On-Boarding New Engineers: Best practices & lessons learned

Vincent Pradeilles (@v_pradeilles) – equensWorldline 🇫🇷

Why?

Why do I care about on-boarding? 🤔

I'm part of the Mobile Team in Lyon 🇫🇷

Our main project is the iOS apps for the French bank [redacted].

This project is built arround a **fairly complex** architecture.

At the same time, our client has an **ambitious roadmap**.

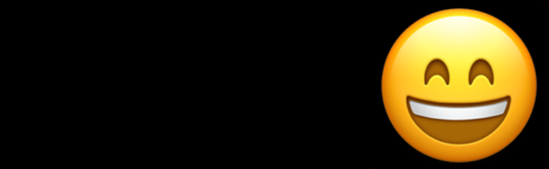
So we often need to bring **new engineers** to the team, in order to **meet the deadlines**.

This situation has been a **real challenge** 💪

To tackle it, we chose to invest in a **realiable on-boarding process** aimed towards our new recruits.

Why should you care about on-boarding? 🤔

I don't think that we are the **only team** that faces such challenges



At Worldline, our projects require that we operate increasingly **complex technical stacks**.

At the same time, our recruitment effort focuses on **young graduates**, that have a limited experience.

If we want them to smoothly and productively **find their place** within our organisation, we need an appropriate **on-boarding process**.

On-boarding?

On-boarding is the process to make someone reliably **independent** within an organisation.

This organisation can be a team, a project, a client portfolio, etc.

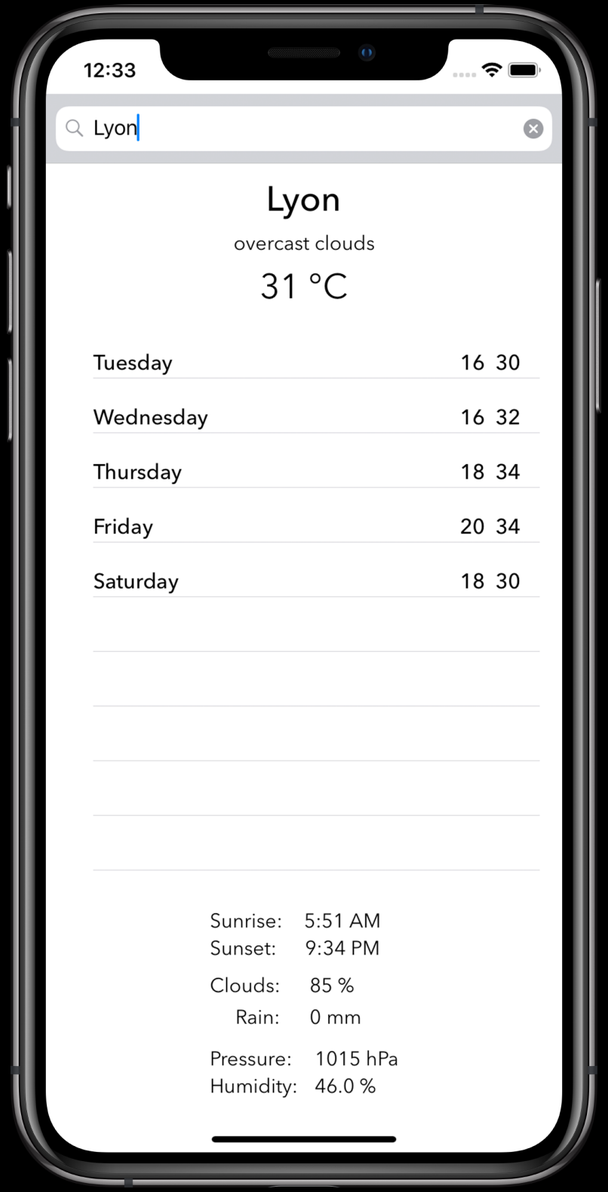
On-boarding is both a **technical** and **managerial** process.

In this talk, I'll focus mainly on the **technical** side.

But it doesn't mean that the other side should be ignored

What's inside our process?

Basically, we **begin** by asking our new recruit to code a mobile app that displays weather data.


12:33
Lyon
Lyon
overcast clouds
31 °C
Tuesday 16 30
Wednesday 16 32
Thursday 18 34
Friday 20 34
Saturday 18 30
Sunrise: 5:51 AM
Sunset: 9:34 PM
Clouds: 85 %
Rain: 0 mm
Pressure: 1015 hPa
Humidity: 46.0 %

This gives us the opportunity to **progressively** introduce our technical stack.

The app is meant to be written **incrementaly**.

Starting with a UI that displays hardcoded data...

…then moving on to decoding static JSON files…

...to performing API calls, etc.

The goal of this approach is to **avoid** a technical **overwhelming**

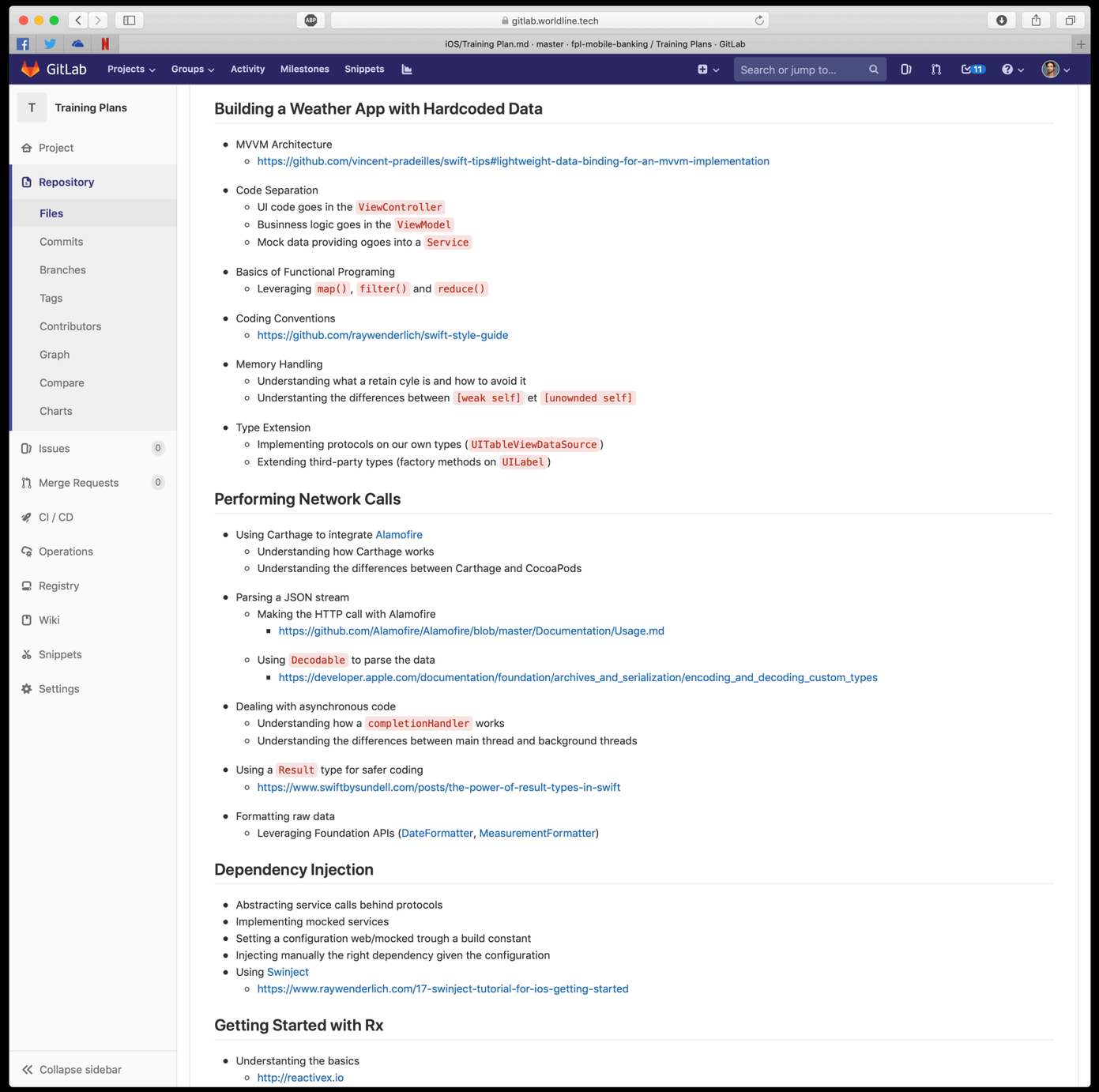
Instead focusing on the value **value** each **individual layer** of the stack brings to the whole.

How is it executed?

The whole process typically lasts between **2 to 4 weeks**, depending on how experienced our recruit is.

This training period is definitely an **investment**, but one that will pay **dividends**.

We have a training plan that is **available** on an internal **GitLab repository**.



Building a Weather App with Hardcoded Data

- MVVM Architecture
  - https://github.com/vincent-pradeilles/swift-tips#lightweight-data-binding-for-an-mvvm-implementation
- Code Separation
  - UI code goes in the `ViewController`
  - Businness logic goes in the `ViewModel`
  - Mock data providing ogoes into a `Service`
- Basics of Functional Programing
  - Leveraging `map()`, `filter()` and `reduce()`
- Coding Conventions
  - https://github.com/raywenderlich/swift-style-guide
- Memory Handling
  - Understanding what a retain cyle is and how to avoid it
  - Understanting the differences between `[weak self]` et `[unownded self]`
- Type Extension
  - Implementing protocols on our own types (`UITableViewDataSource`)
  - Extending third-party types (factory methods on `UILabel`)

Performing Network Calls

- Using Carthage to integrate Alamofire
  - Understanding how Carthage works
  - Understanding the differences between Carthage and CocoaPods
- Parsing a JSON stream
  - Making the HTTP call with Alamofire
    - https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md
  - Using `Decodable` to parse the data
    - https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types
- Dealing with asynchronous code
  - Understanding how a `completionHandler` works
  - Understanding the differences between main thread and background threads
- Using a `Result` type for safer coding
  - https://www.swiftbysundell.com/posts/the-power-of-result-types-in-swift
- Formatting raw data
  - Leveraging Foundation APIs (DateFormatter, MeasurementFormatter)

Dependency Injection

- Abstracting service calls behind protocols
- Implementing mocked services
- Setting a configuration web/mocked trough a build constant
- Injecting manually the right dependency given the configuration
- Using Swinject
  - https://www.raywenderlich.com/17-swinject-tutorial-for-ios-getting-started

Getting Started with Rx

- Understanting the basics
  - http://reactivex.io

On their first day, we hand out a **printed copy** to our recruits.

It helps **communicate** clearly what is **expected** of them.

We also assign a **mentor**, that will oversee the training period.

On a **daily** basis (idealy on the morning), we schedule **meetings** between mentor and mentee.

Those meetings are meant to **review progress** and **set realistic goals** for the day.

In order to be **efficient**, those meetings are **timeboxed**.

At the beginning, they last a **1/2 hour**, then their duration **decreases** as weeks go by.

They are a great time to
discuss difficulties

But also to explicitely **point out** what was **successfully achieved**

Once the trainning plan has been **completed**, it's time to be a part of a **real project**!

How do we deal with the arrival on a real projet?

To facilitate things, it's a good idea to **also** assign a mentor for the first months.

(This second mentor can be different than the first one)

His/Her role will be to help introducing the **project's structure**, **processes**, etc.

He/She'll also be able to give **feedback** to the **management** on how things are going.

The leap from junior to seasoned engineer

After a few **successful months** on the project, it's time to **grow**!

The engineer has acquired **skills** and **proficiency**, he/she's not really a junior anymore.

It's the perfect time to indtrocue a **stretch project**.

It's a task that requires **more autonomy** than before and has a relatively **high visibility**.

It can be a refactoring task, a new challenging feature, etc.

It's an **opportunity** for the engineer to move on to a **more senior** role.

And it's also **natural conclusion** to the on-boarding process.

That's it 

(Not really )

Recap

Recap

What makes a good technical on-boarding process?

Recap

What makes a good technical on-boarding process?

- A well-defined training plan

Recap

What makes a good technical on-boarding process?

- A well-defined training plan
- Realistic and incremental goals

Recap

What makes a good technical on-boarding process?

- A well-defined training plan
- Realistic and incremental goals
- Daily 1-o-1's to discuss progress and difficulties

Recap

What makes a good technical on-boarding process?

- A well-defined training plan
- Realistic and incremental goals
- Daily 1-o-1's to discuss progress and difficulties
- Steps to evolve from junior to seasoned engineer

What are the best practices?

⚠️ Word of advice ⚠️

Taken **individually**, the following best practices all sound kind of **obvious**.

However, getting **all of them** right turns out to be pretty tricky

1

On boarding is a **relationship**: it’s best to assign a single mentor for the whole process.

When several mentors are involved, it can get **confusing** for the mentee.

2

The training period is a **good time to make mistakes**, becomes they won't carry any real consequences.

A possible side-effect of having a written curriculum, is that people might feel that **perfect results** are expected.

It's important to make it clear to the mentee that **mistakes won't be held negatively** against them.

On the opposite, the ability to understand and learn from mistakes is a **valuable mindset**.

3

The same way, it's also important to **encourage autonomy**.

Some mentees might stay **blocked** on problems, because they are **not sure** of how they should be approached.

It's useful to point out that **attempts to solve** a problem will be **valued**, even if they don't end up in success.

4

Critical thinking is important.

A technical stack is the result of **choices** made in the **context** of a given project.

They are by no means an absolute truth!

It's important to point out what **alternative approaches** might look like.

A great trick is to provide links to **conference talks** that discuss those alternatives.

5

Knowing the **difference** between facts, best practices and opinions.

Facts are **demonstrable**, they can be scientifically proven.

You're **code doesn't build**,
that's a **fact**.

You can't **disagree** with it and still **be right**.

Best practices are relevant **choices** given a **particular context**.

Project X **chose** framework Y **because** of constraint Z.

Opinions are **personnal preferences**, based on experience and/or biais.

I **prefer** thiw way of doing things.

Communicating clearly in **which of those** categories a piece of information lies is extremely helpful.

This time it's really over 😊

Questions?

Contact

Twitter: @v_pradeilles