JAK ZOSTAĆ PROGRAMISTĄ?

KROK PO KROKU

KAMIL BRZEZIŃSKI

Dawno już minęły czasy, kiedy programowanie wydawało się być dostępną tylko dla wybranych, tajemną wiedzą, a sami programiści kojarzyli się z ubranymi we flanelowe koszule i nigdy nie wychodzącymi z piwnicy mrukami. Teraz programowanie jest modne, programowanie jest na topie i chce się go uczyć coraz więcej osób. Nic dziwnego, bo praca jest ciekawa, jest jej dużo, a zarobki kuszą i wszystko wskazuje na to, że taka sytuacja utrzyma się jeszcze długo.

Duża część osób chcących nauczyć się programowania napotyka jednak problem już na samym początku – cały czas jak bumerang wracają bowiem takie pytania jak:

- Jak i gdzie nauczyć się programowania?
- Czy trzeba iść na studia?
- Czy można zostać programistą kończąc kilkumiesięczny bootcamp albo robiąc kursy online?
- Czy warto czytać książki o programowaniu?
- Jaki język programowania wybrać na początek?
- Co trzeba umieć, żeby dostać pierwszą pracę w IT?
- Czego spodziewać się na rozmowie kwalifikacyjnej?

Odpowiedzi na nie znajdziesz w tym ebooku.

Różne drogi do zostania programistą

Studia

Studia to wciąż najbardziej oczywisty i najpopularniejszy sposób na zostanie programistą. I pomimo coraz częściej powtarzanych opinii, że studia to bardzo dużo teorii i archaicznej wiedzy, to wciąż pod wieloma względami **najlepszy możliwy wybór**.

Oczywiście, podczas studiów będziecie musieli nauczyć się wielu rzeczy, które w pracy Wam się prawdopodobnie nigdy nie przydadzą i spotkacie na swojej drodze sporo wykładowców - teoretyków, którzy nigdy nie pracowali w branży IT i są od tej rzeczywistości dość mocno oderwani, ale **studia dadzą Wam zdecydowanie więcej niż tylko umiejętność kodowania**. Zdobędziecie bardzo przekrojową, kompleksową wiedzę z zakresu nie tylko programowania, ale całej informatyki.

To również **bardzo dobry wybór, jeżeli macie problem z sumiennością i systematycznością** - regularne kolokwia, egzaminy i projekty mobilizują do nauki nawet najbardziej opornych. I trzeba naprawdę mocno się postarać, żeby po trzech i pół roku studiów inżynierskich (a tym bardziej po pięciu latach studiów magisterskich, jeżeli się na nie zdecydujecie), nie opanować umiejętności programowania na choćby przyzwoitym poziomie.

Studia dają także **możliwości wykraczające daleko poza samą naukę** i nie mam tutaj na myśli weekendowych szaleństw i tego legendarnego studenckiego życia. To dzięki studiom - w ramach programu Erasmus - mamy możliwość **wyjazdu na uczelnię do innego kraju**, a także skorzystania z **ofert zagranicznych staży i prak-**

tyk oferowanych przez instytucje takie jak na przykład IAESTE. i są to zazwyczaj niezapomniane i bardzo istotne doświadczenia - osobiście uważam, że moje **roczne pobyty w Danii (Erasmus) i Stanach Zjednoczonych (praktyki z IAESTE)** były jednymi z najważniejszych okresów w moim życiu, które bardzo mocno procentują do dnia dzisiejszego.

Ostatnia, nie mniej ważna kwestia, to **networking, jaki oferują studia** - to właśnie wtedy poznajemy całą masę osób, z którymi już za kilka lat będziemy współtworzyć branżę IT. Mamy okazję wzajemnie się inspirować, nawiązywać znajomości i tworzyć wspólne projekty. W Polsce wciąż czekamy na pierwszy technologiczny startup, który stanie się jednorożcem, czyli firmą warto ponad miliard dolarów. Kto wie, być może wybierając się na studia poznasz tam osoby, z którymi uda Ci się taki startup stworzyć?

Bootcampy i szkolenia

Rosnąca popularność programowania sprawiła, że jak grzyby po deszczu wyrastają kolejne szkoły programowania oferujące coraz to bardziej wymyślne kursy, szkolenia i bootcampy. Charakteryzują się one **dużą intensywnością i nastawieniem na praktyczną wiedzę** - głównym założeniem tego typu szkoleń jest przygotowanie w ciągu kilku miesięcy absolutnego nowicjusza do pracy w zawodzie. i jest to świetny wybór dla świadomych ludzi, którzy dokładnie wiedzą czego chcą i są w stanie przez tych kilka miesięcy naprawdę dać z siebie wszystko. **Uczyć się po kilka godzin dziennie, eksperymentować, kodować, robić własne projekty**.

Dużą zaletą szkoleń jest również fakt, że trenerzy to często osoby z doświadczeniem zawodowym, na co dzień realizujące komercyjne

projekty w branży IT, a zatem zazwyczaj będące na bieżąco z aktualnymi trendami na rynku.

Należy jednak zachować **bardzo dużą ostrożność** przy wyborze szkoły - niestety bardzo wiele z nich wykorzystuje dobrą koniunkturę na rynku i nieświadomość potencjalnych klientów, oferując szkolenia, w których obiecuje prawdziwe cuda. Postawmy sprawę jasno - **trzy miesiące to nie jest czas wystarczający do zostania programistą**. Absolutnie minimalny czas do zostania Junior Developerem - zakładając, że mamy pewną dozę talentu i jesteśmy gotowi na bardzo intensywną naukę, po kilka, a czasem nawet kilkanaście godzin dziennie - to pół roku, chociaż w większości przypadków znacznie **bardziej realnym terminem będzie okres pomiędzy 9 a 12 miesiącami**. Przeglądając oferty szkół programowania widziałem szkolenia trwające pięć tygodni. Nie muszę chyba dodawać, że pieniądze wydane na takie szkolenie prawdopodobnie nie będą najlepszą inwestycją w życiu.

Właśnie, pieniądze. Kursy i szkolenia to niestety **droga zabawa** - decydując się na wybranie tej opcji musimy liczyć się z wydatkiem pomiędzy 8 a 12 tysięcy złotych. Większość szkół pozwala jednak rozłożyć płatność na raty.

Nauka na własną rękę - książki i szkolenia online

Jeżeli jesteśmy wystarczająco zmotywowani i zdeterminowani, a do tego potrafimy się właściwie zdyscyplinować, **możemy nauczyć się programowania na własną rękę**. Pierwszą opcją, jaka przychodzi na myśl to stare, dobre książki, które pewnie jeszcze długo będą stanowiły świetne źródło wiedzy i są doskonałym wyborem jeżeli chcemy nauczyć się jakiejś konkretnej, nowej umie-

jętności.

Jednak jeżeli mówimy o samym początku drogi - od nowicjusza do pierwszej pracy jako młodszy programista - to tutaj o ich skuteczności nie jestem już tak bardzo przekonany. Na tym etapie, nie mając odpowiedniego doświadczenia, **trudno jest filtrować wiedzę z książek**, trudno jest spośród materiału z kilkunastu książek wybrać te elementy, które są faktycznie istotne do zostania Junior Developerem.

Żebyśmy się jednak dobrze zrozumieli - książki są szalenie ważne i **gdy już będziemy mieli pewne doświadczenie, warto czytać je regularnie**. Świetnym wyborem, gdy chcemy nauczyć się programowania sami, są kursy online. Internet oferuje dostęp do dziesiątek bardzo rzetelnie przygotowanych (i stale aktualizowanych) kursów dopasowanych do każdego poziomu doświadczenia. Ten sposób daje nam dużą elastyczność - **możemy uczyć się we własnym tempie i wybierać tematy, które nas interesują**. Jeżeli jednak potrzebujemy pomocy w doborze tematów, też nie zostaniemy pozostawieni sami sobie - w serwisach takich jak Pluralsight, Codecademy czy Teamtreehouse na początku rozwiązujemy test, na podstawie którego oceniane jest nasze doświadczenie, a następnie pod tym kątem otrzymujemy listę rekomendowanych kursów.

Kursy online to również **bardzo atrakcyjny stosunek ceny do oferowanych możliwości** - na Udemy pojedynczy kurs to wydatek kilkudziesięciu złotych, wspomniane Pluralsight, Codecademy i Teamtreehouse to miesięczny abonament poniżej 150 zł miesięcznie.

Jak się uczyć programowania?

Ucząc się programowania, możemy wybrać dwie drogi . Pierwszą z nich jest podejście nazwane *top-down* - bierzemy jakiś tutorial, przechodzimy go krok po kroku i w ten sposób bardzo szybko realizujemy konkretny projekt. Druga opcja to podejście *bottom-up*, w którym każdy krok w świecie programowania stawiamy bardzo dokładnie, kompleksowo ucząc się teorii i mozolnie dokładając kolejne klocki w naszej edukacji. Jak nietrudno się domyślić i jak to zazwyczaj w życiu bywa, **oba podejścia mają swoje dobre i złe strony**, i wcale nie jest tak łatwo zdecydować się na któreś z nich. A może da się oba te podejścia połączyć?

Wracam czasem myślami do czasów, kiedy napisałem swój pierwszy program w życiu. Miałem wtedy czternaście lat i właśnie rozpocząłem naukę w gimnazjum. Lubiłem komputery i ogarniałem je całkiem nieźle, miałem też już za sobą pierwsze stworzone strony w HTML, ale daleko mi było do takiego typowego komputerowego geeka. w wolnym czasie zamiast siedzieć przed monitorem zdecydowanie bardziej wolałem grać w siatkówkę, koszykówkę i piłkę nożną, a o programowaniu kompletnie nie myślałem. Przejrzałem co prawda kilka odcinków kursu programowania w Pascalu, który prowadzony był w prenumerowanym przeze mnie kultowym magazynie CD-ACTION, ale przedstawiane w kursie problemy - takie jak liczenie silni czy kolejnych wyrazów ciągu Fibonacciego - niespecjalnie mnie ekscytowały. i pewnie długo nie napisałbym swojego pierwszego programu, gdyby nie fakt, że na niektórych przedmiotach wprowadzona została tak zwana średnia ważona. Wiecie o co chodzi, oceny z klasówek mnożone były razy trzy, te z kartkówek razy dwa, a pozostałe oceny zostawiane były z pojedynczym mnożnikiem. a to oznaczało strasznie dużo pracy przy liczeniu oceny

z danego przedmiotu. Trzeba było dodawać, mnożyć, potem znowu dodawać, dzielić, komu by się to wszystko chciało robić? i wtedy właśnie pomyślałem "a może by tak napisać program, który to wszystko zrobi za mnie?". Wtedy nie wiedziałem jeszcze, że istnieje powiedzenie, że "dobry programista to leniwy programista". Wiedziałem jedynie, że jestem piątkowym uczniem, który nie lubi się przepracowywać i to podejście sprawiło, że postanowiłem ułatwić sobie życie i zautomatyzować tę przeraźliwie nudną czynność liczenia średniej. Wziąłem do ręki kilka odcinków kursów, ściągnąłem kompilator Turbo Pascala i napisałem swój pierwszy kod! Działał. i liczył średnią. Nie był to oczywiście nadzwyczaj skomplikowany program, ale dla mnie było to coś naprawdę wyjątkowego. Wtedy **po raz pierwszy poczułem tę satysfakcję i radość pojawiające się, gdy rozwiążemy jakiś problem**, gdy uda nam się stworzyć coś z niczego. Wydawać by się mogło, że to jest ten moment, kiedy pójdę za ciosem, kiedy bez reszty wciągnę się w programowanie i będę pisał jeden program za drugim, ale tak się wcale nie stało. Do programowania - tym razem w PHP i MySQL - wróciłem dopiero po trzech, czterech latach, gdy magazyn muzyczny, z którym współpracowałem uzyskał nową szatę graficzną, a ja podjąłem się stworzenia CMS-a pozwalającego na zarządzanie treściami dodawanymi do serwisu.

Szybkie efekty czy mozolna praca?

Dlaczego w międzyczasie nie czułem potrzeby dalszego rozwoju w dziedzinie programowania? Dlaczego przez ten czas nie napisałem ani jednej linii kodu? Odpowiedź jest prosta - **brakowało mi problemów do rozwiązania**, a co się z tym wiąże także satysfakcji płynącej z ich rozwiązania, satysfakcji napędzającej do dalszej pracy i dalszej nauki. a powiedzmy sobie szczerze - jakie problemy

wymagające pisania kodu może mieć czternasto- czy piętnastolatek?

I właśnie ta satysfakcja i radość, o których po raz kolejny wspominam, to moim zdaniem **klucz do sukcesu**, główny powód, dla którego człowiek chce się uczyć programowania i zgłębiać kolejne zagadnienia. z tego powodu **jestem ogromnym zwolennikiem rozpoczynania nauki programowania od rzeczy efektownych**, takich, w których efekty naszej pracy widoczne są bardzo szybko. Jeżeli pokazując komuś programowanie, wybierzemy mozolną drogę od absolutnych podstaw, tłumacząc najpierw czym jest system dziesiętny i system binarny, potem przechodząc do różnic pomiędzy stosem a kolejką, a następnie opowiadając o charakterystykach typów danych w wybranym języku, istnieje bardzo duże ryzyko, że nasz uczeń / słuchacz / widz, bardzo szybko się zniechęci i uzna, że programowanie jest:

a) bardzo trudne

b) bardzo nudne

A przecież ani punkt a ani (zwłaszcza!) punkt b nie są prawdziwe.

Wybierając drugą ścieżkę - szybko pokazując fajne efekty, które można uzyskać pisząc kod - mamy dużą większą szansę na zaszczepienie w drugiej osobie prawdziwego bakcyla do programowania. Oczywiście nie jest to warunek konieczny do zostania programistą - możliwe, że są osoby, które uczą się programowania tylko dlatego, że słyszały o wysokich zarobkach w branży IT i jest to ich jedyna motywacja - ale jednak **dużo łatwiej o satysfakcjonujące efekty, gdy nauce towarzyszy autentyczna radość**. i teraz bardzo ważne, żebyśmy nie zrozumieli się źle - wszystko to, o czym

wspominam w tym akapicie jest niezmiernie ważne. **Każdy programista powinien potrafić bez najmniejszego problemu przeliczyć liczby pomiędzy systemem dziesiętnym a binarnym**, ani przez chwilę nie powinien mieć wątpliwości czym jest stos a czym kolejka, a zapytany o to, kiedy używamy zmiennej typu long, a kiedy wystarczy int, poprawnej odpowiedzi powinien udzielić bez najmniejszego zająknięcia.

Nie oglądaj się za siebie? Wprost przeciwnie!

Dlatego uważam, że oba podejścia - *top-down* i *bottom-up* - należy nieustannie, przez cały czas, gdy uczymy się programowania, ze sobą łączyć. Jak to może wyglądać w praktyce? Powiedzmy, że mamy do zaimplementowania jakąś nową funkcjonalność albo chcemy nauczyć się nowego frameworku. Nie ma nic złego, jeżeli otworzymy odpowiedni tutorial, wykonamy po kolei opisane w nim kroki i osiągniemy oczekiwany przez nas rezultat, nie rozumiejąc wszystkiego, co robiliśmy po drodze. Bo to jest właśnie ten moment, kiedy **trzeba te zaległości nadrobić - zrobić kilka kroków w tył**, zatrzymać się przy pewnych zagadnieniach, doczytać niektóre tematy w dokumentacji.

Nauka programowania nie jest drogą jednokierunkową, w której nigdy nie oglądamy się za siebie. Czasem pewne rzeczy dobrze jest zrobić szybciej, bez niepotrzebnego zatrzymywania się w jakimś miejscu, po to, by szybciej osiągnąć końcowy cel i po to, by szybciej pojawiła się związana z tym satysfakcja. Ale to nie zwalnia nas z obowiązku robienia tych rzeczy dobrze. Jeżeli na którymś etapie odrobinę się pospieszyliśmy, na moment przycisnęliśmy mocniej gaz, w miejscu gdzie było ograniczenie prędkości, teraz **poświęćmy trochę czasu na naukę podstaw**. Jeżeli to zaniedbamy, w przyszło-

ści zapłacimy za to bardzo wysoką cenę.

Ucz się iteracyjnie!

O nauce programowania lubię myśleć podobnie jak o tworzeniu oprogramowania. w dzisiejszych czasach w dziedzinie zarządzania projektami **prym wiodą metodyki agile'owe czyli metodyki zwinne, w których oprogramowanie wytwarza się w sposób iteracyjny**, przyrostowy. Dla osób, które nie pracowały jeszcze w branży IT szybkie wytłumaczenie - chodzi o to, by z każdą kolejną iteracją dostarczać klientowi pełnoprawny, funkcjonalny produkt i nie dopuścić do sytuacji, kiedy klient mógłby usłyszeć z naszych ust słowa "pracowaliśmy nad tym miesiąc, ale tak naprawdę wszystko rozgrzebaliśmy i póki co nic nie działa". Zamiast rozpoczynać więc pracę nad kilkoma funkcjonalnościami naraz czy próbując wprowadzać bardzo duże zmiany od razu, staramy się wszystko dzielić na mniejsze fragmenty i z każdą kolejną iteracją dokładać małe cegiełki do całości. Czasem jedną funkcjonalność dzieli się na kilka mniejszych, które możliwe są do zaimplementowania jedna po drugiej, tak by w każdym momencie mieć działający i gotowy do zaprezentowania klientowi produkt.

Uważam, że **podobne podejście doskonale sprawdza się podczas nauki programowania**. Nie chodzi o to, żeby wziąć temat a i przerobić go od deski do deski i dopiero potem zabrać się za temat B. w kolejce przecież czekają już tematy C i D, rekrutująca nas osoba chętnie poznałaby naszą opinię na temat tematu E, a w wymaganiach wyraźnie napisane było, że znajomość tematów F i G też jest mile widziana. To co ja proponuję to poznać trochę tematu A, potem przejść do tematu B, wybrać to co jest nam na ten moment potrzebne, a potem zobaczyć czym jest temat C. Gdy poznamy pod-

stawy każdego z interesujących nas tematów i użyjemy tej wiedzy w praktyce - na przykład tworząc prosty projekt - będzie to idealny moment, by wrócić do tematu a i tym razem zgłębić go nieco dokładniej. a potem, w kolejnej iteracji pochylić się nad nim jeszcze mocniej, jednocześnie dokładając kolejne tematy. w ten sposób **w naturalny sposób będziemy zdobywać przekrojową i bardzo konkretną, przydatną w praktyce wiedzę**.

Praktyka, praktyka i jeszcze raz praktyka

Przed chwilą pojawiło się bardzo istotne słowo. Słowo - klucz. Praktyka. **I to właśnie praktyka jest najważniejszym konceptem w całej sztuce nauki programowania**. Bez praktyki, bez pisania kodu, bez faktycznego rozwiązywania problemów, pojawiających się podczas pracy z kodem, nie zostaniesz programistą. Możesz przeczytać dziesiątki książek i obejrzeć setki godzin tutoriali, ale bez pisania kodu, bez tworzenia własnych projektów, będzie to czas stracony. Dlatego twórz dużo, koduj dużo, realizuj projekty (*top-down*), nie zapominając jednocześnie o tym, by na żadnym etapie nie zaniedbywać podstaw (*bottom-up*). A potem zapętl ten cykl i **pracuj ciężko, cierpliwie i wytrwale**. Radość i satysfakcja są tego warte.

Co musisz umieć, żeby dostać pierwszą pracę jako programista?

Pomimo tego, że znaczna część firm IT w Polsce nieustannie poszukuje programistów, nie oznacza to, że poszukuje jakichkolwiek programistów. Jeżeli chcemy otrzymać pierwszą pracę na stanowisku programisty, nie musimy oczywiście być ekspertami, ale każdy potencjalny pracodawca będzie od nas oczekiwał dwóch rzeczy - solidnych podstaw oraz potencjału do szybkiego rozwoju. Co to oznacza w praktyce?

Po pierwsze, **powinniśmy sobie nieźle radzić z logicznym myśleniem i rozwiązywaniem prostych problemów algorytmicznych**. Świetne ćwiczenia znajdziecie na przykład na https://app.codility.com/programmers/ czy https://www.hackerrank.com/.

Po drugie, **konieczne jest opanowanie podstaw wybranego języka programowania**. Tutaj od razu pojawia się pytanie - **jaki język wybrać na początek nauki?** Jeżeli mówimy o absolutnym początku nauki - **nie ma to większego znaczenia**. Języki różnią się między sobą składnią (czyli sposobem w jaki piszemy w nich instrukcje), ale ogólne zasady programowania i logiki są zawsze takie same niezależnie od języka, w którym rozwiązujemy dany problem. Także pierwsze 2-3 miesiące przygody z programowaniem to w zasadzie dowolny język, chociaż warto wybrać taki, który jest popularny, powszechnie lubiany i łatwy do nauki. **Moje rekomendacje to Python, Java lub JavaScript**, ale jeżeli ktoś wybierze C# czy PHP również nie powinien narzekać. Co istotne, dzisiaj nie trzeba nawet niczego instalować, a potem konfi-

gurować, żeby zacząć naukę programowania - dzięki takim stronom jak https://www.hackerrank.com/, https://playcode.io/ czy https://repl.it/ możemy pisać kod otwierając tylko przeglądarkę internetową i wchodząc na wybraną stronę.

Gdy już opanujemy podstawy sztuki programowania, bez wahania odpowiemy czym jest pętla, czym jest instrukcja warunkowe, jakie są najpopularniejsze typy i struktury danych wykorzystywane w językach programowania, a tę wiedzę będziemy potrafili wykorzystać do pisania prostych programów, przyjdzie **pora na już bardziej świadomy wybór języka, którego tajniki będziemy zgłębiać przez kolejne miesiące**. Obecnie na rynku **najwięcej ofert pracy dla młodszych programistów dotyczy języków Java oraz JavaScript**.

Niezależnie od wybranego języka, musimy również opanować **podstawy relacyjnych baz danych oraz powiązanego z nimi języka SQL**. Tutaj też nie musimy kształcić się na ekspertów (na to jeszcze przyjdzie czas!), ale proste operacje na bazie (dodawanie, odczytywanie, aktualizowanie, usuwanie rekordów), wiedza czym są klucze główny i obcy, a także złączenia tabel (tzw. joiny) to tematy absolutnie niezbędne, które po prostu trzeba znać.

Kolejnym **bardzo istotnym zagadnieniem jest REST API**. Nie jest to żadna konkretna technologia, ale styl w architekturze oprogramowania, będący współcześnie standardem w pisaniu aplikacji sieciowych. Powinniśmy znać pojęcie web serwisu (ang. *web service*) oraz potrafić takie web serwisy tworzyć, na przykład z wykorzystaniem frameworka Spring Boot czy środowiska Node.js.

Również **temat testowania aplikacji nie powinien być nam ob-**

cy. Powinniśmy znać różnice między testami jednostkowymi (ang. *unit tests*), integracyjnymi, a akceptacyjnymi, dodatkowym atutem będzie znajomość podejścia **Test-Driven Development** poparta doświadczeniem w pisaniu unit testów.

Poza znajomością wybranego języka, **dobrze jest też się pochwalić znajomością frameworków i narzędzi powiązanych z daną technologią**. w świecie Javy standardem jest używanie frameworków Spring i Hibernate, w przypadku JavaScript jest to Node.js dla backendu i frameworki React.js, Angular i Vue.js w przypadku frontendu.

Każdy programista powinien również znać **zasady pisania dobrego, czytelnego i przejrzystego kodu**. w tym celu warto zapoznać się z ideą wzorców projektowych oraz zbiorze zasad programowania obiektowego nazwanych SOLID, ale na samym początku drogi wystarczy zacząć od dwóch doskonałych książek: *Pragmatyczny programista. Od czeladnika do mistrza* oraz *Czysty kod. Podręcznik dobrego programisty*.

W całym procesie edukacji **najważniejsza jest jednak praktyka, także wiedzę, którą zdobywamy, powinniśmy wykorzystywać do tworzenia własnych projektów**. Nie musi to być oczywiście nic skomplikowanego, ale istotne jest, żeby wszystko, czego się uczymy - nieważne czy mówimy o bazach danych czy o REST API - testować w praktyce, eksperymentować i pisać jak najwięcej kodu.

Żeby zostać programistą - trzeba programować.

A żeby zostać dobrym programistą - trzeba programować jeszcze więcej.

Jak przygotować się do rozmowy kwalifikacyjnej na Junior Developera?

Rozmowa kwalifikacyjna to dla większości osób stresujący moment, zwłaszcza w sytuacji, gdy mówimy o początku kariery i pierwszej pracy. Samo powiedzenie "nie stresuj się" w tym momencie może nie być wystarczające, ale należy pamiętać o kilku bardzo istotnych kwestiach:

1. Nawet jeżeli się nie uda, nie denerwuj się, bo po pierwsze to nie jest koniec świata, na rynku jest wiele innych firm, a po drugie **jest wiele przyczyn, dla których odpowiedź po rozmowie może być negatywna**:

    - twoja wiedza nie jest wystarczająca na stanowisko, na które aplikowałeś
    - poszło ci dobrze, ale byli inni, jeszcze lepsi kandydaci
    - niewłaściwie przedstawiłeś swoją wiedzę, nie udowodniłeś w wystarczający sposób, że nadajesz się na to stanowisko
    - twoja wiedza była na dobrym poziomie, ale nie dogadaliście się na poziomie osobowościowym, ludzie są różni, czasem jedna osoba zupełnie nie pasuje do jakiegoś zespołu, ale świetnie odnajduje się w innym

    Także rozmowa może pójść ci źle, **czasem to będzie twoja wina, ale czasem nie wszystko będzie zależało od ciebie**, dlatego nie należy brać porażki zbyt mocno do siebie.

2. Należy wyciągnąć wnioski, potraktować rozmowę jako feedback, jako informację, czego jeszcze musimy się nauczyć.

3. Warto iść na rozmowę, żeby mieć ten feedback, jeżeli się nie uda, to możemy spróbować znowu za parę miesięcy. Takie firmy jak Google, Amazon czy Apple mają politykę, że po nieudanej rekrutacji możemy aplikować po 18 miesiącach, ale do większości firm możemy odezwać się już po pół roku i spróbować ponownie, także warto chodzić na rozmowy kwalifikacyjne już po kilku miesiącach nauki.

4. Rozmowa kwalifikacyjna to - jak sama nazwa wskazuje - **rozmowa, a nie test**. Co dokładnie kryje się za tym pojęciem, wyjaśnię już za kilka chwil.

Podczas rozmowy - początek

Opowiedz kilka słów o sobie - gdzie pracowałeś wcześniej, czym się zajmowałeś, nad jakimi projektami pracowałeś. W przypadku aplikowania na Junior Developera **możemy nie mieć doświadczenia zawodowego, ale to nie oznacza, że nie mamy o czym opowiedzieć**. Warto opowiedzieć, jak się czujemy w programowaniu, jakie są nasze mocne strony, nad czym pracujemy obecnie, nad czym chcemy pracować i w jakim kierunku się rozwijać.

Należy opowiedzieć o projektach, nad którymi pracowaliśmy, co nam się w nich podobało, co było największym wyzwaniem i **jak sobie z tymi wyzwaniami radziliśmy**. Przygotuj się do tego dobrze, upewnij się, że potrafisz opowiedzieć o tych projektach, użytych technologiach oraz zastosowanych rozwiązaniach - jeżeli na przykład w projekcie użyta została baza nierelacyjna, może paść pytanie o powód, dlaczego nie była to baza SQLowa, jeżeli użyty został Spring Boot, może paść pytanie, jakie są zalety korzystania ze Spring Boota, itd. W skrócie - **powinniśmy mieć wiedzę na temat**

technologii, z którymi pracowaliśmy i powinniśmy umieć o nich opowiadać.

Warto też pamiętać, że rekruter zazwyczaj będzie najbardziej zainteresowany nie tym, co szło łatwo, ale tym, co sprawiało nam trudność - **programista musi rozwiązywać problemy** i to w jaki sposób je rozwiązuje jest bardzo ważne. Nie sztuką jest tworzyć rzeczy łatwe, **sztuką jest radzić sobie z wyzwaniami i rozwiązywać skomplikowane problemy.**

W tym wszystkim bardzo istotne jest, że suche fakty rekruter poznał już czytając nasze CV, **podczas rozmowy kwalifikacyjnej należy zainteresować go historią** i opowiedzieć o naszych dotychczasowych doświadczeniach w jak najbardziej atrakcyjny sposób. Żebyśmy jednak właściwie się zrozumieli - **historia ma być uzupełnieniem naszej solidnej, konkretnej wiedzy**, bo to jest podstawa, której nie zastąpią nawet najpiękniejsze i najbardziej wymyślne słowa i opowieści.

Możesz również spodziewać się, że zostaniesz zapytany, dlaczego chcesz dołączyć akurat do tej firmy i co skłoniło cię do zaaplikowania na dane stanowisko. Dobrze jest więc wiedzieć, czym zajmuje się firma, do której aplikujemy i dlaczego chcemy w niej pracować.

Rozmowa - część techniczna

Część techniczna to zdecydowanie **najważniejsza część rozmowy kwalifikacyjnej**. To w tym momencie rekruter będzie zadawał Ci prawdopodobnie najtrudniejsze pytania i sprawdzał poziom twojej wiedzy. Każda rozmowa kwalifikacyjna jest inna, ale jest kilka typów pytań, których możesz spodziewać się na większości rozmów:

1. Proste problemy algorytmiczne do rozwiązania - sprawdź czy dwa prostokąty się przecinają, znajdź dwie największe liczby w tablicy, znajdź zduplikowane wartości w tablicy, odwróć kolejność liter w słowie, sprawdź czy dwa słowa są anagramami, sprawdź jaka litera w tekście występuje najczęściej, itd.
2. Pytania dotyczące struktur danych - list, tablic, map
3. Pytania dotyczące ogólnej znajomości języka - w przypadku Javy możesz zostać zapytany czym jest JVM, czym się różni JRE od JDK, jakie są dwa typy wyjątków, czym są modyfikatory dostępu, jakie są modyfikatory dostępu dla pól, a jakie dla klas, czym jest przeciążanie (overriding) i przeładowywanie (overloading) metod, jaka jest różnica między interfejsem a klasą abstrakcyjną, jak działa garbage collector
4. Pytania dotyczące dobrych praktyk w tym języku
5. Podstawowe operacje bazodanowe w SQL

Dobrym pomysłem jest **zapoznanie się z przykładowymi pytaniami z rozmów kwalifikacyjnych** - w tym celu warto zapytać wujka Google o takie tematy jak:

- Java interview questions
- SQL interview questions
- Junior Developer interview questions

Dodatkowymi atutami będzie znajomość wzorców projektowych, zasad pisania czystego kodu (Clean Code) oraz dobrych praktyk w programowaniu obiektowym SOLID.

A czy ty masz jakieś pytania?

Gdy część techniczna dobiegnie końca, **zazwyczaj zostaniesz zapytany, czy ty również masz jakieś pytania**. W wielu poradnikach dotyczących rozmów kwalifikacyjnych przeczytasz, że koniecznie musisz w tym momencie zadać jakieś pytanie, żeby pokazać swoje zainteresowania i zaangażowanie, ale osobiście nie rozumiem tej presji. Wspominałem, że rozmowa kwalifikacyjna to rozmowa, a nie test, także prawdopodobnie zdążysz zapytać rekrutera o wiele rzeczy już na wcześniejszym etapie i naprawdę nie musisz w tym momencie na siłę wymyślać kolejnych pytań. Rzeczy, o które według mnie warto pytać są następujące:

1. Jeżeli jest to rozmowa na konkretny projekt, **warto dopytać o szczegóły projektu**, jak jest prowadzony, jakie technologie są w nim wykorzystywane. Jeżeli jest to rozmowa niezwiązana z konkretnym projektem i aplikujemy ogólnie do firmy, warto zapytać jakie projekty są prowadzone w firmie, przy jakich projektach jesteśmy brani pod uwagę.

2. Jak wygląda zarządzanie projektami w firmie? W jakiej metodyce prowadzone są projekty? Jak duże są zespoły projektowe?

3. Jak wygląda kwestia rozwoju? Czy firma ma budżet szkoleniowy, albo na wyjazdy na konferencje, szkolenia, certyfikaty albo na kursy online (Safari, O'Reily, Pluralsight)? Czy ma budżet na książki?

Wymagania finasowe

Są takie głupie powiedzenia jak *człowiek uczy się na błędach, co nas nie zabije to nas wzmocni,* ale wszystkie te powiedzenia to nic

w porównaniu z absolutną głupotą pod tytułem **dżentelmeni nie rozmawiają o pieniądzach**. Nie wiem, kto jest autorem tej bzdury, ale **o pieniądzach zdecydowanie warto rozmawiać**. Warto też wiedzieć, jak o nich rozmawiać.

Co prawda obecnie większość firm w ofercie podaje widełki, więc aplikując na stanowisko, automatycznie je akceptujesz, ale jeżeli widełek nie ma, możesz zostać zapytany o twoje wymagania finansowe. Z tego powodu jeszcze przed wysłaniem CV należy zrobić rozeznanie rynku - **musisz wiedzieć, jakie są stawki w danym mieście na stanowisku, na które aplikujesz**. Stawka, którą podasz musi być rozsądna - jako Junior Developer nie możesz oczekiwać 10.000 złotych brutto, ale z drugiej strony jeżeli rzucisz kwotę w okolicach 3.000, słusznie wzbudzisz podejrzenia, że coś jest nie tak. **Nie można zaniżać swojej wartości** i poddawać w wątpliwość swoich umiejętności, a tym z całą pewnością jest podawanie wymagań finansowych zdecydowanie poniżej stawek rynkowych.

Po rozmowie

Bardzo istotne jest, żeby po każdej rozmowie kwalifikacyjnej wyciągnąć odpowiednie wnioski. W tym celu:

1. Zapisz wszystkie pytania, które pamiętasz i przeanalizuj je dokładnie
2. Zastanów się co podczas rozmowy sprawiło ci najwięcej trudności
3. Jeżeli uważasz, że rozmowa poszła źle, zastanów się jaka jest tego przyczyna

Pamiętaj też, że możesz trafić na niewłaściwych, niekompetentnych ludzi, którzy nie potrafią właściwie przeprowadzać rozmów kwalifikacyjnych. Zdarzają się ludzie, którzy zamiast rozmawiać, będą zachowywali się arogancko, zamiast rozmawiać będą cię przepytywać jak na egzaminie, może zdarzyć się, że będą zadawali ci absurdalne pytania. Takimi rozmowami się nie przejmuj, ale z każdej rozmowy wyciągaj wnioski i staraj się, żeby każda kolejna była coraz lepsza.

I przede wszystkim - **nie bój się chodzić na rozmowy kwalifikacyjne**. Jeżeli chcesz zostać programistą, jest to punkt, którego zdecydowanie nie możesz pominąć :)

Linki i kontakt

YouTube:
https://www.youtube.com/JakNauczycSieProgramowania

www:
https://JakNauczycSieProgramowania.pl/

Grupa na Facebooku:
https://www.facebook.com/groups/693124164480151/

mail:
kamil@jaknauczycsieprogramowania.pl