블로그
5. 쿠팡 기술 블로그

기술 블로그에 올라가는 글은 보통 팀 내에서 검증을 거친 글들이다. 때문에 개발 블로그 글 중에는 가장 퀄리티가 높은 편이다. 게다가 실전 경험에 대한 내용이 많기 때문에 실무적으로 도움되는 내용들이 정말 많고, 개발자들이 어떻게 일하는지 간접적으로 경험할 수 있기 때문에 무조건 유익한 글이다.

그럼 이 블로그들을 매일 클릭해서 봐야 하는가? 아니다. Feedly 라는 RSS Reader 서비스에 기술 블로그들의 주소를 등록해두면, 새 글이 올라올 때마다 Feedly 에 올라온다. 매일 아침 출근길에 가장 먼저 Feedly 부터 접속하는 것이 나의 습관이다. 나는 총 137 개의 블로그를 구독하고 있다. 이렇게 구독해도 매일 새롭게 올라오는 글은 약 10 개이기 때문에 부담 없이 읽을 수 있다.

일하고 싶은 회사가 있으면 그 회사의 기술 블로그의 글들을 모두 읽어보는 것을 추천한다. 어려운 내용도 많지만, 모든 내용을 이해할 필요는 없다. 그 회사에서 어떤 기술들을 사용하는지 확인하고(채용 공고에는 없었던 기술들도 알 수 있다.), 개발 문화가 어떤지, 어떤 성향의 개발자들이 일하고 있는지 확인할 수 있어 유익하다.

2.4. 학원

학원은 다른 학습 도구와 다르게 정보가 한정적이고, 기회비용이 크기 때문에 선택에 더욱 신중해야 한다. 그런데 광고 때문에 좋은 학원을 선택하기가 쉽지 않다. 이 챕터에서는 학원을 선택할 때 고려해야 할 사항들과, 학원에서 학습한 경험에 대해 공유한다.

여기서 말하는 '학원'은 비정기적인 오프라인 강의도 포함된다. 요즘에는 학원이라는 개념보다 비정기적으로 개강하는 오프라인 강의들이 주를 이루고 있는데, 통칭해서 '학원'이라고 부르겠다.

주변에 아는 개발자가 없는 비전공자는 자연스럽게 학원으로 이끌린다. 나도 개발자 지인이 한 명도 없어서 국비지원학원에 다니며 개발을 공부했다. 국비지원학원은 고용노동부에서 교육비를 전액 지원해서 구직자에게 직업훈련을 제공하는 학원이다. 즉, 나랏돈으로 운영되는 사설 학원이다.

그런데 교육비만 공짜인 게 아니라, 생활비까지 제공된다. 2018 년 기준 매 월 30 만 원(취업성공패키지 진행 시 40 만원)을 받았다. 나는 이 생활비 때문에 국비지원학원을 선택했는데, 결론적으로는 후회했다.

2.4.1. 국비지원학원 다닌 것을 후회하는 이유

우선, 국비지원학원은 **강의의 질이 낮을 수밖에 없는 구조**다. 고용노동부는 돈(세금)을 쓰는 쪽이고, 돈(혜택)을 받는 쪽은 구직자들이다. 즉, 정부에서 쓰는 돈으로 구직자들이 직업 훈련을 받는 것이다. 그런데 실제 돈의 흐름은 다르다.

국비지원학원에 등록한 수강생 수 만큼, 학원으로 돈이 지급된다. 수강생의 손을 거치지 않는 것이다. 여기서 무엇이 간과되었는가? 바로 강의의 퀄리티다. 강의 퀄리티에 상관없이, 국비지원학원들은 수강생들을 최대한 많이 모아서 정해진 기간 동안 교육하면 돈을 버는 것이다. 그래서 구하기 쉬운(=실력이 부족한) 강사를 고용하고, 강의 퀄리티보다는 수강생들을 학원에 묶어두는 것에만 집중하는 것이다.

올바른 돈의 흐름은, 수강생들에게 먼저 교육비가 지급되어 수강생들이 학원에 직접 교육비를 지불하는 것이다. 그래야만 학원들이 수강생들에게 어필하기 위해서 강의 퀄리티에 신경 쓰고, 실력이 뛰어난 강사진을 고용할 것이다.

실제 소비자에게 돈(힘)이 주어지지 않고, 공급자에게 돈(힘)이 주어지면 제품(강의)의 퀄리티는 낮아질 수밖에 없다. 공산품을 배급하는 북한과 소비자가 직접 선택해서 소비하는 남한의 공산품 퀄리티 차이를 생각하면 이해하기 쉽다. 국비지원학원의 강의는 공산주의 국가에서 배급하는 공산품과 같다.

그런데 국비지원학원을 선택한 것은 결국 수강생이 아니냐고 반문할 수 있을 것이다. 맞다. 개발을 1 도 모르는 수강생들은 네이버 검색창에 '국비지원학원'을 검색해서 몇몇 국비지원학원 홈페이지에 접속해보거나, Okky.kr 에서 학원 광고 글을 보고 상담 예약을 하고, 국비지원학원을 '선택'해서 등록했을 것이다. 수강생이 선택한 것이 맞다. 그런데 문제는 수강생에게 주어지는 정보가 너무나 제한적이고, 학원이 마음에 들지 않아도 다른 학원으로 옮기기가 쉽지 않다는 것이다.

특이하게도 국비지원학원은 면접을 본다. 이 사람이 끝까지 수강할 사람인지 체크하는 것이다. 이 사람이 강의에 만족하든 말든, 정해진 기간까지 수강해야 고용노동부로부터 교육비를 제대로 받기 때문이다. 지금까지 언급한 문제들은 각 학원의 문제라기보다 구조적인 문제다. 누구나 국비지원학원 원장의 입장이 되면 똑같이 학원을 운영할 것이다. 더 유능한 강사를 고용하고, 더 좋은 강의를 제공할 이유가 없기 때문이다.

국비지원학원에 다닌 것을 후회한 두 번째 이유는, 취업 연계를 하기 위해 아무런 거리낌 없이 **수강생들을 열악한 환경의 SI 업체로 연결**하기 때문이다. SI 업체를 간단히 설명하자면, 프로젝트 발주를 받아서 제작해주는 회사다. 모든 SI 업체가 나쁜 것은 아니고, SI 업체에서 직접 일해본 것이 아니라 조심스럽지만, SI 개발자가 많이 모이는 Okky.kr 에서 SI 라고 검색해서 나오는 글들과 댓글들을 읽어보면 감이 잡힐 것이다. 검색하기 귀찮다면, 아래 3 개의 글을 읽어보자.

1. 후배 개발자에게
2. SI 개발 10 년차인데 코드 좀 봐주세요
3. 개발자의 삶 – SI 산업의 문제점 (2)

나의 경우, 6 개월의 교육 과정이 끝나기 약 1 개월 전부터 학원에서 취업 연계를 해주었다. 이력서를 써내면 학원에 채용공고를 보낸 회사들로 대신 이력서를 제출해주는 것이다. 그런데 연계해주는 회사들이 죄다 이름 없는 SI 업체들이었다.

나는 온갖 개발 블로그 글을 닥치는 대로 보면서 지냈기 때문에, SI 업체는 가면 안 된다는 사실을 이미 알고 있었다. 그런데 학원에서는 내 실력이나 가능성은 전혀 고려하지 않고, 개발자 채용을 힘들어하는 SI 업체들로 대충 취업시키려고 했다. 물론 유명한 회사로 취업 연계를 기대한 것은 아니었지만, 수강생들을 이렇게 무책임하게 취급하는 것을 보고 국비지원학원에 다닌 것을 후회했다.

사실 대부분의 국비지원학원 수강생들은 SI 업체에서 커리어를 시작한다. 실력 부족으로 어쩔 수 없는 측면도 있지만, 더 좋은 회사에서 일할 수 있는 잠재력 있는 수강생들까지도 무책임하게 SI 업체로 취업을 유도하기 때문이다.

2.4.2. 국비지원학원의 장점

내가 도중에 그만두지 않고 6 개월의 교육과정을 끝마친 이유는 국비지원학원의 장점도 있었기 때문이다. 경험한 사람마다 차이가 있겠지만, 내가 생각하는 국비지원학원의 장점은 다음과 같다.

우선, **지각과 출석 관리를 철저하게 해준다.** 물론 수강생을 위해서가 아니라 학원을 위한 일이다. 수강생의 지각과 결석이 많으면 제대로 수료하지 않은 것으로 판단하고 고용노동부에서 교육비를 덜 지급하기 때문이다. 하지만 철저한 출석 관리는 결국 수강생에게도 큰 도움이 된다.

수강생들은 보통 졸업생과 퇴사한 백수들이다. 때문에 넘쳐나는 시간을 스스로 관리해야 하는데, 생각보다 쉽지 않다. 학교와 회사에서 정해진 일정 속에 살다가 갑자기 24 시간이 본인에게 온전히 주어지면, 무언가 해야 한다는 압박감에 오히려 불안해진다. 게다가 개발 학습은 컴퓨터를 사용하기 때문에 방해되는 것들(게임, 유튜브, 쇼핑, 채팅 등)이 많아서, 독한 의지 없이는 시간 관리에 실패할 가능성이 크다.

나는 처가에 얹혀사는 한 가정의 가장으로서, 빨리 개발자로 취업해야 한다는 강력한 학습 의지가 있었다. 그런데도 주말에 집에서 스스로 10 시간 학습하기가 쉽지 않았는데, 오히려 학원에 다니는 평일에는 학원 스케쥴을 따라서 학습시간 12 시간을 쉽게 넘길 수 있었다. 학원에서 8 시간, 집에 와서 4 시간을 학습에 투자한 것이다.

개발을 처음 접하는 비전공자에게 요구되는 학습량은 무자비하다. 이 학습량은 효율적인 학습과는 별개로, 물리적으로 많은 시간이 요구된다. 개발은 암기하는 것이 아니라 실제 행위가 본질이기 때문이다.

핵심만 콕콕 집어주는 족집게 강의를 수강하더라도, 실력 향상을 위해서는 직접 코드를 작성하고, 컴파일하고, 디버깅하는 과정이 필수적이다. 때문에 특히 비전공자는 절대적인 양의 학습 시간을 확보해야 한다. 국비지원학원의 철저한 출석 관리는 이러한 학습 시간 확보에 도움이 되었다.

두 번째 장점은 **동기들이 생긴다는 것**이다. 이 동기들은 자신과 가장 비슷한 상황에 처한 사람들이라 공통점이 많다. 6 개월간 같은 공간에서 9 시부터 6 시까지 동고동락 하다 보면 자연스레 친해지고, 인연은 학원을 수료한 이후에도 이어진다. 개발자 지인이 한 명도 없었던 나는, 동기들의 존재가 여러모로 큰 힘이 되었다.

우선, 심리적 안정에 도움이 된다. 개발을 학습하다 보면, 막히는 부분이 많다. 혼자 학습하다 막히면 내가 막힌 부분이 어떤 난이도의 문제인지 감이 없어서 흔히 말하는 '삽질'을 많이 하게 된다. 삽질이 반복되면 지나치게 스트레스를 받아서 개발자의 길을 포기하고 싶어질 수 있다. 그런데 동기들도 옆에서 같이 삽질을 하고 있으면, 내가 맞이한 문제의 난이도를 대략 알 수 있고, 서로 도와가며 삽질을 조금이라도 줄일 수 있다. (물론 삽질은 성장의 밑거름이다.)

또한, 혼자 공부하다가 어려운 문제를 맞이했을 때는 **'나는 개발에 소질이 없는 건가?'** 라며 과도하게 좌절하기 쉽다. 그런데 옆에 똑같이 못 하는 동기들이 있으면(무조건 있다.), **'나만 못하는 건 아니네? ☺'** 라는 생각에 자신감을 잃지 않을 수 있다.

자신감을 잃지 않는 것은 개발자가 되려는 비전공자에게 굉장히 중요하다**.** 개발을 공부할 때에도 그렇지만, 특히 회사에 입사해서 다른 개발자들과 함께 일하다 보면, 그들에게 당연한 것이 나에게는 결핍되어 있을 때가 있다. 이 때 사실은 익숙하지 않을 뿐인 경우가 많은데, 자신의 자질 부족이라 오해하고 자책해서 자신감을 잃기 쉽다.

개발자가 되어 회사에서 일할 때는 비전공자와 전공자 상관없이 모두가 같은 개발자고, 합당한 실력과 잠재력을 인정받고 채용된 것이다. 부족한 부분은 당연히 개인적으로 학습해서 보완해야 하지만, 익숙하지 않아서 못하는 것을 자신의 자질 부족으로 오해하지 않아야 한다. (참고 - 비전공 개발자가 전공자보다 정말 불리할까?)

자신감을 잃고, '나는 비전공자라서 못한다'라고 스스로 못 박는 순간, '못하는 개발자'라는 포지션에 스스로를 가두는 것이다. 아직은 본인의 한계를 정해버리면 안 된다. 난관을 마주하더라도, '이 기술은 내가 익숙하지 않을 뿐이다.' 라는 마음가짐으로 자신감을 잃지 않고 학습을 통해 극복해야 한다. 개발자로서 내게 도움 되었던 마음가짐은 부록에서 자세히 다룰 예정이다.

세 번째 장점은 **팀 프로젝트를 할 수 있다**는 점이다. 취업하기 파트에서 다룰 예정이지만, 개발자는 결국 개발팀 팀원들과 협업하는 직업이다. 혼자서 모든 것을 만드는 슈퍼개발자도 있고 프리랜서들도 있지만, 기본적으로 개발자는 협업이 기본인 직업이다. 때문에 채용 면접에서 가장 주된 포인트는 '이 사람이 팀에 잘 녹아 들어서 협업할 수 있는 사람인가?' 이다. 때문에 팀 프로젝트 경험을 묻는 질문이 자주 등장하는데, 학원에서 진행한 팀 프로젝트 경험을 이야기할 수 있어서 좋다.

네 번째 장점은 **생활비를 받을 수 있다**는 점이다. 국비지원학원에 다니면, 고용노동부에서 한 달에 30~40 만 원의 생활비를 지원해준다. 적다면 적은 돈인데, 나는 이 생활비 때문에 국비지원학원을 선택했다. 처가살이하는 백수 유부남에게 생활비 30 만 원은 정말 큰 영향을 미쳤다. 이 글을 읽는 분들 중에도 한 달에 30 만 원을 받으면서 개발을 배울 수 있다는 사실을 가장 큰 장점으로 생각하는 분들이 계실 것이다. 개발을 배우는 동안 생활비 마련이 어렵다면, 국비지원학원은 좋은 선택지가 된다. (국비지원학원에 다니는 대신, 알바 같은 것은 하지 않는 것이 좋다. 개발 학습에만 전념해도 시간이 부족하다.)

2.4.3. 내가 다닌 학원

나는 쌍용교육센터 라는 국비지원학원에 다녔다. 강남과 강북에 지점이 있는데, 집은 강남 지점이 가까웠다. 그런데 신혼여행에서 돌아오는 일정과 가장 가까운 개강 반은 강북 지점에 있어서, 홍대입구역 근처의 강북캠퍼스에 있어서 수강했다. 내가 수강한 과정은 'Java 를 활용한 웹 개발자 양성 과정'이었고, 박용우 강사님이셨다.

a. 학원 선택 기준

나의 학원 선택기준은 가장 많은 사람이 추천하는 학원이었다. 국비지원학원을 실제 경험한 사람들이 어떤 학원을 추천하는지 알아보기 위해 Okky.kr 에서 '국비지원학원'이라는 키워드로 검색해서, 최근 2 년 치의 모든 게시글과 댓글들을 전부 읽었다. 그 결과, 2017 년 11 월 기준으로 쌍용교육센터와 비트교육센터 두 곳으로 좁혀졌다. 쌍용과 비트 둘 중 어디를 가도 비슷하겠다는 생각이 들었고, 개강 반의 일정이 가장 가까운 쌍용교육센터를 선택한 것이다.

다만, 검색과정에서 자바 기본서 중 가장 많이 팔리는 자바의 정석을 집필하신 남궁성 님께서 비트교육센터에서 6 개월 교육과정을 맡으셨다는 정보를 얻어서 수강하고 싶었으나, 이미 개강한 상태라 수강할 수 없었다. 위에서 국비지원학원의 강의 퀄리티에 대해 악평을 했지만, 모든 국비지원학원 강사님들의 강의 퀄리티가 낮다는 것은 아니다. 남궁성 님 같은 강사님들도 분명 곳곳에 계실 것이기 때문이다.

b. 교육과정 선택 기준

두 번째로, 나의 교육과정 선택 기준은 Java 에 집중하는 과정이었다. 국비지원학원들의 교육과정은 당시에 유행하는 기술을 살짝 끼워 넣는 경우가 많은데, 나는 미끼 기술 없이 Java 만 판매하는 'Java 를 활용한 웹 개발자 양성 과정'을 선택했다.

2017 년에는 빅데이터가 유행했기 때문에 'Java 기반의 빅데이터 전문가 양성과정'이 많았고, 2020 년에는 프론트엔드 개발자 수요가 많기 때문에 'Java 기반의 웹 프론트엔드 개발자 양성과정'과 같이, Java 를 기본으로 깔고 수강생들을 Hooking 하는 미끼 기술을 끼워서 교육과정을 구성한다. 프론트엔드 개발자는 Java 가 아닌 Javascript 를 기반으로 해야 한다. 그런데 Javascript 를 가르칠 강사를

(국비지원학원에서는) 구하기 힘들다 보니, Java 를 기반으로 프론트엔드 개발자를 양성한다는 어처구니없는 교육과정이 생기는 것이다. 비전공자들은 이런 사실을 모르고 요즘 핫한 '프론트엔드 개발자 양성'이라는 키워드를 보고 수강하게 된다.

c. 강사 선택 기준

교육과정과 강사는 학원에서 미리 구성해둔다. 때문에 강사를 먼저 선택하고 그 강사가 진행하는 교육과정을 수강하거나, 교육과정을 먼저 선택하고 강사의 평판을 알아보아야 한다. 나는 후자를 택했다. Java 에 집중하는 교육과정을 선택하고 강사님의 성함을 구글에 검색해서 강사님의 평판을 최대한 확인하기 위해 노력했다. 학원의 시설이나 교육과정보다도 강사님이 개발 학습에 가장 큰 영향을 미치기 때문이다.

강사님 성함으로 구글링하니 오, 캡틴! 나의 캡틴! 이라는 글을 발견할 수 있었다. 나중에 인문학도, 개발자 되다 라는 책을 쓰신 마르코 님이 나와 같은 학원에서 같은 강사님께 배우셨던 것이다. 글에서도 알 수 있듯이 박용우 강사님의 평판은 좋았고, 나는 교육과정에 바로 등록했다.

나는 '학원 선택-교육과정 선택-강사평판 확인' 순으로 진행했지만, 사실 가장 중요한 것은 강사님의 평판이었다. 우리 반은 도중에 그만두는 사람이 없었지만, 옆 반은 강사가 중간에 바뀌어서 그만두는 학생들도 많았다. 즉, 국비지원학원에서의 실패를 최소화하려면, 좋은 강사님을 찾아서 그분의 강의를 수강하는 것이 최선이다.

d. 내가 학원에서 실패한 것들

나는 학원에서 두 가지에 실패했다. 우선, 학원 교재에 의존하고 개인적으로 Java 기본서를 학습하지 않았다. 이 실패 덕분에 학원을 수료할 때까지도 Java 의 기본적인 몇몇 문법과 JVM 의 동작에 대해 알지 못했다. 어떤 언어를 배우더라도 배우는 주 언어의 기본서나 온라인 강의를 병행하기를 추천한다.

두 번째는 팀 프로젝트를 완성하지 못한 것이다. 마르코 님의 글에서 알 수 있듯이, 강사님은 진짜 사업이 될 프로젝트를 만들길 원하셨고, 나는 정말 사업으로 키워서 카카오 플랫폼에 팔 생각으로 프로젝트 팀장을 맡아서 진행했다. 결과는 게시판 하나도 제대로 못 만들고 끝났다. 팀 프로젝트는 완성을 목표로 가볍게 기획해야 한다.

2.4.4. 추천하는 학원들

만약 수중에 여윳돈이 있거나 주변의 도움을 받을 수 있다면, 사설 학원을 추천한다. 국비지원학원에 비해 금전적인 부담은 크지만, 더 현실적인 커리큘럼을 제공하고, 강사들의 실력이 더 뛰어난 편이다. 직접 수강해보지는 않았으므로 학원을 선택해 줄 수 없지만, 커리큘럼만 보았을 때 괜찮아 보이는 학원들을 추천해본다.

- CODE STATES
  - 미국의 코딩 부트캠프 시스템을 도입한 학원이다.
  - Pre 코스, 16 주 풀타임 코스, 온라인 파트타임 코스, 20 주 몰입 과정으로 나뉘어 있어, 본인의 상황에 맞는 코스를 선택 수강할 수 있다.
  - 컴공 기본 지식과 Git 협업 방법, 페어 프로그래밍 등이 커리큘럼에 포함되어 있어 유익해 보인다.

- 코드스쿼드
  - 인기가 많아 2020 년 초 기준, 2021 년에 개설되는 강의를 기다려야 한다.
  - 비전공자 지인이 이 학원을 수료한 후에 좋은 회사의 개발자로 취업했다.
  - 마스터즈코스의 경우, 컴공 기본지식을 4 주간 배우고 각자 선택한 모바일, 프론트엔드, 백엔드 코스로 나뉘어 4 주간 배운 후에 다시 모여서 16 주간 프로젝트를 진행하는 점이 유익해 보인다.

- 패스트캠퍼스 Python 백엔드 개발 스쿨
  - 패스트캠퍼스에서 진행하는 풀타임 강의다. (온라인+오프라인)
  - 많은 학원들이 Java 와 Spring 을 가르치는 반면, 활용 범위가 더 넓은 기술인 Python 과 Django 를 배울 수 있는 점이 특징이다. (아직 한국에서는 Java & Spring 개발자 수요가 더 많긴 하다.)
  - 오전 10 시부터 밤 10 시까지 학습을 관리해 주기 때문에 의지가 약한 경우 많은 도움이 될 것 같다.

- 스파르타 코딩 클럽
  - 현직 개발자들이 직접 가르치는 강의다.
  - 기간이 8 주라서 짧은 편이지만, 개인적인 학습이 뒷받침된다면 짧고 굵게 학습할 수 있는 커리큘럼으로 판단된다.
  - 강의 기간이 짧은 만큼 비용 부담이 적다.

2.5. 그룹 스터디

그룹 스터디는 다양한 분야에서 사용되는 학습법이다. 개발 학습에도 그룹 스터디는 도움이 될 수 있다. 같은 내용을 남들과 같이 공부하면, 더 자세하고 정확하게 학습할 수 있다. 하지만 그룹 스터디는 주로 오프라인에서 이루어지고, 시간과 장소가 정해지기 때문에 제대로 하지 않으면 시간 낭비가 될 수 있다. 양날의 칼인 것이다.

2.5.1. 내가 했던 그룹 스터디

내가 처음으로 했던 그룹 스터디는 컴공 전공 지식 스터디였다. 학원에 다니던 때, 매주 토요일 오전에 이 스터디에 참여했다. 나는 대학교에서 컴퓨터와 관련되 강의를 일절 들어보지 못했기 때문에 컴퓨터 공학 전공자들은 무엇을 배웠는지 언제나 궁금했고, 그런 것들을 모른다는 사실에 언제나 불안했다.

국비지원학원에서는 취업을 위한 기술들만 가르치기 때문에 컴공 기초를 배울 수 없었다. 무슨 일이든 기본기가 중요하다고 생각했기 때문에, 나는 컴공 전공 지식을 따로 학습하기로 마음먹었다. 여기서 말하는 컴공 전공 지식은 컴퓨터 구조, 운영체제, 자료구조, 알고리즘 등의 기초 과목들이다. 이 과목들은 혼자서 학습하기 지루하고 어렵다는 생각에 그룹 스터디를 활용했다.

주로 남궁성의 코드초보스터디 까페에 자주 방문하며 그룹 스터디 모집 글들을 읽었다. 어느 날 컴공 4 학년 학생이 주도하는 컴퓨터공학 기초 스터디 모집 글이 보여서 바로 신청하고 참여했다. 스터디를 주최자는 취업 시즌을 앞두고 기초 지식을 한 번 정리할 목적으로 스터디를 개설한 것이다. 컴공 기초 지식이 전무한 나 같은 비전공자에게는 아주 적합한 스터디였다.

총 4 명이 진행한 이 스터디는 운영체제와 자료구조 책의 챕터를 분배한 후, 일주일 간 학습해온 내용을 팀원들 앞에서 발표하는 방식으로 진행되었다. 그런데 내가 학습해서 다른 사람들에게 가르친 내용은 제대로 학습이 되었지만, 다른 사람이 학습해온 내용을 강의식으로 듣는 것은 별로 유익하지 않았다. 주체적으로 학습하지 않으면 결코 효과가 없었던 것이다.

개발자가 된 이후에는 사내 개발자들과 알고리즘 문제 풀이 스터디를 하고 있다. 오프라인으로 만나지는 않고, 매주 정해진 포인트의 문제들을 각자 LeetCode 에서 풀어서 제출하는 스터디다. 알고리즘 문제풀이는 취업이나 이직에도 도움이 되지만, 개발자의 기본기라는 생각으로 매일 조깅하듯이 꾸준히 푸는 것을 목적으로 하고 있다.

2.5.2. 그룹 스터디의 장단점

그룹 스터디의 가장 큰 장점은 남에게 설명할 수 있는 수준으로 학습하게 되는 것이다. '가르칠 때 가장 많이 배운다'는 말이 개발 학습에도 정확하게 적용된다. 컴공 기초 지식 스터디에 참여한 지 2 년이 지난 지금, 내가 발표했던 내용은 아직도 기억이 난다. 하지만 남이 발표했던 챕터는 주제조차 기억이 나지 않는다. 2 년 전에 학습한 내용을 기억하기는 쉽지 않다. 나는 기억력이 좋은 편이 아니고, 당장 지난주에 학습한 내용도 잊어버리는 일이 잦기 때문이다. 그런데도 2 년 전에 학습한 내용이 아직도 기억에 남는 것은 그만큼 완전한 학습을 했기 때문이다.

그룹 스터디에서 맡은 주제를 발표하기 위해서는, 발표 이후에 스터디원들의 질문에도 대비해야 한다. 스스로 예상 질문을 생각해보고, 그 질문에 대한 답을 구글링하며 대비하는 과정에서 완전한 학습이 이루어지는 것이다. 만약 혼자서 학습한다면, 내용을 읽어보고는 '아, 이해됐네.'하고 다음 챕터로 곧바로 넘어갔을 것이다. 누군가 질문하지 않을 것을 알기에, 스스로도 질문 없이 진도만 쭉쭉 빼는 것이다. 그 결과, 2 주 뒤면 학습한 내용은 기억에 남지 않는다. 하지만 그룹 스터디에서는 팀원들에게 떳떳하기 위해 더 철저하게 학습하게 된다.

단점은 명확하다. 버려지는 시간이 많다는 것이다. 스터디 팀원을 구하는 시간, 팀원들과 일정을 조율하는 시간, 스터디 장소로 왕복 이동 시간, 발표에 소질이 없는 팀원의 발표를 들어주는 시간, 스터디에서 먹을 간식을 구매하는 시간, 팀원들과 회식하는 시간, 잘 아는 내용을 다시 학습하는 시간 등. 버려지는 시간이 생각보다 많다.

때문에 시간이 촉박한 비전공자들은, 이 스터디가 시간을 쏟을 가치가 있는지 지속적으로 판단해야 한다. 물론 100%의 효율성을 가질 수는 없다. 하지만 스터디원들과 잡담하는 시간이 많거나, 참여자들의 열정이 식었다고 판단되거나, 혼자 하는 것이 더 잘 된다고 생각될 때에는 과감하게 스터디를 그만두어야 한다.

2.5.3. 그룹 스터디 관련 참고 자료

- 백기선 님 영상 - 개발자 질문, “스터디 운영 어떻게 해야하죠?”
- 이동욱 님 글 - 5-5. 학습 방법의 변화
- 권희정 님 글 - 같이 개발 공부를 잘하는 법, 입사 후에 더욱 발전하는 법
- 그룹 스터디 모집 글이 올라오는 커뮤니티
    - Okky.kr - 정기모임/스터디 게시판
    - 남궁성의 코드초보스터디 까페

2.6. TIL (Today I Learned)

TIL 저장소를 활용한 학습 방법에 대한 설명이다. 개발자가 아니라면(혹은 개발자 중에서도) TIL 이라는 단어가 생소할 것이다. 하지만 요즘의 주니어 개발자들은 TIL 저장소에 학습한 내용을 카테고리별로 정리하고, 다른 사람들에게 공개하고 있다.

챕터 **2.3. 블로그 글**에서 블로그 글을 작성하면서 학습하는 방법에 대해서는 일부러 언급하지 않았다. 이 TIL 과 다소 겹치는 부분이 있기 때문이다. TIL 저장소는 Today I Learned 의 줄임말이며, 자신이 오늘 배운 내용을 정리하는 저장소다.

2.6.1. 참고할 만한 TIL 저장소

백문이 불여일견이다. 아래의 TIL 저장소들을 둘러보면 어떤 것인지 감이 올 것이다.

- Cheese10Yun 님의 TIL 저장소
- HomoEfficio 님의 문제해결 저장소
- 김남준 님의 TIL 저장소
- 진유림 님의 TIL 저장소
- 이현주 님의 TIL 블로그
- 나의 TIL 저장소
- Jbee 님의 나는 어떻게 개발 공부를 했나, 2 편

2.6.2. TIL 저장소를 활용한 학습 방법

우선 GitHub 에 TIL 저장소를 만들거나, 개인 블로그가 있다면 블로그에 TIL 이라는 카테고리를 추가한다. 정해진 포맷은 없지만, 학습한 내용들을 카테고리별로 구분하는 것이 관리하기 편리하고 보기에도 깔끔하다. 포맷은 김남준 님의 TIL 저장소를 참고하는 것을 추천한다. (나도 남준님의 TIL 저장소를 참고해서 만들었다.)

만약 GitHub 에 가입되지 않은 상태라면 가입해야 한다. GitHub 저장소는 개발자의 포트폴리오라고 해도 무방할 정도로 중요한 저장소이며, 취업과 이직에도 큰 영향을

미치기 때문이다. 아이디는 앞으로 개발자로서 사용할 닉네임과 일치시키는 것이 좋다. GitHub 사용법은 구글링하면 친절한 설명이 많다.

TIL 저장소는 기본적으로 배운 내용들을 정리해서 기록해두는 저장소다. 학습한 내용이나, 알게 된 내용이 있으면 TIL 저장소에 마크다운(.md) 문서로 작성해서 기록한다. 한 번에 완벽하게 작성할 필요는 없다. 알게 된 내용을 이해한 만큼만 정리해서 작성해 두고, 추후에 다시 해당 주제에 대해 알게 되는 것들이 생길 때 기존에 작성해둔 문서에 내용을 추가하면 된다. 이처럼 다시 문서를 수정할 일이 발생하기 때문에, 카테고리를 나누어서 보기 좋게 관리하는 포맷이 필요한 것이다.

이해한 만큼만 작성하는 이유는, 배운 내용을 정리하는 일이 부담되거나 스트레스받는 일이 되면 안 되기 때문이다. 개발 실력과 경험에 따라 같은 기술을 보더라도 보이는 깊이가 다르다. 그러므로 아직은 문서를 완벽하게 정리해야 한다는 강박을 가질 필요가 없다. 즉, 큰 주제를 한 번에 쓰는 것보다 작은 주제를 많이 쓰는 것이 좋다.

하나라도 배운 것이 있어야 TIL 저장소에 글을 추가할 수 있기 때문에, 이 저장소는 존재만으로도 학습을 유도한다. 자신의 글로 알게 된 내용을 정리하는 것은 개발자에게 가장 효율적인 성장 방법 중 하나다. TIL 에 작성한 내용은 추후에 개인 블로그에 다시 정리해서 올릴 수 있고, TIL 저장소 자체가 포트폴리오가 되기도 한다. 취업하기 파트에서 다시 설명하겠지만, 면접관들이 비전공자에게 기대하는 것은 '성장 가능성'이다. 그리고 성장 가능성을 보여줄 몇 안 되는 아이템 중 하나가 바로 꾸준하게 작성된 TIL 저장소다.

3. 취업하기

3.1. 이력서

보통 채용의 서류전형에서는 이력서와 자기소개서가 매우 중요하다. 평가할 잣대가 이력서와 자기소개서뿐이기 때문이다. 하지만 개발자는 기술자이기 때문에 이력서와 자기소개서보다 포트폴리오(경력기술서)와 깃헙 코드가 더 중요하다.

게다가 요즘에는 이력서와 자기소개서를 받지 않고, 프로그래머스 같은 플랫폼을 통해 과제부터 받은 후에 코드 퀄리티를 확인하고 면접을 진행하는 회사들이 많아졌다. 점점 이력서와 자기소개서의 중요도는 떨어지고, 진짜 코딩 실력을 먼저 검증하는 추세다. (참고 – 이직하며 느낀 2020 년 1/4 분기 개발자 채용 과정)

자기소개서는 사라지는 추세이기 때문에, 이 챕터에서는 이력서에 쓰지 말아야 할 것들만 언급할 것이다. 쓰지 말아야 할 것들만 쓰지 않아도 평균 이상의 이력서가 된다.

3.1.1. 망한 이력서의 표본 (feat. 내가 실제로 제출한 이력서)

아래의 이력서는 내가 개발자로서 첫 취업에 도전할 때 실제로 제출했던 이력서다. 잘 썼다고 공개하는 이력서가 아님을 명심해야 한다. 망한 이력서의 표본이다. 면접에 들어갈 때마다 이력서 지적을 받았다. 덕분에 개발자 채용 면접관들은 어떤 이력서를 원하는지 알 수 있었다. 아래의 이력서가 왜 망했는지 하나씩 설명하겠다.

a. 사진이 너무 크다.
스타트업 면접에서 사진 사이즈를 지적받았다. 부실한 내용을 가리기 위해서 사진을 크게 넣은 것이냐는 질문을 받았다. 정확한 눈썰미였다. 쓸 내용이 부족해서 사진을 크게 넣었는데, 오히려 역효과를 냈다. 내용이 부실하다고 사진을 크게 넣을 필요는 없다. 개발자는 영업 직무가 아니다. 외형적인 것보다 실력이 중요하다.

b. 프로젝트 설명이 부실하다.
신입 개발자 이력서에서 가장 중요한 내용은 기술 경험이다. 이직의 경우 개발자로서 일한 경력이 레퍼런스가 되지만, 신입 개발자의 경우(게다가 비전공자라면) 가장 영향력 있는 레퍼런스는 프로젝트 경험이다. 기술경력서를 따로 제출한 경우가 아니라면, 프로젝트에 대한 설명을 가장 공들여야 한다. 길게 쓰라는 것이 아니다.

면접관들에게 충분한 정보를 제공하되, 가독성을 유지해야 한다. 내 이력서의 경우, 가독성만 중요시 한 바람에 충분한 정보를 제공하지 못했고, 질문 폭탄을 맞았다. 만약 프로젝트에 자신 있다면, 간결하게 작성해서 면접관들의 질문을 유도하는 전략도 나쁘지 않다. (나의 경우, 미완성 프로젝트였기 때문에 질문받기가 두려웠다.)

한

Back-End Developer

Telephone 010
Email rhe
Birth 199
Address 경기

간략한 소개

만들고 싶던 서비스의 웹사이트를
외주로 제작한 경험이 있습니다.
이 때 프로그래밍을 처음 접했고,
개발의 재미에 푹 빠져서
말 그대로 꿈에서도 코드를 짭니다.

새로운 지식을 습득하고 적용하는
과정에서 행복을 느낍니다.
그리고 습득한 지식을 카테고리화 하여
정리하는 습관이 있습니다.

Technical Skills

Java ●●●○○
HTML/CSS ●●●○○
Javascript ●●●○○
SQL ●●●○○
Linux ●●●○○
AWS ●●●○○
Git/Github ●●●●○

프로젝트

< Haru Sketch 하루스케치 >
역할: 개인 프로젝트
사용 기술: Java, Spring Hibernate, Javascript, Mysql, AWS, Linux Ubuntu, Github(GitKraken)
기간: 2018.04 ~ 현재
설명: 수공예 작가의 포트폴리오 겸 작품 판매 쇼핑몰

< Soin 소인 >
역할: 팀장(팀원7명), 견적요청 게시판, Header 디자인
사용 기술: Java, JSP, Javascript, Oracle DBMS, Github
기간: 2018.01 ~2018.03
설명: 소규모 인테리어 시공 중개플랫폼 웹사이트

교육 및 학력

Java를 활용한 웹 개발자 양성과정 2017.12-2018.05
쌍용 교육센터

사범대학 체육교육 전공 2009.03-2017.02

업무관련 역량

Java, Spring, Hibernate, Mysql, Css에 자신 있고, 개인 프로젝트를 통해 지속적으로 공부 중입니다. Linux는 다루어 보았지만 아직 손에 익지 않아서 숙달이 필요합니다. AWS는 Linux Ubuntu 인스턴스를 생성하여 Mysql로 데이터베이스를 연동해보았습니다. Git은 Git Kraken으로 사용 중이고, SQL은 Join에 대한 경험이 부족합니다.

미국 어학연수 경험과 무역업에 종사하며 영문 서류들을 다룬 경험으로 인하여 영어 사용에 두려움이 없습니다. 때문에 기술 관련 Document와 Stackoverflow 등에서 새로운 프로그래밍 지식과 문제해결 방법들을 능동적으로 습득할 수 있습니다.

경력사항

2017.01-2017.07

- 전 세계에 오픈된 선박과 벌크 화물을 매칭시켜 운항, 운임수익 추구
- 선주, 화주, 선장, 선주/화주 대리인, 브로커와 소통하며 선박 운항.

2015.07-2015.08

- 스포츠 의류 브랜드 | 브랜드마케팅 전략 제안

c. 왼쪽 중간의 간략한 소개는 불필요했다.

왼쪽 중간 공간이 애매하게 남는 것 같아 간략한 소개를 넣었는데, 이력서의 구성을 해치는 역할을 했고, 내용도 어정쩡했다. 게다가 면접관은 내가 외주를 맡긴 것이 아니라, 외주를 받아서 개발한 것으로 오해했다. 면접관에게 내용을 정정하는 설명을 했을 때, 면접관의 실망한 눈빛은 아직도 기억에 남는다.

나는 이 이력서의 포맷이 깔끔하다는 생각에 선택했는데, 깔끔한 포맷이 오히려 이력서의 구성을 망쳤다. 양쪽으로 나뉘는 디자인은 일관된 흐름이 아니기 때문에 집중도가 떨어지고, 구성이 애매해진다. 기본적인 단면 포맷을 사용하자.

d. Technical Skills 부분은 최악이었다.

기술 숙련도를 게이지로 표현하는 것은 최악의 선택이다. 실제 실력은 동그라미 5 개 중에 1 개도 채우면 안 되는 실력이었기 때문에 면접관들을 자극했다. 신입 개발자가 생각하는 숙련도의 기준과 경력 10 년 이상의 면접관들이 생각하는 숙련도의 기준은 완전히 다르기 때문이다. 자신감을 갖는 것은 좋다. 하지만 기술 숙련도를 자신감과 연결시키면 안된다.

실제로 면접관 한 분은 내가 동그라미 3 개로 표시한 기술들 각각에 대해 한 가지씩 질문했다. 나는 6 개의 질문 중 단 1 개도 답변하지 못했고, 그 면접관은 각 질문이 동그라미 2 개에 해당하는 수준의 질문이었다고 했다. 결과는 불합격이었다.

현재 기술 숙련도가 낮으면 낮다고 솔직하게 표현하고, 앞으로 어떻게 높일 것인지, 지금까지 어떤 노력을 해왔는지 어필하면 된다. 비전공자에게 높은 기술 숙련도를 요구하는 신입 개발자 채용 면접은 없다. 면접관들이 비전공자에게 면접 기회를 주는 이유는 잠재력을 보기 위한 것이다. 현재 실력이 개판이라는 것은 모든 면접관이 안다. 어설프게 포장하면 무조건 역효과다 .

현재 재직중인 회사에 지원할 때, 자기소개서 문항 중에 세 가지 기술의 숙련도를 상중하로 표시하고 설명하라는 문항이 있었다. 나는 솔직하게 하, 하, 하로 표시했고, 숙련도가 부족한 이유와 앞으로의 학습 계획을 작성했다. 결국 면접에서 이와 관련한 질문을 받지 않았고, 합격했다.

3.2. 포트폴리오

개발자, 디자이너, 건축가 등 무언가를 만드는 기술직은 포트폴리오가 매우 중요한 평가 기준이 된다. 어떤 제품과 서비스를 어떤 기술을 사용해서 만들었는지, 포트폴리오(혹은 경력기술서)를 통해 어필해야 한다. 전공자들은 학교에서 과제로 제출한 프로젝트라도 써먹을 수 있지만, 대학을 졸업한 비전공자들은 포트폴리오에 작성할 내용이 없는 경우가 많다.

비전공자들은 채용 시장에 뛰어들기 전에 미리 포트폴리오를 준비해야 한다.
포트폴리오를 준비하지 않은 상태로 개발자 취업 시장에 뛰어들면, 지원할 수 있는 회사의 폭이 확 줄어들기 때문이다. 포트폴리오를 거창한 것으로 생각할 필요 없다. 평소에 학습한 것들을 잘 정리해서 공개하고, 만들고 싶은 (가벼운) 서비스를 만들어보는 것이 전부다.

이 챕터에서는 비전공자가 개발 공부를 시작할 때부터 꾸준히 하면 포트폴리오에 바로 써먹을 수 있는 활동들을 설명하겠다.

3.2.1. 토이 프로젝트

축구 선수가 이론만 학습하지 않고 연습게임을 하면서 실력을 갈고닦듯이, 개발자도 토이 프로젝트를 만들면서 실력을 키울 수 있다. 토이 프로젝트는 본인이 만들고 싶은 제품이나 서비스를 만드는 것이다. 만드는 주체가 개인이든 팀이든 상관없지만, 팀 프로젝트는 따로 설명할 것이기 때문에 여기서는 개인 프로젝트로 이해하면 된다.

개인 프로젝트는 포트폴리오에서 가장 영향력이 큰 아이템이다. 때문에 개발 학습을 시작할 때부터 토이 프로젝트로 무엇을 만들고 싶은지 계속 고민하는 것이 좋다. 단, 아래의 기준을 지키면 실패할 가능성을 줄일 수 있다.

a. 혼자 만들기에 부담스럽지 않은 규모의 주제를 택한다.
b. 주변 사람들이 겪는 사소한 불편함을 해소해주는 서비스를 만든다.
c. 본인이 일하고 싶은 분야에서 많이 사용되는 기술을 사용한다.

a. 혼자 만들기에 부담스럽지 않은 규모의 주제를 택한다.

토이 프로젝트의 아이디어를 떠올리다 보면, 자연스럽게 기업에서 만들 법한 서비스를 기획하는 경우가 있다. 위험한 생각이다. 특히 개발 경험이 부족한 상태에서는 최대한 보수적인 기획을 해야 한다. 사용자가 발생하면 좋겠지만, 아무도 안 쓸 것 같은 서비스라도 혼자 만들기에 부담스럽지 않은 것은 중요하다. 혼자 만들다가 포기하면, 사용자가 한 명도 없는 서비스보다 못한 결과이기 때문이다.

아직 국비지원학원에 다니던 시절, 나는 프리랜서 디자이너로 일하는 와이프를 위해 '프리랜서 디자이너들을 위한 포트폴리오 공유 플랫폼'을 토이 프로젝트로 만들려고 했다. 주제만 보면 큰 규모가 아닌 것 같지만, 결제 기능과 온갖 잡다한 기능들을 모두 기획했었다. 그 결과는? 4 주 동안 설계만 하다가 내가 할 수 있는 규모의 서비스가 아니라는 것을 깨닫고 포기했다. 토이 프로젝트는 말 그대로 장난감 같은 프로젝트를 만드는 것임을 명심해야 한다.

b. 주변 사람들이 겪는 사소한 불편함을 해소해주는 서비스를 만든다.

주변 사람들이 겪는 사소한 불편함 중에는 의외로 프로그래밍으로 쉽게 해결되는 문제들이 있다. 주로 컴퓨터가 잘하는 일을 사람이 직접 할 때 주로 발견되는데, 이러한 불편함을 해소해주는 프로그램을 만들면 주변 사람들이 서비스를 사용해보기 때문에 별다른 홍보 없이 사용자를 획득할 수 있다. 주변 사람들이 신기해하고 잘 사용하기 시작하면, 밤을 새워서라도 더 많은 기능을 개발하게 되는 자신을 발견할 것이다. 즉, 주변 사람들로부터 피드백을 받을 수 있기 때문에 동기부여가 된다.

내가 처음으로 만들었던 토이 프로젝트는, 이전 회사 동기들이 겪는 사소한 불편함을 해소해주는 간단한 챗봇이었다. 그 회사에서는 매일 오전 '간편식'이라는 도시락을 선착순으로 신청할 수 있는데, 간편식을 신청하면 사내 식당에서 점심을 먹을 수 없었다. 그래서 간편식을 신청했지만 사내 식당을 이용하려는 동기들과, 간편식을 신청하지 못했지만 간편식을 먹고 싶은 동기들 간에 간편식 거래가 매일 아침 단톡방에서 이루어지고 있었다. 그런데 일반 대화와 간편식 거래 대화가 섞이다 보니 거래자들은 대화 내용을 위로 올리면서 다른 거래자를 찾아야 하는 불편을 겪고 있었다. 나는 이를 해소해주기 위해 '간편식 거래봇'이라는 챗봇을 만들어서, 간단한 명령어를 입력하면 간편식을 누가 사고파는 지 한눈에 파악할 수 있게 했다. 동기들은 신기해했고, 나는 피드백을 받고 신나서 밤새 여러 가지 기능을 추가하곤 했다.

c. 본인이 일하고 싶은 분야에서 많이 사용되는 기술을 사용한다.
물론 다양한 기술을 사용해보는 경험은 좋지만, 시간이 부족한 늦깎이 비전공자는 가고 싶은 회사와 팀에서 어떤 기술을 사용하는지 확인하고, 애초에 그 팀에서 사용하는 기술들을 사용하는 것을 추천한다. 채용공고를 보면 각 팀마다 사용하는 기술을 파악할 수 있다. '우리 팀에서 사용하는 기술들'이라고 직접 명시하는 경우도 있지만, 주로 '지원 자격'과 '우대사항'에 적힌 기술들이 해당 팀에서 사용하는 기술로 이해하면 된다.

이 점은 생각보다 중요하다. 비전공자가 6 개월 정도 학습한 후에 채용 시장에 뛰어든다고 가정했을 때, 실제로 프로젝트 경험에 쏟을 수 있는 시간은 많아야 2~3 개월에 불과할 것이다. 그 기간 동안 팀 프로젝트를 우선시한다고 가정했을 때, 토이 프로젝트에 투입할 수 있는 시간은 더 줄어든다. 이때 지원할 팀에서 사용하는 기술로 토이 프로젝트를 만들어 두는 것이 채용에는 훨씬 유리하다. (장기적인 관점의 개발자 성장 측면에서는 정답이 없다.)

공부할 시간도 부족한데 토이 프로젝트를 왜 해야 하는지 궁금하다면, 코딩덕후 라는 토이 프로젝트를 운영 중인 권용근님의 프로젝트가 장난이야?! 라는 글을 읽어보자. (Java 개발자가 될 예정이라면, 앞으로 이분의 글이나 발표를 볼 기회가 많을 것이다.)

3.2.2. 팀 프로젝트

회사에서 개발자로 일한다는 것은, 팀에 소속되어 다른 개발자들과 함께 일한다는 것이다. 때문에 채용 과정에서도 협업 능력을 중요하게 평가한다. 프로젝트 경험과 함께 협업 능력까지 어필할 수 있는 것이 바로 팀 프로젝트다.

전공자들은 학교에서 팀 프로젝트를 여러 번 수행했을 것이고, 마음이 맞는 학우들과 함께 팀을 짜서 취업용 팀 프로젝트를 만드는 경우도 많다고 한다. 그에 반해 비전공자들은 팀 프로젝트 경험을 갖기가 쉽지 않지만, 아래 세 가지 방법을 소개한다.

- 그룹 스터디 팀원들과 팀 프로젝트 진행하기
- 학원 동기들과 팀 프로젝트 진행하기
- IT 동아리에 가입해서 팀 프로젝트 진행하기

a. 그룹 스터디 팀원들과 팀 프로젝트 진행하기

만약 참여 중인 그룹 스터디가 없다면, 그룹 스터디를 모집하는 카페와 커뮤니티에서 팀 프로젝트를 함께 할 사람들을 모집하거나 참여하는 방법이 있다. 그런데 모르는 사람들과 만나서 팀 프로젝트를 지속하기는 쉽지 않기 때문에, 처음에는 그룹 스터디로 모였다가 목표로 했던 책이나 강의가 끝나면 자연스럽게 팀 프로젝트를 함께 진행하는 것을 추천한다. 그룹 스터디 과정에서 서로에 대해 신뢰가 쌓인 상태로 진행할 수 있으며, 그룹 스터디에서 학습했던 기술을 사용한다면 서로 간의 기술 격차가 줄어들어 팀 프로젝트를 지속하기 수월해진다.

b. 학원 동기들과 팀 프로젝트 진행하기

두 번째 방법은 학원의 같은 반 동기들과 팀 프로젝트를 진행하는 것이다.
국비지원학원의 경우 보통 팀 프로젝트를 위해 팀을 구성해주는 편인데, 혹시나 팀을 구성해주지 않는 학원에 다니더라도 같은 반 동기 중 열심히 하는 동기들과 팀 프로젝트를 진행하는 것을 추천한다. 학원 수강생들은 기술 베이스도 비슷하고, 취업을 해야 하는 공통의 목표도 있기 때문에 팀 프로젝트를 진행하기에 최적의 조건이다.

다만, 열심히 노력하는 동기들과 팀을 구성하는 것이 좋다. 대학교 수업처럼 학점이 걸린 문제가 아니라 취업이 걸린 문제이기 때문이다. 프리라이더나 게으른 사람을 억지로 끌고 갈 필요가 없다. 정말 열심히 하려는 동기들과 모여서 완성을 목표로 간단하지만 포트폴리오로 사용할 수 있는 수준의 프로젝트를 만들면 된다.

c. IT 동아리에 가입해서 팀 프로젝트 진행하기

세 번째 방법은 디프만 같은 IT 동아리에 가입해서 팀 프로젝트를 경험하는 것이다.
디프만의 경우 직장인도 가입할 수 있으며, 디자이너들과 개발자들이 한 팀이 되어 프로젝트를 진행하는 동아리다. 내가 직접 경험해보지는 않았지만, 디프만 페이스북에 공개하는 프로젝트 결과물들을 보면, 학원이나 커뮤니티에서 만나서 만든 팀 프로젝트보다 퀄리티가 뛰어난 경우를 많이 볼 수 있어서 추천한다.

실제 개발자로 일할 때는 기획자, 디자이너 등 다른 직무의 사람들과 협업한다. 때문에 디자이너와 함께 프로젝트를 진행하는 것이 개발자들끼리 모여서 프로젝트를 진행하는 것보다 훨씬 실무에 가까운 경험이 될 수 있다.

3.2.3. 개발 블로그

개발자의 길에 뛰어든 이후 가장 놀랐던 것은, 개발자들이 블로그를 통해 개발 지식을 공유하는 문화였다. 내가 몸담았던 해운업계에서는 업무 관련 지식을 공유하는 블로그를 정말 찾기 힘들었다. 특히 벌크선 관련 업무를 수행하기 위해 알아야 하는 지식이 광범위하고 깊이도 있었는데, 온라인에서 지식을 공유하는 사람은 손에 꼽았다. 때문에 업무를 위한 지식을 사내에서 도제식으로만 배울 수 있었는데, 개발자 세계에서는 너도나도 아는 지식을 블로그에 공유하고 있어서 놀라웠다.

이처럼 유독 개발 블로그가 많은 이유는 개발 블로그도 개발자의 포트폴리오로 사용되기 때문이다. 비전공자라면 특히 개발 블로그를 반드시 만들어서 포트폴리오로 활용할 것을 추천한다. 이유는 단순하다. 채용하는 회사에서 원한다.

위의 이미지는 개발자들에게 인기 많은 어느 IT 회사의 입사 지원 페이지다. (요즘에는 구구절절한 자기소개 없이 간단한 개인 정보와 포트폴리오, 개인 블로그와 GitHub 링크를 제출하는 것으로 지원서 작성이 끝나는 경우가 많다) 빨간색으로 표시한 부분을 보자. 선택사항이지만 개인 블로그나 GitHub 링크를 요구하고 있다. 비슷한 이력서와 포트폴리오를 갖추었다면, 개인 블로그와 GitHub 링크를 추가한 지원자가 서류 전형을 통과할 가능성이 훨씬 높다.

비전공자들은 반드시 개발 블로그와 GitHub 링크를 추가해서 채용담당자 눈에 띄어야 한다. 비전공자들은 전공자들에 비해 나은 구석을 찾기가 힘든데, 이러한 노력마저 없으면 채용담당자는 무조건 전공자를 선택한다. (합리적인 선택이다)

단순히 눈에 띄기 위한 용도는 아니다. 현재 실력이 부족하더라도 잠재력을 어필할 수 있고, 블로그 글을 통해 개발 실력 외의 협업 능력이나 성실성 등 다양한 가치를 채용담당자가 눈여겨보기 때문이다. 물론 개인 블로그와 GitHub 링크를 요구하지 않는 회사들도 있지만, 내가 이직 과정에서 직접 경험한 바에 의하면 좋은 회사들의 90% 이상은 개인 블로그와 GitHub 링크를 선택사항으로 요구했다.

Q. 네이버 블로그로 개발 블로그를 만들면 되나?
절대 네이버 블로그로 시작하면 안 된다. 네이버 블로그는 네이버에서 검색되는 것에 특화된 블로그이기 때문에 제약이 많다. 특히 개발 블로그 작성에 주로 사용되는 MarkDown 문법을 사용할 수 없다. (참고 - 네이버 블로그에서 마크다운 기반 글 올리기)

MarkDown 문법으로 작성된 글은 HTML 등 다른 문서 형태로 변환이 가능하기 때문에 개발자는 어떤 문서를 작성하더라도 마크다운 문법으로 작성하는 것이 기본이다. 그런데 마크다운 문법을 사용할 수 없는 것은 치명적인 단점이다.

게다가 개발자들은 개발 지식을 네이버가 아니라 구글에서 검색한다. 네이버를 싫어해서 그런 것은 아니다. 네이버에서 일하는 개발자들도 개발 지식 검색은 구글에서 한다. 때문에 네이버에서 검색되는 것에 특화된 네이버 블로그를 사용할 이유가 없다. (네이버 블로그는 구글 애드센스를 붙일 수도 없다)

Q. 그럼 블로그를 어디서 만드나?

처음에는 Tistory(티스토리)에서 블로그를 만드는 것을 추천한다. 특히 최근에 업데이트되어 디자인과 사용성이 더 좋아졌다. Java 개발자에게 가장 유명한 블로그인 이동욱 님의 블로그도 티스토리 블로그다. 네이버 블로그와 달리 마크다운 문법을 쉽게 적용할 수 있고, 블로그 기능에 대한 자유도가 훨씬 높다. 네이버 블로그에 비해 구글에서 검색도 잘된다. (구글 애드센스를 달 수 있어서 구글에서는 당연히 네이버 블로그보다 상위에 노출되는 것으로 생각된다) 때문에 아직 Git 을 사용할 수 없는 상태에서는 티스토리 블로그가 최선의 선택이다.

Git 을 사용할 수 있게 된 이후에는 GitHub Page 와 Static Site Generator(정적 웹사이트 생성기)를 이용한 GitHub 블로그를 만드는 것을 추천한다. 물론 만드는 과정에서 삽질도 많이 하고 불편한 점들도 있지만, 본인만의 블로그를 만든 경험은 그 자체로도 채용담당자에게 어필될 수 있다.

나도 처음에는 티스토리 블로그를 운영하다가, Git 사용법을 터득한 이후에는 Hugo 라는 정적 웹사이트 생성기와 GitHub Page 를 연동해서 개발 블로그를 만들었다. (참고 – Hugo 로 Github.io 블로그 만들기)

내가 사용한 Hugo 는 Go 로 만들어진 정적 웹사이트 생성기다. 때문에 문법이 생소해서 커스텀 하기가 조금 어려웠다. 하지만 기본적인 기능은 이 튜토리얼을 통해 대부분 익힐 수 있었다. 보통의 경우는 Hugo 가 아닌 Jekyll 을 사용하며, 요즘에는 Next.js 나 Gatsby 도 많이 사용한다. (참고 - 정적 웹사이트 생성기를 한눈에 비교해보는 사이트)

'깃헙 블로그 만들기'라고 구글링하면 친절한 설명의 블로그 글들이 수두룩하다. Git 사용법을 터득했다면, 깃헙 블로그를 만들어보자. 그 과정에서도 많이 배울 것이다.

Q. 어떤 내용을 작성하나?

앞서 설명한 TIL 저장소처럼, 학습한 내용을 정리하는 것이 기본이다. 나는 TIL 저장소에 정리한 내용 중, 조금 더 자세하게 작성하고 싶거나 공유하고 싶은 내용을 블로그에 올린다. 개인적인 생각도 자주 올리고, 연말 회고도 올린다. 개발 관련 세미나 혹은 컨퍼런스에 참여한 후기도 채용 담당자에게 나쁜 인상으로 남지 않을 것이다.

Q. 참고할만한 블로그가 있나?

아래의 블로그들을 참고하면 도움이 될 것이다.

- 기계인간 John Grib
    - 위키 형식의 블로그인데, 개발자에게 가장 최적화된 형태의 블로그라고 생각한다. 개발 학습의 깊이와 방식, 도구 자동화 등에 대해 많은 영감을 얻을 수 있으며, 따라 하고 싶은 유익한 내용이 많다.
- 기억보단 기록을
    - 앞서 언급한 이동욱 님의 블로그다. 개발자 블로그 중에 일 방문자가 가장 많을 것으로 추측된다. 그만큼 유익한 내용이 넘쳐난다.
- 기본기를 쌓는 정아마추어 코딩블로그
    - 학습한 내용이 잘 정리되어 꾸준하게 올라오는 블로그다. 학습 내용 정리 방법을 모르겠다면, 이 블로그를 롤모델로 삼아도 된다. 내가 만약 채용담당자라면 이 블로그 주인은 꼭 채용하고 싶을 것 같다.
- Velog
    - 블로그를 따로 만들지 않더라도 블로그 글을 올릴 수 있는 일종의 개발 블로그 플랫폼이다. 실제로 개인 블로그를 만들지 않고, Velog 에 글을 올리는 개발자들도 많다.
- Jbee.io
    - 블로그 디자인이 매우 깔끔하다. 블로그 템플릿을 오픈소스로 공개하셔서, 이 블로그와 같은 템플릿의 블로그들이 많이 보인다. 만약 블로그 템플릿을 고르기 어렵다면, 이분이 공개한 템플릿을 사용하는 것을 추천한다.
- Rinae's Devlog
    - 유익한 번역 글들이 많이 올라온다.

3.2.4. GitHub

앞서 언급되었지만, GitHub(깃헙)은 개발자 세계에서 가장 중요한 플랫폼이다. GitHub 과 Gitlab, Bitbucket 이 3 대장이라고 생각하면 된다. 하지만 GitHub 이 압도적으로 시장을 지배하고 있으므로 GitHub 을 기준으로 설명하겠다.

깃헙의 역할은 굉장히 다양하고 계속 발전하고 있지만, 지금 수준에서는 프로젝트 저장소이면서 협업 플랫폼이라고 이해하면 된다. Git 에 대한 이해가 필요하지만

가볍게 부연 설명을 하자면, 어떤 서비스를 만들 때 보통 다수의 개발자가 협업한다. 그런데 하나의 프로젝트에 여러 개발자가 각자 코드를 작성하면, 그 코드는 어떻게 통합되고 저장되며, 또 버전은 어떻게 관리될까? 그것을 가능하게 해주는 도구가 Git 이고, Git 을 사용하여 프로젝트를 저장하고, 공유하고, 협업하는 플랫폼이 GitHub 이다. (더 자세한 설명은 Git 과 GitHub 을 구글링 해보자.)

포트폴리오로서 GitHub 링크를 제출하기 위해, 비전공자가 현재 단계에서 GitHub 에 해야 하는 일들은 다음과 같다.

a. TIL 저장소 만들고 매일 학습한 내용 기록
b. 학원 또는 인강을 수강하며 작성한 코드를 올리는 GitHub 저장소 만들고 관리
c. (개인, 팀) 프로젝트를 올리고, 프로젝트를 설명하는 README.md 파일 추가

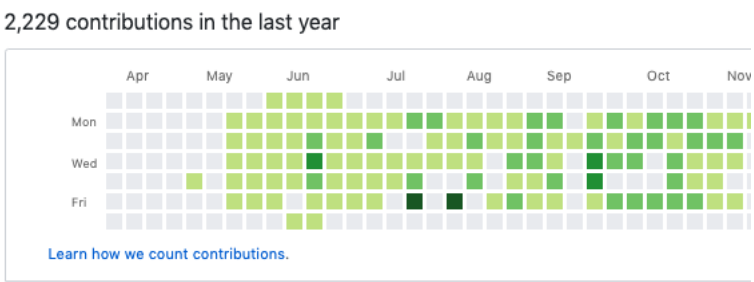
포트폴리오와 상관없이 GitHub 에 해야 하는 일들은 다음과 같다.

d. 본받을 점이 많은 개발자들 Follow 해서 GitHub 에 접속할 때마다 자극받기
e. 유명한 오픈소스를 Fork 해와서 코드 살펴보면서 배우기
f. 관심 있는 오픈소스에 기여하기

a. TIL 저장소 만들고 매일 학습한 내용 기록

학습하기 파트의 2.6 TIL(Today I Learned) 챕터 마지막 부분에 이유를 설명했다.

b. 학원 또는 인강을 수강하며 작성한 코드를 올리는 GitHub 저장소 만들고 관리



위의 커밋 그래프는 6 개월간 국비지원학원에 다닌 친구가 거의 매일 학원에서 작성한 코드를 깃헙에 올린 기록이다. 학원에서 작성한 코드를 매일 깃헙에 올리라고 한 나의 조언을 따라 한 것이다. (이 친구는 학원을 수료하자마자 좋은 스타트업에 취업했다.)

학원에 다니거나 인강을 수강하면, 배우면서 작성하게 되는 코드가 꽤 많을 것이다. 이 코드들을 그냥 흘려보내는 사람들이 많지만, 매일 깃헙에 올리는 것을 추천한다.
학원에서 작성한 코드를 깃헙에 올려서 관리하면 2 가지 장점이 있다.

우선, 코드의 History 가 기록되어 작성한 코드를 복습하기 편해진다. 학원에서 작성하는 코드가 추가, 수정, 삭제될 때마다 작업 단위를 묶어서 Commit 하고 깃헙 저장소에 Push 해두면, 코드의 History 가 저장된다. 복습하는 과정에서 현재의 코드가 어떤 과정을 거쳤는지 기억나지 않는 경우가 많을 것이다. 이때, Github 에서 코드의 이전 모습을 쉽게 볼 수 있고, 원하는 버전으로 돌아갈 수도 있다. 코드를 USB 에 담아오거나 클라우드 저장소에 올릴 필요가 없는 것은 덤이다.

두 번째로, 깃헙에 올린 코드는 그 자체로 포트폴리오에 사용할 수 있는 아이템이 된다.
채용담당자들은 학원에서 배운 것들의 구구절절한 설명 보다, 코드를 보고 싶어 한다.

Java 를 6 개월 동안 배웠다고 추상적으로 어필하는 것보다, 배운 것들을 코드로 보여주는 것이 확실하다. 물론, 각 저장소에 대한 설명과 네이밍이 제대로 되어 있어야 한다. 아무런 설명 없이, 네이밍도 제대로 안 된 깃헙 저장소는 아예 무시될 것이다.

또한, 위에 보여준 커밋 그래프를 보면 알 수 있듯이, 이 사람이 얼마나 꾸준하게 코딩을 했는지 알 수 있다. 커밋 그래프가 개발 실력과 정비례 관계는 아니지만, 적어도 신입 개발자 채용 과정에서 비전공자를 평가하는 채용담당자들은 눈여겨볼 항목이다.
게다가 Git 과 GitHub 사용 능력까지 자연스럽게 검증된다.

c. (개인, 팀) 프로젝트를 올리고, 프로젝트를 설명하는 README.md 파일 추가
프로젝트를 진행했거나 진행 중이라면, 깃헙 저장소에 올려서 관리하는 것을 추천한다.
취업뿐만 아니라 이직할 때에도 GitHub 저장소에 올려진 (개인, 팀) 프로젝트는 채용담당자들이 GitHub 에서 가장 먼저 확인하는 저장소이기 때문이다.

학습한 내용을 올린 저장소는 사실 따라 친 코드가 많기 때문에, 꾸준히 학습했다는 것과 학습 내용을 어필하는 용도다. 반면, 프로젝트 저장소는 진짜 개발 실력을 평가할 수 있는 저장소이기 때문에 제대로 관리해야 한다.

d. 본받을 점이 많은 개발자들 Follow 해서 GitHub 에 접속할 때마다 자극받기

나는 개발 블로그를 보다가 블로그 주인이 열정적인 개발자라는 생각이 들면, 개발자의 깃헙에 들어가서 팔로우한다. 그러면 GitHub 에 처음 접속했을 때, 내가 팔로우한 개발자들이 최근에 한 활동들이 SNS 처럼 보여진다.

팔로우한 개발자가 어떤 프로젝트에 '좋아요(Star)'를 눌렀다면, 그 프로젝트 저장소에 방문해보자. 그 과정에서 흥미로운 프로젝트들을 발견하기도 하고, 학습에 도움이 되는 저장소를 발견할 것이다.

비전공자인 우리는 개발자 인맥이 부족하기 때문에, 앞서간 개발자들의 활동을 염탐(?)이라도 해서 정보를 얻어내야 한다. 이유는? 기술은 빠르게 변하기 때문이다. 그 말인즉슨, 늦게 진입한 우리에게도 앞서나갈 기회가 생긴다는 뜻이다. 물론 빠른 변화를 모두 따라갈 필요는 없지만, 적어도 기술 생태계가 어떤 방향을 흘러가는지 알아야 기회에 대비할 수 있다.

기술 생태계의 흐름을 파악하는 가장 좋은 방법은 앞서나간 개발자들을 팔로우하고, 그들이 하는 이야기와 행동을 관찰하는 것이다. 그분들이 들으면 기분 나쁠 수 있는 표현이지만, 그들을 일종의 정찰병으로 활용하는 셈이다.

개발을 처음 접하는 비전공자들은, 아직 기본기 학습만으로도 벅찰 것이다. 그렇다고 저 멀리 앞서나간 개발자들의 활동은 본인과 별개라 생각하는 것은 위험하다. 고개 숙인 상태로 눈앞에 닥친 일들만 처리하면 자연스럽게 도태되기 때문이다. 고개를 들고 정찰대가 보내는 메시지를 받으면서, 기술 생태계의 흐름에 역행하지 않도록 주의하고, 변화의 시기에 남들보다 앞서나갈 준비를 해야 한다.

e. 유명한 오픈소스를 Fork 해와서 코드 살펴보면서 배우기

이 일은 감히 '정석'이라고 할 만한 개발 학습법이다. 나도 아직 제대로 하지 못하고 있지만, 반드시 해야 할 숙제라고 생각하고 있다. 유명한 오픈소스는 대한민국 전국구 레벨이 아니라, 전 세계적 레벨의 개발자들이 만들어가는 경우가 많기 때문에 배울 점이 무궁무진하다고 한다.

f. 관심 있는 오픈소스에 기여하기

이것 또한 나는 못 하고 있는 것인데, 우연히 친분을 쌓게 된 미국 AirBnb 본사에서 일하는 시니어 개발자분께서 특히 강조하신 부분이다. 미국에서는 채용과정에서 가장 큰 임팩트를 줄 수 있는 것이 오픈소스에 기여한 경험이라고 한다. 한국에서도 오픈소스 기여의 중요성이 강조되기 시작했기 때문에, 오픈소스 기여 경험이 개발자 취업과 이직에 점점 더 큰 영향을 끼칠 것으로 생각된다. 오픈소스는 보통 깃헙에 저장소가 있다. 깃헙에서 관심 있는 오픈소스를 찾았다면 작은 기여라도 해보는 것을 추천한다.

3.3. 코딩 테스트

대부분의 좋은 회사들은 코딩 테스트(혹은 알고리즘 테스트) 전형을 진행한다. 좋은 회사에 가지 못한다는 생각에 코딩 테스트 준비를 아예 안 하는 비전공자들이 있는데, 무조건 준비하는 것이 좋다. 이 챕터에서는 1.코딩 테스트를 준비해야 하는 이유와 2.준비 방법, 3.내가 코딩 테스트에 합격한 노하우를 설명한다.

3.3.1. 비전공자도 코딩 테스트를 준비해야 하는 이유

비전공자들은 코딩 테스트에 아예 눈길조차 주지 않고 취업 시장에 뛰어드는 경우가 많다. 본인이 감당할 레벨은 아니라고 착각하고 스스로 한계를 정해버리는 것이다. 내가 그랬었다. 그때 당시의 내 생각은, '당장 학원에서 배우는 것들도 잘 모르겠는데, 코딩 테스트를 준비할 필요가 있나?' 였다. 이미 잘하는 개발자들만 코딩 테스트를 준비하는 것으로 착각했다. 알고리즘 문제 풀이를 해보지도 않고 겁부터 먹은 것이다.

물론 비전공자가 처음 입사하는 회사는 채용 과정에서 코딩 테스트까지 진행할 여력이 되지 않는 회사일 가능성이 높다. 하지만 준비는 해야 한다. 세 가지 이유가 있다.

a. 절대 평가로 코딩 테스트 합격 여부를 결정하는 회사들도 있다.
코딩 테스트를 오랫동안 준비해온 전공자들을 비전공자가 단기간에 압도하기는 어렵다. 때문에 상대 평가로 코딩 테스트 합격 여부를 결정하는 회사의 경우, 코딩 테스트에서 통과하기가 쉽지 않을 수 있다. 하지만 절대 평가로 합격 여부를 결정하는 회사들도 있다. 절대 평가의 경우, 단기간 준비한 비전공자라도 충분히 승산이 있다.

b. 코딩 테스트와 알고리즘 대회를 구분하면 두려움을 극복할 수 있다.
내가 코딩 테스트를 두려워했던 이유는, 코딩 테스트와 알고리즘 대회를 구분해서 생각하지 않았기 때문이다. 우연히 CodeForces 의 알고리즘 대회 문제와 그 풀이를 적은 블로그 글을 보았는데, 이때 코딩 테스트에 대한 잘못된 선입견이 생겼던 것이다. 코딩 테스트와 알고리즘 대회를 구분해서 생각해야 한다. 코딩 테스트에서 마주할 문제들의 수준은 충분히 준비할 수 있다. 알고리즘 대회 입상을 목적으로 하는 Competitive Programmer 들이 존재하는데, 그들만의 리그라고 여기고 먼 산 바라보듯이 바라보자. 비전공자들은 우선 취업을 위한 코딩 테스트만 준비하면 된다.

c. 첫 회사에서 생각보다 빨리 이직을 생각하게 될 것이다.

이 이유는 경험에 의한 것이다. 내가 다닌 학원에는 23 명의 동기가 있었고, 절반 정도의 동기들이 첫 회사에서 6 개월도 못 버티고 퇴사하거나 이직을 했다. 나 또한 첫 회사(작은 스타트업)에서 7 개월 만에 이직했는데, 이직할 때 코딩 테스트 때문에 고생했다. 취업했더라도 더 좋은 회사에 이직하고 싶다면, 당장 준비를 시작해야 한다.

비전공자가 처음 취업한 회사가 10 점 만점에 3 점짜리 회사라고 가정하자. 그런데 입사한 지 얼마 지나지 않아 이직하고 싶어졌다. 그럼 보통 5 점짜리 회사로 이직할 계획을 세울 텐데, 잘못된 판단이다. 7 점짜리 회사로의 이직을 준비해야 한다.

3 점짜리 회사에서 5 점짜리 회사로 이직하는 데 드는 비용(노력)이 100 이라면, 3 점짜리 회사에서 7 점짜리 회사로 이직하는 데 드는 비용은 120 이고, 10 점짜리 회사로 이직하는 데 드는 비용은 200 이다. 10 점짜리 회사로의 이직은 정말 어려울 수 있지만, 추가 비용 20 을 더 투자해서 7 점짜리로 이직하는 것은 충분히 가능하다. 여기서 말하는 추가 비용 20 이 바로 코딩 테스트 준비다. 그리고 7 점짜리 회사에서 경력을 더 쌓아서 3~5 년차 때 10 점짜리 회사로 이직하면 된다. (채용 시장에서 3~5 년차 개발자의 수요가 가장 많기 때문에 이직하기 가장 쉬운 시기다.)

내가 추천하는 방향은, (급여를 제대로 지급하고 야근이 심하지 않다는 가정하에) 3 점짜리 첫 회사에 1 년 정도 다니면서 이직 준비를 열심히 하는 것이다. 그러다가 7 점짜리 회사들이 신입 개발자를 채용할 때, 중고 신입으로 지원하는 것이다.

3 점짜리 회사 취업도 힘들었는데 7 점짜리 회사에 어떻게 합격하느냐고 반문할 것이다. 3 점짜리 회사에 취업하기 힘들었던 이유는, 경력이 0 년 0 개월이었고 개발자로서 어필할 재료들도 충분하지 않았기 때문이다. 그런데 1 년간 이직 준비를 해왔다면 어필할 재료들도 꽤 쌓였을 테고, 코딩 테스트도 어느 정도 준비되었으며, 무엇보다 1 년이라는 경력도 생겼다. 0 에서 3 보다 3 에서 7 이 더 쉽다.

첫 회사에 입사한 첫날부터, 7 점짜리 회사를 목표로 이직 준비와 코딩 테스트 준비를 시작해야 한다. 미리 준비하지 않으면, 3 점짜리 회사에서 3 년을 일해도 5 점짜리 회사조차 이직하기에 벅찰 것이다. 준비를 빨리 시작해서 손해 볼 일은 없다.

3.3.2. 코딩 테스트 준비 방법

목표로 하는 회사에 따라, 개인의 실력에 따라, 주어진 시간에 따라 코딩 테스트를 준비하는 기간과 방법은 달라진다. 비전공자의 입장에서 단기적으로 코딩테스트를 준비하는 방법과 장기적으로 준비하는 방법으로 나누어서 설명하겠다.

a. 단기적으로 코딩 테스트 준비하기

단기적으로 코딩 테스트를 준비하는 가장 효율적인 방법은, 본인이 원하는 회사에서 채용할 때 사용하는 코딩 테스트 플랫폼의 문제를 최대한 많이 풀어보는 것이다. 물론 직접 플랫폼을 구축해서 사용하는 회사들도 있지만, 앞서 말한 7 점짜리 회사들은 코딩 테스트 플랫폼을 활용하는 경우가 많다. 아래는 대표적인 코딩 테스트 플랫폼들이다.

- LeetCode
- 프로그래머스
- Codility
- Hackerrank

위의 플랫폼들은 연습문제를 제공한다. 플랫폼마다 문제가 다른 것은 아니지만, 원하는 회사가 사용하는 플랫폼의 연습문제를 미리 풀면서 코드 작성 환경과 테스트 케이스 추가 방법 등을 미리 익혀두는 것이 좋다. 또한 문제가 영어로 주어지는 플랫폼의 경우, 코딩 테스트 문제 설명에 자주 나오는 영어 표현들을 미리 익혀서 실제 테스트를 진행할 때 영어사전을 검색하는 일이 없어야 한다.

만약 자료구조와 알고리즘에 대한 개념이 전혀 없다면, 알고리즘 도감 이라는 책과 어플로 빠르게 학습하고 시작하는 것을 추천한다. 아무리 단기적으로 코딩 테스트를 준비하더라도, Big O 표기법 및 시간복잡도와 공간복잡도의 개념, 주요 자료구조와 알고리즘의 개념은 이해하고 문제를 풀어야 하기 때문이다.

코딩 테스트에 합격하더라도, 면접에서 코딩 테스트 문제에 대해 다시 물어보는 경우가 있다. 이때, 가장 중요하게 여겨지는 부분이 제출한 풀이의 시간복잡도에 대한 것인데, 그 개념조차 알지 못한다면 코딩 테스트를 통과하더라도 면접에서는 탈락할 것이다. 본인이 처한 상황이 어쩔 수 없다면 코딩 테스트를 단기적으로 준비해야겠지만, 어떻게든 장기적으로 준비하는 것을 추천한다. 단기적으로 준비하면 실패 확률이 높다.

b. 장기적으로 코딩 테스트 준비하기

장기적으로 준비할 경우, 문제 풀이부터 시작하는 것이 아니라 자료구조/알고리즘부터 제대로 학습해야 한다. 전공자들은 자료구조와 알고리즘을 학부 수업에서 배우지만, 비전공자들은 아예 처음 접하는 것들이기 때문에 인터넷 강의와 책 등을 이용한다.

다양한 자료구조/알고리즘 인터넷 강의가 있지만, 내가 추천하는 강의는 권오흠 교수님의 영리한 프로그래밍을 위한 알고리즘 강좌다. 이 강의는 무료이기 때문에 부담 없고, 교수님이 가르치기 때문에 차근차근 제대로 설명해주신다. 물론 이 강의로 모든 알고리즘을 커버할 수는 없기 때문에 추가적인 학습이 필요하다. 추가적인 학습은 미국의 온라인 자료구조/알고리즘 강의를 수강하면 좋다.

미국에서는 개발자 채용 과정에서 한국보다 자료구조/알고리즘 및 코딩 테스트의 비중이 크기 때문에 유명 대학 강의를 포함해서 질 좋은 강의들이 많다. 사실 좋은 강의들이 너무 많아서 선택하기 어려울 정도이기 때문에, 2020 년 최고의 자료구조/알고리즘 무료 강의 모음을 통해 하나씩 맛보기 강의를 들어보고 발음이 잘 들리는 강의를 선택하는 것을 추천한다.

비단 코딩 테스트뿐만 아니라, 개발자로 일할 때도 자료구조에 대한 이해는 반드시 필요하다. 때문에 본인이 사용하는 언어로 쓰인 자료구조 책을 한 권 제대로 학습하는 것을 추천한다. 나는 자바로 배우는 쉬운 자료구조라는 책으로 학습했는데, 기본적인 자료구조를 학습하기에는 나쁘지 않았지만, 정성을 쏟은 책은 아니었다.

문제 풀이 플랫폼은 LeetCode 가 대세로 자리 잡았다. 무료로 풀 수 있으며, 문제 해설은 유료로 막혀있는 경우도 있지만 문제마다 Discussion 탭에 들어가면 다른 사람들이 자신이 제출해서 통과한 코드를 올려 둔다. 때문에 다른 사람의 풀이 중 가장 투표를 많이 받은 풀이를 보면서 충분히 학습할 수 있다.

코딩 테스트를 준비하기 위한 책 중에 정석은 Cracking the Coding Interview 라는 책이다. 이 책은 미국 개발자들이 코딩 테스트 준비할 때 보는 수학의 정석 같은 책이다. 나도 구매해서 틈날 때마다 보고 있는데, 영어 독해만 된다면 시중에 이 책보다 나은 선택지는 없다고 생각한다.

3.3.3. 10 일 준비하고 코딩 테스트에 합격한 노하우

앞서 코딩 테스트를 미리 준비해야 한다고 한 이유는, 내가 그렇게 하지 않아서 힘들었기 때문이다. 나는 첫 회사에서 예정보다 빠르게 퇴사 결심을 하게 되어 갑작스럽게 코딩 테스트를 준비했다. (2019 년 회고 참고)

당시 기준으로 6 개월 전에 그룹 스터디를 통해 자료구조 책 1 권을 학습한 상태였고, 알고리즘 강의를 듣거나 문제를 풀어보지는 않은 상태였다. (권오흠 교수님의 강의도 이직한 이후에 수강했다) 코딩 테스트를 봐야 하는 회사에 지원할 생각이 없었기 때문에, 코딩 테스트 준비가 하나도 안 된 상태였다.

탈락하리라 생각했던 회사의 서류 전형에 통과한 후, 일주일 내로 Codility 에서 코딩 테스트를 응시해야 했다. 시간이 너무 촉박하다고 생각한 나는 인사팀에 연락해서 시간을 조금 더 달라고 부탁했다. 다행히 3 일을 더 받아서 총 10 일의 준비 기간을 마련할 수 있었다. 결과적으로 코딩 테스트에 통과했는데, 거의 백지상태에서 10 일 동안 어떻게 준비했는지 공유해본다.

a. Codility 연습문제 풀이에 집중

우선, 서류 합격 메일을 통해 Codility 를 사용한다는 것을 알게 되었고, Codility 에 무언가 얻을 것이 있는지 둘러보았다. 다행히 연습 문제들을 제공하고 있었는데, 17 개의 주제로 나뉘어 있었고, 각 주제별로 2~4 문제가 있었다. 이 문제들만 풀어도 시간이 부족할 것 같아서 이 문제들에 집중하기로 했다. 게다가 각 챕터 별로 개념을 설명해주는 읽기 자료들도 제공해줘서, 읽기 자료를 먼저 읽고 문제 풀기를 시도했다.

b. Youtube 에서 풀이 영상 보기

물론 문제는 잘 풀리지 않았다. 문제를 어떤 방식으로 해결해야 할지 전혀 감을 잡지 못했다. 고등학교를 졸업한 이후, 이런 종류의 문제 풀이에 머리를 써본 적이 없기 때문에 멍청이가 된 기분이었다. 이렇게는 안 되겠다 싶어서 연습문제 제목을 Youtube 에 검색해보았다. 별다른 기대를 안 하고 검색했는데, 차근차근 설명해주는 문제 풀이 영상들이 있었다. 누군가 문제를 풀어나가는 과정을 시각적으로 보고 나니, 그제야 조금씩 감이 잡히기 시작했다.

c. 시간복잡도부터 생각하며 문제 풀기

코딩 테스트는 Big O Notation 의 개념을 이해하는 것이 중요했으며, 어떤 방식으로 풀더라도 시간복잡도 O(N) 혹은 O(nlogN)으로 푸는 것이 핵심이었다. 처음에는 모든 문제를 For 문을 중첩해서 풀려고 했는데, 시간복잡도를 우선적으로 생각하며 문제에 접근하니 다양한 자료구조가 떠올랐다. (자료구조 책을 읽었던 것이 이때 도움 되었다)

d. 안 풀려도 최소 1 시간은 고민하기

알고리즘 문제들을 풀다 보니, 고등학생 때 4 점짜리 수학 문제들을 고민하며 풀던 것과 비슷한 느낌이었다. 4 점짜리 문제들은 단번에 해결 방법이 보이지 않았지만, 고민하다 보면 실마리가 보이는 문제들이었다. 코딩 테스트도 그랬다. 물론 1 시간 동안 고민해도 해결책을 못 찾거나 테스트 케이스를 통과하지 못하곤 했지만, 1 시간 동안 이리저리 시도하는 과정에서 문제 풀이 능력이 길러지는 느낌이었다.

코딩 테스트에서 문제 해결의 핵심에 접근하기 위해서는, 각 단계의 갈림길에서 올바른 선택을 해야 한다. 그런데 1 시간 동안 고민하는 과정에서는 잘못된 길을 선택해보는 것이다. 갈림길에서 잘못된 길을 선택해보고 그 끝은 막다른 길이란 것을 깨달아야, 다른 문제에서 비슷한 갈림길을 만났을 때 옳은 길을 선택할 수 있는 감(혹은 실력)이 생긴다. 이 방법이 효율적인 방법인지는 확신할 수 없으나, 나에게는 맞는 방법이었다.

e. 출퇴근 길 지하철에서 나 홀로 타임어택하기

코딩 테스트를 준비하던 10 일간, 여전히 기존 회사에 출근하던 퇴사 예정자였다. 퇴사 예정자라고 대놓고 사무실에서 알고리즘 문제를 풀 수는 없었기 때문에 준비할 시간이 턱없이 부족했다. 최대한 많은 시간을 코딩 테스트 준비에 투자하고 싶어서 방법을 고민하다가, 출퇴근길 지하철에서도 문제를 풀었다.

방법은 간단하다. 전날 밤에 미리 노란색 옥스퍼드 노트에 Codility 연습문제 2 개를 적어둔다. 난이도가 낮다고 표시된 문제들만 적었다. 그리고 출근길과 퇴근길 지하철에서 각각 30 분간 노트에 적힌 문제를 펜으로 풀었다. 30 분 안에 1 문제를 반드시 풀어야 하는 타임 어택이라 생각하고, 실제 코딩 테스트처럼 시간 압박을 느끼며 문제 푸는 연습을 한 것이다.

이 방법은 효과가 굉장히 좋았다. 지하철에서는 서 있어도 양손이 자유로웠으며, 도착 시간이 정확하고 도착하면 무조건 내려야 하므로, 따로 시간을 재지 않아도 자연스럽게 시간 압박을 느낄 수 있었다. 실제 코딩 테스트에서, 시간이 촉박한 상황에도 이상하게 집중력이 좋았는데, 지하철에서 내리기 직전까지 집중해서 문제를 풀던 것이 도움 되었다고 생각한다. 무의미하게 보내던 출퇴근 길 지하철에서 1~2 개의 문제를 더 풀면서, 효율적으로 코딩 테스트를 준비할 수 있었다.

f. 운도 따라줬다.

총 3 문제를 2 시간 40 분 내에 풀어야 했는데, 가장 어려운 세 번째 문제는 종료 시간 20 분을 남겨두고 시작해서 거의 포기한 상태였다. 그런데 끝까지 집중해서 해결책을 의외로 쉽게 찾을 수 있었는데, 시간이 부족해서 제대로 풀이를 작성하지 못하고 끝났다. 하지만 Codility 는 부분 점수도 있기 때문에 세 번째 문제에서도 부분 점수를 따서 합격할 수 있었다.

또한 우리 회사의 코딩 테스트가 상대평가가 아니라 절대 평가라는 점에서 운이 좋았다. 물론 절대 평가라도 점수가 높을수록 면접에서도 좋은 평가를 받겠지만, 면접을 볼 수 있다는 것만으로도 운이 좋았다고 생각한다. 상대 평가였다면, 같이 합격한 동기의 엄청난 실력을 고려했을 때, 나는 면접까지 갈 기회조차 없었을 것이다.

3.4. 기술 면접

개발자 채용에서의 면접은 대부분 기술 면접이다. 이력서와 포트폴리오를 열심히 준비하고, 코딩 테스트를 통과했더라도 기술 면접에서 역량을 어필하지 못하거나 잠재력을 보여주지 못하면 무조건 불합격한다. 이 챕터에서는 비전공자에게는 생소한 기술 면접은 어떤 면접인지 설명하고, 기술 면접 준비 방법을 공유한다.

3.4.1. 기술 면접은 어떤 면접인가?

개발자는 기술직이기 때문에 일반적인 면접이 아니라 기술 면접을 진행한다. 일반적인 면접에서는 주로 가치관에 관해 묻거나 사회/경제 이슈를 직무 역량과 연결한 질문들을 통해 지원자들을 평가한다. 반면, 기술 면접에서는 기술적인 질문들을 통해 이력서와 포트폴리오에 가려진 지원자의 기술 이해도를 평가하고, 개발자로서의 성장 가능성, 팀 플레이어로서의 의사소통 능력 등을 평가한다.

a. 기술 이해도 평가

가령 A 라는 기술이 팀에서도 쓰이는 기술이고 지원자가 A 기술을 사용한 경험이 있을 경우, 혹은 팀에서는 쓰이지 않지만 지원자가 프로젝트에 사용한 기술의 경우 면접관은 지원자의 기술 이해도를 평가하기 위해 아래와 같은 질문들을 할 것이다.

- A 기술의 핵심 개념들은 무엇이 있고, 설명할 수 있는지
- A 기술의 핵심 개념들과 연결되는 기초적인 개념들을 제대로 이해하고 있는지
- 다른 대체 기술들을 사용하지 않고, A 기술을 선택한 이유가 무엇인지
- A 기술의 한계는 무엇이고, 한계 상황에서 어떻게 대처할 것인지

b. 성장 가능성 평가

성장 가능성, 잠재력은 신입 개발자 채용 면접에서 가장 중요하게 평가되는 요소다. 신입 개발자 채용의 목적은 지금은 조금 부족하더라도 잠재력 있는 개발자를 채용하는 것이다. 특히 비전공자라면 더욱 중요하다. 면접관들은 대부분 전공자 출신에 개발을 오랫동안 해온 분들이다. 때문에 전공자들이 어떤 것들을 배우고 익혔는지 알고 있는 만큼, 비전공자들이 전공자에 비해 부족한 부분들도 알고 있다. 그럼에도 불구하고 비전공자를 면접에 불러들인 이유는, 성장 가능성(잠재력)을 기대하기 때문이다.

잠재력을 어필하려면 재료가 필요하다. 재료는 이력서와 포트폴리오 챕터에서 언급한 것들이면 충분하다. 면접에서는 그 재료들에 대해 질문을 할 것인데, 자신이 얼마나 개발을 즐기고 있고, 얼마나 열정적으로 노력하는지 보여주면 된다.

c. 의사소통 능력 평가

개발자가 혼자 일하는 직업이라고 착각하는 경우가 있다. 틀렸다. 개발자는 전적으로 협업하는 직업이다. 게다가 개발자들끼리만 협업하는 것이 아니라, 개발자와 타 직군(기획자, 디자이너, 퍼블리셔, 보안 담당자 등) 간의 협업도 필요하다. 이렇듯 개발자의 업무는 코드를 작성하는 일에만 그치지 않고, 제품과 서비스를 함께 만들어가는 많은 사람들과 의사소통하고, 함께 문제를 해결하고, 함께 새로운 것을 만드는 것까지 확장된다.

때문에 면접에서는 의사소통 능력을 반드시 평가한다. 물론 면접에서 대놓고 '지금부터 의사소통 능력을 평가하겠습니다.'라고 알려주지 않지만, 면접 중에 오고 가는 대화를 통해 은연중에 지원자의 의사소통 능력을 평가한다. 코딩 테스트에서 만점을 받고 기술 이해가 완벽해도, 의사소통 능력이 부족하면 면접에서 떨어진다. (실제 우리 팀 면접에서도, 뛰어난 개발 실력을 갖춘 지원자들이 의사소통 능력 부족으로 탈락하는 경우가 많다)

여기서 말하는 의사소통 능력이란, 언변이 뛰어난 사람을 뜻하지 않는다. 기존의 개발자들도 언변이 뛰어난 경우가 드물다. (문과 계열 직무에서 일했을 때와 비교하면 차이가 심하다) 여기서 말하는 의사소통 능력이란 A 라는 질문을 했으면 질문을 경청하고 A 에 대해 답변하는 기본적인 능력이다.

그런데 A 를 물어봤는데 B 에 대해 대답하고 C 에 대해 언급하는 등, 면접관이 하는 질문의 의도를 파악하지 못하고 자신의 생각만 말하는 개발자들이 엄청나게 많다고 한다. 대화가 자연스럽게 이어지지 않는 것이다. 이 경우 무조건 탈락이다. 그 지원자가 아무리 개발을 잘해도, 기존의 팀원들과 소통이 안 되면 불필요한 리소스가 낭비가 발생할 것이다. 개발자 리소스는 회사 내에서도 가장 비싼 편에 속하기 때문에, 면접관은 그 지원자를 채용하지 않을 것이다. 면접관은 실력이 약간 부족하더라도, 기존 팀원들과 소통이 잘 되고, 팀워크가 잘 맞을 것 같은 지원자를 채용한다.

3.4.2. 기술 면접 준비 방법

처음에는 기술 면접이 무엇인지 몰라서 막막했다. 심지어 'Java 개발자를 채용하는 면접이니까 Java 만 더 공부하면 될까?'라는 식으로 생각했었다. 하지만 다양한 기술 면접 후기 글들을 찾아보고, 기술 면접 대비 자료들을 찾다 보니 노하우가 생겼다.

나는 아래의 순서로 기술 면접을 준비하기를 추천한다.

a. 기술 면접에 대비하기 위한 학습 범위와 우선순위를 정한다.
b. 면접에서 자주 나오는 질문들을 한글/영문으로 검색해서 정리한다.
c. 학습 내용을 본인 글로 써서 정리하고, 말로 설명할 수 있을 때까지 학습한다.
d. 최소한 면접 3 일 전부터 모의 면접을 진행한다.

a. 기술 면접에 대비하기 위한 학습 범위와 우선순위를 정한다.
개발 기술은 광범위하고 깊이도 깊기 때문에, 기술 면접 대비를 위해서는 학습 범위를 좁히는 것이 좋다. 신입 개발자 채용 면접인 만큼, 모든 것을 커버한다는 생각보다는 중요한 기술들에 대해 제대로 답변하는 것을 목표로 준비해야 한다. 학습 범위와 우선순위는 아래의 기준으로 결정하면 된다.

학습 범위

- 주로 사용하는 프로그래밍 언어 (Java, Python, Javascript 등)
- 위의 언어와 함께 쓰이는 프레임워크 (Spring, Django, Express, Vue.js 등)
- 나의 이력서/포트폴리오에 쓴 기술들
- 지원한 팀이 사용하는 기술들

학습 우선순위

- 주 사용 언어
- 주 사용 프레임워크
- 지원한 팀에서도 사용하고, 나의 이력서/포트폴리오에도 쓴 기술
- 지원한 팀에서는 사용하지 않지만, 나의 이력서/포트폴리오에 쓴 기술
- 지원한 팀에서는 사용하지만, 나의 이력서/포트폴리오에 없는 기술

b. 면접에서 자주 나오는 질문들을 한글/영문으로 검색해서 정리한다.
정해진 것은 없지만, 각 기술마다 면접에서 자주 나오는 질문들이 있다. 앞서 결정한 우선순위에 의해, 가장 먼저 학습해야 할 기술이 Java 라고 가정해보자. 물론 Java 기본서를 다시 처음부터 끝까지 읽는 방법도 있지만, 비효율적이다. 이미 면접을 본 사람들이 정리한 Java 면접 질문 리스트를 최대한 모아서 정리한 후, 몰랐던 내용을 먼저 학습하는 것을 추천한다. 한글과 영문 각각 검색(ex. Java 면접 질문/ Java interview questions)해야 하는 이유는, 다양한 자료를 조합해야 더 완벽한 예상 질문 리스트를 만들 수 있기 때문이다.

c. 학습 내용을 본인 글로 써서 정리하고, 말로 설명할 수 있을 때까지 학습한다.
앞서 언급한 방식대로 예상 질문 리스트를 정리하면, 그 양이 꽤 많아서 다른 사람이 작성해둔 설명만 보고 넘어가기 쉽다. 하지만 면접에서는 그 내용을 내 입으로 말해야 한다. 내 입으로 말하기 위해서는 내가 주로 사용하는 언어로 정리되어야 하므로, 본인의 언어로 직접 내용을 정리(작성)해야 한다.

Notion 같은 메모 앱을 활용해서 정리해두면, 작성하기도 편리하고 작성해둔 내용을 PC, 노트북, 휴대폰에서 별다른 동기화 없이 바로바로 확인할 수 있다. 나 또한 Notion 에 예상 질문 리스트를 만들어두고, 각 질문에 대한 답변을 틈틈이 나의 언어로 작성했다. 덕분에 면접 보러 가는 길에는 Notion 에 정리한 내용을 한번 쭉 읽기만 하면 되었다.

d. 최소한 면접 3 일 전부터 모의 면접을 진행한다.
모의 면접은 반드시 진행해야 한다. 앞서 언급한 예상 질문 리스트를 정리했다면, 면접 3 일 전부터는 예상 질문 리스트의 질문 부분만 보고 소리 내서 답변하는 연습을 해야 한다. 분명히 제대로 답변하지 못하는 질문들이 있을 것이다. 제대로 답변하지 못하는 것은 제대로 이해하지 못했기 때문이다. 최소한 면접 3 일 전에는 제대로 이해하지 못한 부분들을 찾아야 다시 학습할 시간이 있다.

3.4.3. 기술 면접 대비 자료 모음

- Technical Interview Guidelines for Beginners
  - 한재엽 님이 오픈소스 형태로 운영하시는 깃헙 저장소
  - 50 명 이상의 개발자들이 Contributor 로 참여해서 만들어졌다.
  - 방대한 양의 기술 면접 준비 자료가 카테고리별로 잘 정리되어 있다.

- Awesome Interviews
  - 믿고 보는 Awesome list 중 하나인 저장소다.
  - 다양한 기술의 면접 질문 리스트 링크가 모여있다.

- 프론트엔드 인터뷰 핸드북 (번역)
  - 깃헙 Star 를 4 만 4 천개나 받은 Front-end Job Interview Questions 라는 프론트엔드 면접 질문 저장소에 올라온 질문들에 대한 답변을 제대로 찾을 수 없어서 만들어진 저장소
  - 한국어로 번역되어 편리하다.

- Tech Interview
  - 3 명의 컴공 4 학년 졸업 예정자분들이 직접 기술 면접을 준비하며 면접 대비 내용을 정리한 저장소.
  - 저장소에 질문들만 나열되어 있고, 답변은 따로 저장되어 있기 때문에 질문만 보고 답변해보는 연습을 할 수 있어 유익하다.

- TechnicalNote
  - 위 저장소와 비슷하게 3 명의 주니어 개발자들이 필기시험과 면접에서 직접 경험한 내용 위주로 기술 면접 대비 자료가 정리되어 있다.

- 개발자 인터뷰 시 면접관에게 할 만한 질문 모음
  - 면접은 내가 지원한 회사와 팀에 대해 알아보는 기회이기도 하다. 이 글은 면접에서 면접관에게 질문할 만한 내용을 담고 있다. 아직 개발자로 일해본 경험이 없는 사람들에게는 매우 유익한 정보다.

- 프론트엔드 개발자 인터뷰 후기 (면접 질문 정리)
  - 프론트엔드 경력직 면접을 보신 분께서, 기초적인 내용들을 묻는 면접 질문들을 경험한 후에 작성하신 글

4. 부록

4.1. 입사하기 전에 알아두면 좋을 것들

서류 전형과 코딩 테스트, 기술 면접까지 모두 통과하고 최종 합격했다면, 고생한 만큼 여행도 가고 푹 쉬면서 리프레시하기를 추천한다. 하지만 입사일을 앞두고 있다면 아래 설명하는 3 가지만이라도 미리 준비해두고 입사 첫날을 맞이하자.

4.1.1. 질문 잘하는 방법

앞서 이야기했듯이 개발자는 협업하는 직업이기 때문에 동료 개발자들에게 질문할 일들이 많다. 그런데 신입 개발자의 경우 질문 거리가 산더미처럼 쌓인다. 때문에 질문 잘하는 방법을 미리 알고 입사하는 것이 좋다. '질문은 그냥 하면 되지 않나?'라고 생각한다면, 질문을 잘하는 개발자 이 글을 반드시 읽어보자. 이 글 하나에 내가 전달하고 싶은 내용이 다 포함되어 있다. 아래 내용은 사족이니 넘겨도 무방하다.

질문하지 않으면 절대 알 수 없는 것들이 있다. 서버 정보, 각종 보안 설정, 협업 규칙, 팀 내에서만 사용하는 자체 라이브러리 등등, 동료 개발자에게 질문해서 얻어야 하는 정보가 있는 것이다. 이 정보에 대해 질문하지 않는 것 또한 문제다. 신입 개발자라도 능동적으로 행동하는 것이 좋다. 능동적으로 행동하는 것은 질문하지 않고 알아서 하는 것이 아니라, 필요한 정보를 얻기 위해 자발적으로 질문하는 것이다.

반면, 동료 개발자에게 묻지 않아도 구글링을 통해 알 수 있는 정보들이 있다. 질문하지 않으면 알 수 없는 정보와 구글링을 통해 알 수 있는 정보가 혼재된 상태에서, 고민 없이 무조건 동료 개발자에게 질문하는 것은 최대한 지양해야 한다. 동료 개발자에게 질문하는 것은 동료 개발자의 시간을 빼앗는 행동이다. 때문에 구글링으로 해결할 수 있는 문제는 최대한 구글링으로 해결하는 것이 동료를 배려하는 길이다.

나는 질문 잘하는 방법을 모른 상태로 입사했다. 질문 거리가 생기면 두뇌를 거치지 않고 바로 주변 동료들에게 질문했었는데, 돌이켜보니 굉장한 민폐였다. 개발을 하다 보면 Context(맥락)라는 것에 갇힌다. 어떤 복잡한 일에 몰두하다가 그 맥락에 깊이 빠져든 상태인데, 그 상황에서 갑자기 전혀 다른 Context 의 질문을 받으면 본인이 집중하고 있던 맥락에서 질문의 맥락으로 Context Change 가 이루어져야 한다. 그런데

이때 집중력이 쉽게 깨져버린다. 개발자들은 코드를 짜다가 갑자기 다른 맥락의 코드에 대해 대화를 시작하면 "아, Context Change 가 안된다. 무슨 내용이었지?"라는 식으로 직전에 몰두하던 일의 맥락에서 아직 벗어나지 못했다는 표현을 할 때가 있다. 이처럼 Context Change 는 집중력과 생산성을 떨어트린다. 질문 잘하는 방법을 미리 인지하고, 불필요한 질문으로 동료 개발자의 Context Change 를 유발하지 말자.

4.1.2. Git-flow 사용 방법

Git-flow 는 일종의 Git 브랜치 전략이다. 쉽게 말해, 개발자들 각자가 작성한 코드를 어떤 방식으로 통합하고 관리할 것인지에 대한 전략 중 하나다. 물론 소속된 팀에서 어떤 브랜치 전략에 쓰냐에 따라 Git-flow 를 사용하지 않을 가능성도 있다. 하지만 Git-flow 를 사용하지 않더라도, Git-flow 에서 변형된 브랜치 전략을 사용할 가능성이 크기 때문에 Git-flow 는 기본적으로 알고 입사하는 것이 좋다.

Git-flow 에 대해서는 아래 2 가지 글을 읽으면 충분히 이해될 것이다.

- 우린 Git-flow 를 사용하고 있어요
- git-flow cheatsheet (번역)

4.1.3. 기본적인 리눅스 명령어

회사와 팀마다 환경이 다르고, 백엔드 개발자가 아닌 경우에는 크게 필요 없을 수 있지만, 기본적인 리눅스 명령어는 알아두고 입사하는 것이 좋다. 백엔드 개발자의 경우 서버에서의 작업도 많기 때문에 기본적인 리눅스 명령어는 반드시 숙지해야 한다.

'리눅스 명령어' 혹은 'linux command' 라고 구글링하면 자료가 넘쳐난다. 개인적으로 만화로 배우는 리눅스 시스템 관리라는 책으로 많이 배웠다. (표지가 조금 유난스러워서 집 안에서만 읽었다) 유익했던 글은 37 Important Linux Commands You Should Know 였고, 인터넷 강의는 Linux Sysadmin Basics 라는 유튜브 강의 하나면 대부분 커버된다. 한국어 강의는 인프런의 리눅스 커맨드라인 툴 이라는 무료 강의가 있다.

입사하면 동료 개발자들은 전공자들일 가능성이 크다. 전공자들에게 기본적인 리눅스 명령어는 '당연히 알아야 하는 것'이기 때문에 비전공자라면 반드시 학습하자.

4.2. 개발자에 도전하며 도움 되었던 마음가짐

29 살에 비전공자로서 개발자의 길에 호기롭게 도전했지만, 사실 두려운 마음도 컸다. 개발자로 취업을 하더라도, 내가 개발을 너무 못해서 동료 개발자들에게 피해를 주거나, 팀 내에서 애물단지가 되는 상상을 하면 포기하고 싶기도 했다.

이따금 간단한 환경설정 문제로 코드 한 줄 작성하지 못하고 주말 하루를 통째로 날릴 때면, 내가 개발자가 될 수 있을지 의문이 들기도 했다. 또한, 학원을 수료할 때까지도 이 실력으로는 개발자로 취업이 되어도 문제라는 생각에 어딘가로 숨어버리고 싶었다.

하지만 포기하고 싶지 않았다. 한 가정의 가장으로서 신혼여행에서 돌아오자마자 백수 신분으로 개발자의 길에 뛰어들었고, 처가에서 살았기 때문에 포기하면 장인 장모님 볼 면목도 없었으며, 무엇보다 스스로에게 부끄럽고 싶지 않았다. 그래서 포기하지 않기 위해 스스로 마음을 다잡으려 노력했는데, 아래 2 가지 마음가짐이 크게 도움 되었다.

4.2.1. This is not rocket science!

직역하면 '이것은 로켓 과학이 아니다.'라는 뜻이지만, 실제로 이 표현을 사용할 때는 '이거 어려운 일 아닌데 (왜 어렵게 해?)'라는 의미로 사용된다. 나는 개발을 하다가 숨 막히는 어려움을 맞이하면 이 문장을 되새김질했다.

우주로 로켓을 쏘는 과학에는 정말 답이 없을 수 있지만, 내 수준에서 만나는 문제들은 반드시 문제의 원인과 답이 있을 거라고 스스로를 안심시켰다. 실제로 울고 싶을 정도로 해결되지 않던 문제도, 막상 이유를 알면 대수롭지 않은 것인 경우가 많았다.

This is not rocket science! 이 문장은 나 자신에게 '별것 아닌 일에 청승 떨지 마라!' 고 다그치는 역할을 한다. 요즘에도 어려운 문제를 맞이하면 마법의 주문을 외우듯이 이 문장을 되새김질하면서 심리적 안정을 찾고, 차분하게 문제를 해결하려고 노력한다.

4.2.1. 먼 산 바라보듯 바라보기

개발 학습을 하거나 개발자로 일하다 보면, 정말 잘하는 개발자들(이하 슈퍼개발자들)을 목격하게 된다. 슈퍼개발자를 목격하면 자연스럽게 내 개발 실력과 비교하게 되었고, '나는 언제쯤 저렇게 할 수 있을까? 나는 벌써 30 대인데…' 라는 생각으로 스트레스를 받곤 했다.

어릴 때부터 개발을 해온 슈퍼개발자들과 나의 실력 차이는 도저히 따라잡을 수 없는 것처럼 느껴졌다. 잘하고 싶은 욕심 때문에 슈퍼개발자가 보일 때마다 스트레스를 받고, 개발 공부가 손에 잡히지 않았다. 그런데 아래와 같이 생각하니 마음이 편해졌다.

- 인정하자. 슈퍼개발자들의 저 경이로운 실력은 그들이 개발에 쏟아온 시간과 노력에 대한 정당한 보상이다.
- 나는 나대로 열심히 살아왔고, 다양한 분야에 도전해온 것은 시간 낭비가 아니라 값진 경험이다.
- 조급해하지 말고, 나와는 관계없다는 생각으로, 먼 산 바라보듯 슈퍼 개발자들을 바라보자. 언젠가 저 산에 오를 수 있다고 믿고, 나만의 속도로 꾸준히 성장하면 된다.

이제는 슈퍼 개발자들을 목격해도, 그저 멀리 보이는 큰 산을 보듯이 바라본다. 정말 멋지고 큰 산이지만, 더 이상 내 삶에 큰 영향을 끼치지 않는 것이다.

4.3. 개발자로서 나의 롤모델

롤모델을 정하고 그들의 경험을 통해 간접적으로 배우는 것은 매우 효율적인 성장 방법이다. 이 글을 읽는 분들처럼, 나도 앞서간 사람들을 열심히 따라 하며 성장했다. 개발을 처음 접해보는 나이 많은 비전공자라면 특히, 앞서간 비전공자를 롤모델로 삼고 그들의 노하우와 시행착오를 디딤돌 삼아 최대한 효율적으로 성장하기를 추천한다.

나는 이동욱 님과 이종립 님을 개발자 롤모델로 삼고, 그분들의 성장 과정과 행보를 꾸준히 참고하고 있다. 내가 롤모델을 선정한 기준은 다음과 같다.

- 나와 비슷한 백그라운드
  - 두 분 모두 비전공자
  - 두 분 모두 Java 백엔드/서버 개발자
  - 두 분 모두 국비지원학원 수료
  - 두 분 모두 첫 회사에서 빠르게 이직
  - 다양한 직업 경험 (이종립 님)
  - 줌인터넷 포털개발팀 출신 (이동욱 님)
- 노력파
  - 두 분은 천재일 수도 있으나, 노력까지 엄청나게 한다.
- 많은 개발자에게 인정받는 개발자
  - 작은 규모의 집단에서만 인정받는 개발자가 아니라, 많은 개발자로부터 인정받는 개발자를 롤모델로 정하는 것이 안전하다.

4.3.1. 이동욱(jojoldu) 님

이동욱 님은 따로 설명이 필요 없을 만큼 유명하고 대단한 개발자가 되셨다. 혹시 이분을 모르는 경우, 지방대 개발 비전공자가 배달의민족 리드 개발자가 되기까지 라는 인터뷰 영상을 시작으로, 기억보단 기록을 이라는 블로그와 깃헙에 방문해보면, 이분의 엄청난 열정에 많은 자극을 받을 것이다. 특히 이동욱 님 블로그의 3 번째 직장에 오기까지 시리즈는 이동욱 님이 국비지원학원에 다니던 시절부터 우아한형제들에 입사하기까지의 과정이 상세하게 설명되어 있어 매우 유익하다.

만약 Java 개발자로 취업할 예정이라면 이분이 출판하신 책으로 학습하며 토이 프로젝트를 만들어보기를 강력하게 추천한다. 나 또한 이 책이 출판되기 전에 블로그 글이었던 당시에, 블로그 글을 따라 하며 AWS 도 처음 사용해보았고, 서버에 프로젝트를 배포한다는 개념도 처음 배웠다.

이분은 몇 년째 일일커밋을 지속해오고 있다. (일일커밋 3 주년 회고) 나도 처음에는 이분을 따라 일일커밋을 시도했었는데, 100 일 정도 만에 끊겼던 기억이 난다. 이분의 꾸준함과 열정은 따라하기도 힘든 수준이고, 내가 살면서 본 대단한 사람 중에서도 손에 꼽는다. 아마 수많은 비전공자들이 이분을 롤모델로 삼고 있을 것이다.

4.3.2. 이종립(johngrib) 님

이종립 님에 대해 알게 된 것은, 이동욱 님이 이분의 세미나에 참여하고 올린 후기를 읽은 후였다. (<비전공 학원출신 SI 개발자, 유명스타트업 들어간.ssul> 참석 후기)

이종립 님은 철학을 전공하시고, 나보다도 더 다양한 직업(운동선수, 학원강사, 보험영업 등)을 경험하셨다. 게다가 31 살에 국비지원학원에서 개발 공부를 시작하셨는데, Vim 을 정말 잘 다루시고 우아한형제들을 거쳐 마켓컬리에서 일하신다.

종립님의 블로그가 대박이다. 위키 형식으로 관리되는 블로그인데, 학습의 진정성과 깊이, 꾸준함이 타의 추종을 불허한다. 이분처럼 효율적으로, 완벽하게 일상생활을 컨트롤하는 사람은 본 적이 없다. 가끔은 이종립 님의 일과를 옆에서 계속 지켜보면서 어떻게 일과 학습과 생활을 병행하시는지 과외라도 받고 싶을 정도다.

종립님의 스타일을 엿볼 수 있는 가벼운 주제의 글들 몇 가지를 추려보았다.

- 2019 년 회고 – 1 년간 읽은 책이 76 권.
- 나의 키보드 히스토리 – 키보드를 이 정도까지 본인에게 맞춘다.
- 빅 오 표기법 (Big O Notation) – 상상을 초월하는 학습 깊이.
- 나의 이메일 관리 방법 – 내 Gmail 도 정리해 주셨으면 좋겠다.
- 목과 허리 건강을 위한 물리적 장비 구매 노하우와 경험 – 갑자기 허리가 아프다.

4.4. 개발자가 된 비전공자들의 글 모음

앞서 개발자가 된 비전공자들의 글을 읽으며, 그들의 성공과 실패의 경험을 관찰하자. 그리고 그들보다 더 빠르고 안정적으로 개발자가 되는 전략을 세우고 실행하자.

- 3 번째 직장에 오기까지 시리즈
- 인문학도 개발자 되다 (책)
- 문돌이가 개발자가 되기까지 시리즈
- 문과생의 카카오 개발자 이직기 시리즈
- 늦은 나이, 개발자로 시작해도 좋을까요 – 30 대 초반 비전공자의 고민
- 문과생 비전공자가 웹 개발자가 되기까지..
- 32 살에 개발에 입문한 비전공자가 인프런 창업한 이야기
- 체대 출신 개발자의 2018 년 회고 (내가 썼던 글)

4.5. 맺으며

(마지막은 경어체로 하겠습니다. ☺)

잘 팔리지도 않을 전자책을 150 시간 넘게 공들여 쓰는, 이 바보 같은 짓이 끝났습니다. 다른 전자책들처럼 20~30 페이지 대충 쓰고, 제목만 자극적으로 지으면 될 것을. 왜 그렇게 하지 못했을까요.

아마도 제가 개발자가 되는 과정에서, 개발자가 된 후에도, 그리고 오늘 회사에서도, 이름 모를 개발자들이 공유해준 정보들로 저의 하루를 채웠기 때문일 것입니다.

앞서나간 개발자들의 사랑 아닌 사랑을 받고 자랐기에, 저도 모르는 사이 빚을 졌다는 마음이 틈틈이 쌓인 것 같습니다. 결과물의 퀄리티는 읽는 사람이 판단할 몫이지만, 제가 진 빚을 갚아 나가는 마음으로 열심히 쏟아냈습니다. 그리고 무료로 공개하지 못한 만큼, 구매하신 분들에게는 반드시 도움 되는 내용을 담으려 노력했습니다.

긴 글 끝까지 읽어주신 것만으로 감사합니다. 이 전자책이 개발자가 되는 과정에 조금이나마 도움 되기를 바랍니다. 언젠가 개발자로서 함께 일하게 되었을 때, 이 전자책 봤었다고 살짝 귀띔해주시면 더없이 기쁠 것 같습니다. ☺

궁금한 점 있으신 분들은 리뷰에 질문 남겨주시면 답변 드리겠습니다.
언제나 건강하시고, 원하는 바 모두 이루시길 바라겠습니다. 감사합니다!