

# Technologist Interviews

A guideline for technologist job interviews

## Interview Process



At modanisa, we have multiple steps of technologist interview. We see this as a way of knowing our candidates better. We want to make sure that we can work together as well as live and share together. There might be **two** different **positive** outcomes of an interview;



1- Candidate might get an above average score, **and** candidate is a **curious & enthusiastic** technologist, we offer to candidate a grade senior or above depending how well they have done.



2- Candidate might get a below average score, **and** candidate is a **curious & enthusiastic** technologist, we offer to candidate a grade around junior/entry level grade.




We **don't take** past experiences, diploma, having graduated from a reputable university **into account** while offering. We want to make our interview process perfect enough to do all these assessments based on **empirical evidences**. We don't really look at in CV. We care more about being a hard core technologist by closely following developer communities, attending them, speaking at them, writing blog posts about technology, contributing to open source projects... You get the idea. We are open source lovers. We want our colleagues to use open source code as well as contribute back.

There might be several **negative** outcomes of an interview;



1- Candidate is **not** a **curious** nor a **passionate** technologist. Learning and experimenting is in our DNA. We can't think of a technologist without passion and curiosity.



2- Candidate is a **curious** and a **passionate** technologist. But **does not have** even the basic understanding of **computer science**, we recommend them to **skydome academy** where we guide enthusiastic technologists to a self-taught learning experience. We are following [teachyourselfcs.com](https://teachyourselfcs.com)  as hard core computer science agenda. And we support this agenda with **software delivery** mindset with **running lean** style **product development** culture.

### We have 3 steps of a technologist interview

- 1 | Codility Test
- 2 | Assignment
- 3 | Pair Programming & Computer Science

Let's talk about steps briefly:

## But first:

---



<https://en.wikipedia.org/wiki/RTFM> 

## 1- Codility Test

---

This is an online test assessing candidate's basic skills on following topics;

- A very simple frontend test
- An algorithm design question
- A simple linux terminal challenge
- A multi-choice general Computer Science test

## 2- Assignment

---

Web Based ToDo List Application

## 2.1 Abstract

---

We want you to **build a simple ToDo list application** including all the phases of building a real software project. You can use *any technology* to implement any requirement of this assignment. Mentioned phases and expectations will be clearly explained in this document. This assignment consists opportunity to experiment all the fundamental steps of software development **from programming to deployment**, so we are highly suggesting you to focus on the processes itself instead of struggling on **tiny details**.

## 2.2 Requirements

---

Assignment consists of 4 parts as follows;



1- User interface for **ONLY** adding ToDo's



2- Back-end service to store a persistent state of ToDo list and ability to **ONLY** adding ToDo



3- Writing deployment files of your *front-end* and *back-end*



4- Automating *build*, *test* and *deployment* of your application via CI/CD pipeline

### 2.2.1 User Interface

---

UI will be consist of a simple feature: **Adding a TODO item**. We expect you to build a very simple UI that covers **only** listed functionality and write automation tests for your UI. You can use any technology you want to use for UI and automation tests. Expected behavior of application is listed below.

```
1 | Given Empty ToDo list
2 | When I write "buy some milk" to <text box> and click to <add button>
3 | Then I should see "buy some milk" item in ToDo list
```

### 2.2.2 Back-end

---

### We expect you to;



1- Store the state of your ToDo list to a **database** through a **back-end service**.

### You can;



2- Not use any dinosaur languages (like java, c++, c, .net).



3- Write with a modern languages (rust, go, elixir, kotlin, clojure etc.).



4- Use protocols (http/protobuf/grpc) and databases (you can even use in-memory map or any data structure you prefer) for `ui-->service` communication.



5- **TDD** forever.

### 2.2.3 Deployment

---

### You should;



1- Dockerize both your **front-end** and **back-end** application to make them ready for deployment.



2- Deploy your application to a **Kubernetes cluster** or **Docker Swarm** or into a **VM** using a cloud provider.

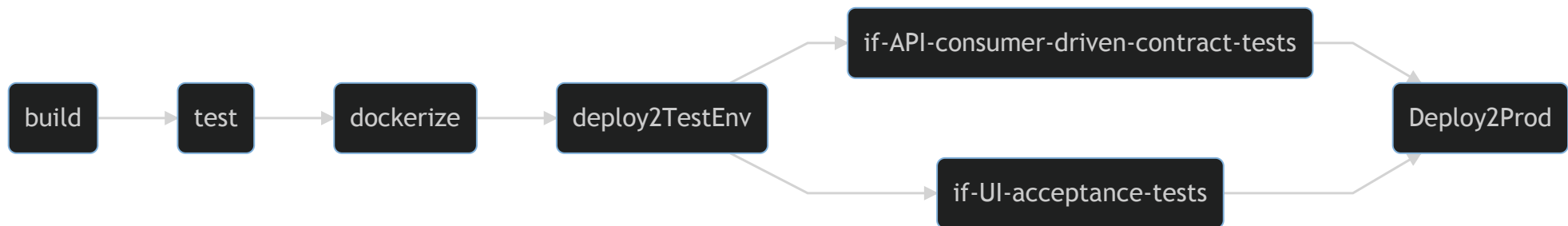
- Google Cloud Platform which also offers 300\\$/ free credit,
- Digital Ocean has very cheap instances available,
- AWS offers 1 core machine for free.



3- Write deployment configurations(i.e. Kubernetes config files, Docker Swarm files etc.) for each project.

## 2.2.4 Pipeline Automation

All the steps mentioned above can be automated via CI/CD pipelines. We expect you to automate all the steps;



You can use any CD/CI pipeline tool(gitlab-ci, circle, travis etc).

## Development Tips and Recommendations

**We want you to make it through this assignment. And we mean it! So, it would help you to listen to these recommendations as close as possible.**

At *Modanisa Technology*, we have a strong understanding of **better software** or **software quality**. To demonstrate what we mean by **quality** or **better**, we will show you how we deal with day to day feature requests which we call it **A-TDD cycle**, meaning **Acceptance Test Driven Design**;



We expect you to follow the following cycle as much as possible. We know most of candidates do not have experience on this kind of development flow and heavy testing. We promise you will gain a better development experience once you try to apply this cycle.



Testing steps in the following cycle is way MORE IMPORTANT than using fancy technologies. A candidate might FAIL if they did NOT write Acceptance,Component,Unit,Contract tests but they WILL PASS if they could not figure out Kubernetes deployment

```
1  foreach "acceptance criteria" in "story" do:
2    -> write AcceptanceTest
3    -> first acceptance test fails (Red)
4    -> it is time to start thinking about UI/Frontend code;
5        -> write a "widget/component/high level" test that ensures your first piece of UI elements are in place
6        and rendering correctly given correct "data" (in react/vue, it is shallow rendering and testing components)
7    -> component/widget test fails (Red, so far not a single line of actual code written)
8    -> write the dumb component that `only` satisfies your first component test's requirement
9    -> component test still fails since there is no business logic behind it, just presentation logic
10   -> write your first unit test asserting first business logic piece
11   -> unit test fails (Red)
12   -> write first business logic code that `only` satisfies first unit test
13   -> this might be the time you are actually calling APIs, so you should start thinking about designing your Backend S
14       -> if there is an API call to Backend Service;
15           -> write a contract test (consumer driven contract) with pact.io or apibluprints or whatever you feel comfortable
16           -> if you are using any of pact/apibluprints, you will have a "mock service" and a "failing test for your backend
17   -> unit test succeeds (Green)
18   -> refactor your code, tidy up, clean up, house keeping time (Refactor)
19   -> repeat
20       --> component test --> component code(presentation logic)
21       --> unit test --> business code
22           --> CDC test --> refactor
23       until satisfying your first acceptance criteria
24   -> this is what we call Red --> Green --> Refactor cycle at large
25 -> first acceptance test succeeds with depending on mock backend with pact/apibluprint mocks.
26
```

```
27 -----
28 -> time to move on to backend: Warning, we have only written single acceptance test and made it pass at Frontend with
29
30 -> write a CDC provider test satisfying written "contract test at Frontend code"
31 --> we have a failing CDC test now for our Backend Service as our first A-TDD step
32 -> write an integration test
33 -> integration test fails (Red)
34 -> create the resource routing by just writing resource to handler maps in your code
35 -> write your first unit test asserting first business logic piece
36 -> unit test fails (Red)
37 -> write first business logic code that `only` satisfies first unit test
38 -> unit test succeeds (Green)
39 -> refactor your code, tidy up, clean up, house keeping time (Refactor)
40 -> repeat
41 --> integration test --> routing code(presentation logic)
42 --> unit test --> business code
43 --> refactor
44 until satisfying your first CDC provider test
45 -> first CDC test succeeds (Green)
```

## Final Notes

---

- ▶ You should write README files for your projects which describes the project and necessary steps to develop, build, test and deploy corresponding project. (You can search for “awesome-readme” from Github and checkout for good examples.)
- ▶ We expect you to write your architectural decisions to README.
- ▶ We expect you to run TDD for every step of development.

## 3 - Pair programming & Computer Science Session

---

### 3.1- Pair Programming Session (45 mins)

---

During this session, we assess candidate's ability to extend a piece of code by the guideline recommended in **Development Tips and Recommendations** section.

### 3.2- Computer Science Basics (45 mins)

---



- ▶ During this session, we mostly start by an algorithm design/complexity analysis question. We mostly use algorithm questions from popular algorithm community questions as a reference. Like codewars, leetcode, hackerrank. You would fly in this session if you enjoy solving algorithm questions as hobby.
- ▶ Then we continue session by asking basic understanding around how a **computer program** works.
- ▶ Next is, we want to assess basic understanding of how a computer works by design. How a modern computer handle concurrency, multi-tasking, resource sharing etc.
- ▶ Next is, we start asking candidate around how a modern operating system works. We mostly ask around **linux, kernel, bootloaders, init process, process priority, basic console/terminal operations**, how modern packaging systems work(docker, lxc etc.)
- ▶ Next is **Computer Networking**. We want to make sure that candidate has basic understanding of how to communicate **reliably** on **unreliable** network mediums.
- ▶ Next is **Databases**. We want to make sure that candidate has enough understanding of how to reliably work on databases, scale them with trade-offs.
- ▶ Next in this session is **Languages and Compilers**. In this part of the session, candidate **demonstrate** what makes a programming language **good or bad** by **design**, by **type system**, by **ecosystem**.
- ▶ Last in this session is **Distributed Systems**. We mostly try to understand if candidate has an understanding of how a **internet scale** business would scale. We mostly discuss around **consensus protocols, reliable system design approaches, basic scalability**.