Implementasi *Unit Testing* Menggunakan Metode *Test-First Development*

Muhamamd Agung Rizkyana[1], Yunanto, Yoga, Caca Arif Herdian, Ahmad Ainul Yaqin R
Magister Sistem Informasi, STMIK LIKMI, Bandung
Jl. Ir.H. Juanda No. 96, Lebakgede, Kec. Coblong, Kota Bandung, 022 2502121
agung.rizky@gmail.com[1]

Diterima : 11 Desember 2020. Disetujui : 19 Maret 2021. Dipublikasikan : 30 Mei 2021.

Abstract - Today we are faced with a disruption era that brought some challenges for any business field, especially the software development field. Software development should be adaptive to those challenges. One of the most used and implemented for today in software development lifecycle was Agile. Agile Methodology can push out the software to be released. Meanwhile, the release velocity was not enough to keep the software accepted and used by users. The quality of the software was necessary because the alignment belongs to user needs showing those qualities. One of the methods in Agile that assuring the quality was Test-Driven Development (TDD). TDD has various techniques. This research used Test-First Development (TFD). TFD technique used for guidance unit testing of data service application with REST API architecture. The objective was to show TDD implementation that aligns with the concept and theory for the case study.

Keywords: agile, test-driven development, test-first development, unit tests, rest api

Abstrak* - Tuntutan untuk dapat terus bersaing dan berkualitas di era distriupsi saat ini menjadi tantangan khususnya pengembangan perangkat lunak. Untuk menjawab tantangan tersebut salah satu metodologi pengembangan yang berkembang dan banyak diimplementasikan adalah *Agile*. Metodologi *Agile* memungkinkan perangkat lunak untuk dapat segera diluncurkan. Namun, kecapatan saja tidak cukup untuk dapat menjaga sebuah perangkat lunak atau aplikasi dapat diterima dengan baik oleh pengguna. Kualitas dari perangkat lunak adalah hal penting, karena kesesuaian dengan kebutuhan pengguna menunjukan kualitas perangkat lunak tersebut sehingga timbul keterikatan antara pengguna dan aplikasi. Salah satu metode pada *Agile* yang menjamin hal tersebut adalah *Test-Driven Development* (TDD). TDD mempunyai beberapa teknik dalam praktisnya, pada penelitian ini menggunakan *Test-First Development* (TFD). Teknik TFD digunakan untuk menjadi panduan pengujian unit (Unit Testing) pada aplikasi penyedia data dengan arsitektur REST API. Tujuan dari penelitian ini adalah menunjukan langkah - langkah sistematis implementasi TDD sesuai konsep dan teori pada suatu studi kasus.

Kata kunci: agile, test-driven development, test-first development, unit tests, rest api

I. PENDAHULUAN

Siklus pengembangan perangkat lunak telah mengalami perkembangan seiring dengan kompleksitas dan distrupsi saat ini. Kebutuhan untuk dapat segera rilis agar dapat terus bersaing dan eksis. Namun, jika hanya sekedar rilis tanpa mempertimbangkan kualitas akan mengurangi *engagement* pengguna, manfaat dari perangkat lunak atau bahkan kualitas pengembangan perangkat lunak itu sendiri. Salah satu siklus pengembangan yang sedang populer saat ini adalah *Agile*. Metode *Agile* mempunyai prinsip pengembangan yang dinamakan *Agile Manifesto*. Prinsip tersebut antara lain, terbuka akan perubahan (*Welcoming Changes*), rutin mengirimkan kemajuan produk (*Delivering Frequent Working Product Iterations*), dan mengizinkan setiap anggota tim untuk mengelola secara mandiri (*allowing teams fo self-organize*) [1].

Agile tidak hanya sebagai proses atau metodologi, juga berisi sekelompok praktis, nilai dan prinsip. Keunggulan metode *Agile* dibandingkan dengan pengembangan perangkat lunak antara lain [2]:

1. Perubahan dinamis dari kebutuhan sistem dapat langsung diterapkan.
2. Pelanggan dilibatkan dalam setiap proses pengembangan.
3. Komunikasi antara pengembangan dan pelanggan sangat intensif.
4. Pelanggan tidak perlu menunggu hingga projek selesai.
5. Organisasi dapat menjamin bahwa produk dapat diterima oleh pelanggan setelah projek selesai.
6. Pengembang dapat dengan mudah melakukan ubahan sesuai permintaan pelanggan.
7. Hampir 100% pelanggan merasa puas dengan metode *Agile*.

Sementara itu, dalam praktis *Agile*, terdapat berbagai metode, salah satunya adalah *Test-Driven Development*.

Metode *Test-Driven Development* (TDD) membantu validasi dan verifikasi kualitas perangkat

lunak dengan kebutuhan pengguna. Terdapat beberapa teknik dalam TDD, salah satunya adalah *Test-First Development* (TFD) merupakan teknik membuat kode pengujian sebelum kode tersebut menjadi suatu fungsi yang utuh. Kode pengujian tersebut berdasarkan kepada kebutuhan sistem. Pengembang aplikasi agar dituntut untuk dapat membuat kode pengujian berdasarkan kebutuhan tersebut. Pengembang dapat mendekomposisi setiap kebutuhan sistem menjadi bagian-bagian kecil pengujian (*unit testing*). Bagian tersebut dapat berupa fungsi, metod atau *class*.

Penelitian ini merupakan perkembangan dari penelitian-penelitian sebelumnya. Adapun beberapa penelitian yang menjadi inspirasi dari penelitian ini terdapat pada tabel 1. Karya ilmiah tersebut menjadi acuan untuk uraian konsep dan teori *Test-Driven Development*, proses pada TDD, perbandingan TDD dengan BDD, dan implementasi TDD.

TABEL I. PENELITIAN TERKAIT

Judul	Tahun	Penulis
Assessing the Effectiveness of Test-Driven Development and Behavior-Driven Development in an Industry Setting	2019	Avishek Sharma Dookhun, Leckraj Nagowah
Comparative Study of Test-Driven Development, Behavior-Driven Development, and Acceptance Test-Driven Development	2019	Myint Myint Moe
Implementasi Test-Driven Development Pada Pengembangan Aplikasi Android untuk Mahasiswa Universitas Ahmad Dahlan	2018	Jamalludin, Herman Yuliansyah, Sri Winiati, Imam Riadi

Penelitian ini bertujuan untuk menunjukan langkah – langkah implementasi TDD menggunakan teknik *Test First Development* dan *Unit Testing* pada aplikasi *backend* penyedia data pengelolaan manajemen laboratorium kalibrasi. Modul dari aplikasi yang menjadi bagian penelitian ini adalah modul pengelolaan order dan autentikasi pengguna. Aplikasi menggunakan arsitektur *REST* sebagai teknik untuk mendistribusikan data.

Konsep makalah ini terdiri dari beberapa bagian antara lain Pendahuluan, Metode Penelitian, Hasil dan Pembahasan, Kesimpulan. Pendahuluan mengulas tentang siklus pengembangan perangkat lunak, uraian singkat TDD, penelitian terkait dan konsep makalah.Metode penelitian mengulas tentang literasi dari TDD, *Unit Testing*, *REST API Service*, *Refactoring*, dan Studi kasus. Hasil dan pembahasan mengulas tentang implementasi TDD pada studi kasus. Kesimpulan mengulas tentang intisari dari makalah ini.

II. METODE PENELITIAN

Penelitian ini menggunakan metode studi literatur dan eksperimen. Studi literatur berfokus pada topik *Test-Driven Development* (TDD), *Unit testing* dan *REST API service*. Eksperimen berfokus pada praktis – praktis TDD yaitu dengan teknik *Test-First Developmen*t (TFD) dan *refactoring*.

A. Test-Driven Development

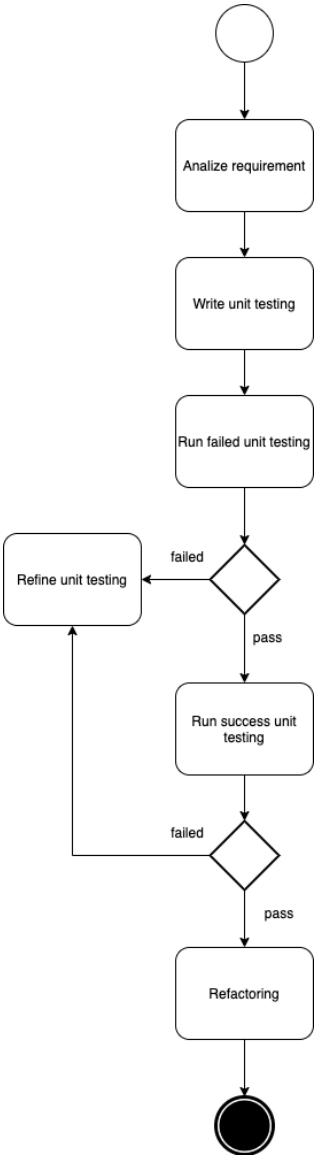
Test-Driven Development (TDD) menurut Ida Bagus, TDD adalah metode pengujian yang mewajibkan setiap pengembang untuk membuat rincian tujuan daripada pengujian sebelum membuat kode. Setelah pengujian dibuat, pengembang harus membuat kode yang memenuhi persyaratan uji hingga kode tersebut tidak mempunyai kesalahan atau kegagalan saat dibuat [13]. TDD memberikan kepastian terhadap peningkatan kualitas dan produktivitas, di tunjang dengan perkembangan kakas pengujian sebagai dukungan [7]. TDD merupakan salah satu teknik dalam pengembangan perangkat lunak yang menggabungkan rancangan program dan pengujiannya dalam satu iterasi kecil (*micro-iterations*). TDD juga memberikan kepercayaan diri terhadap setiap pengembang serta meningkatkan jangkauan uji (*test coverage*) pada sistem [3].

Secara umum, implementasi TDD menggunakan terminologi siklus *red-green-refactor*. Red menunjukan pengujian kegagalan terhadap kode. Green menunjukan kode perlu untu diubah dan diuji keberhasilannya. Refactor menunjukan perubahan atau perbaikan kode [3]. Efektifitas TDD dalam peningkatan kualitas pengembangan dibuktikan dari beberapa penelitian yang dilakukan oleh perusahaan teknologi informasi skala global, antara lain Microsoft, IBM dan Ericsson.

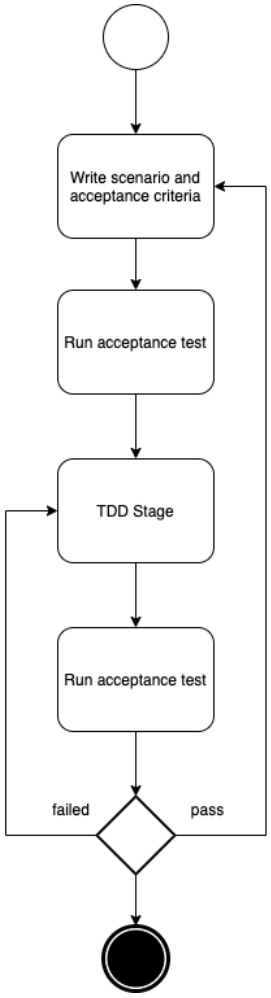
Berdasarkan penelitian yang dilakukan oleh Microsoft, penerapan TDD pada suatu projek telah meningkatkan kualitas kode hampir 2 (dua) kali lipat namun menggunakan waktu 15% lebih banyak untuk menulis kode pengujian. Sementara itu, IBM menyatakan bahwa penurunan 40% dari laporan kegagalan tanpa mempengaruhi produktivitas tim. Kemudian, Ericson dalam suatu studi kasus menunjukan peningkatan kualitas kode sebagai dampak dari peningkatan produktivitas tim sebesar 16% [3].

TDD memiliki sudut pandang dari pengembang (*developer*). Pengembang menggunakan *requirement* sistem kemudian diubah menjadi kode uji unit. Sementara itu, terdapat sudut

pandang lain dengan proses pengujian yang relatif sama. *Behavior Driven-Development* adalah alternatif penerapan TDD dengan penambahan sudut pandang berdasarkan skenario bisnis. BDD melibatkan *developer*, pengguna dan *tester* untuk validasi dan verifikasi kualitas perangkat lunak. Sebagai gambaran, pada gambar 1 ditunjukan pengujian dengan pendekatan TDD dan pada gambar 2 ditunjukan pengujian dengan pendekatan BDD [21].



Gambar 1. Alur kerja pada Pendekatan TDD [21]



Gambar 2. Alur kerja pada Pendekatan BDD [21]

Praktek TDD bersifat iteratif, di setiap iterasi pada umumnya melalui beberapa tahapan sebagai berikut [5]:

1. Membuat kode pengujian untuk fungsionalitas yang akan dimasukan ke dalam sistem.
2. Jalankan semua kode uji dan perhatikan setiap kegagalan
3. Buat kode perbaikan terhadap setiap pengujian yang gagal
4. Jalankan semua kode uji dan perhatikan setiap pengujian yang lolos.
5. Lakukan *refactor* kode untuk meningkatkan kualitas.

Setiap pengembang yang menerapkan TDD, harus memelihara rangkaian kode-kode pengujian dan hanya mengaplikasikan kode yang sudah lulus uji. Setiap kode uji adalah *executable spesication* yang artinya dapat dijalankan melalui perintah-perintah program. Sementara itu, kode uji tersebut akan membantu pengembang untuk memahami alasan mengapa fitur tersebut harus dibuat, menunjukan bagaimana cara fungsi tersebut di panggil atau apa hasil yang ingin dicapai [5].

Proses-proses yang terdapat pada TDD dapat dibagi menjadi 3 bagian utama antara lain: *granularity*, *uniformity* dan *sequencing*. *Granularity* dan *uniformity* berhubungan dengan proses iteratif. Sementara itu, *sequencing* berhubungan dengan urutan setiap proses di dalam siklus iterasi [6]. Dalam siklus iterasi, TDD berfokus pada sudut pandang "*inside-out*" yang bermakna bahwa setiap pengujian dibuat dari sudut pandang pengembang. Pengembang memaknai setiap kebutuhan sistem menjadi kasus uji (*test-case*). Selanjutnya, pengembang membuat kode uji terhadap kasus uji tersebut yang harus di pastikan bisa lolos. TDD memaksa pengembang berfokus kepada kebutuhan sistem [11].

Fokus tersebut membuat suatu perangkat lunak dianggap berkualitas. Ukuran kualitas tersebut adalah bekerja sesuai apa yang diinginkan pengguna dan proses bisnisnya. Pada TDD, diterapkan pengujian unit (*unit testing*) yang berfungsi untuk menjaga pengembangan perangkat lunak tetap sesuai dengan kebutuhan [12]. Selain itu, tujuan akhir dari TDD adalah membentuk kode yang lebih jelas, sederhana dan *bug-free* [11].

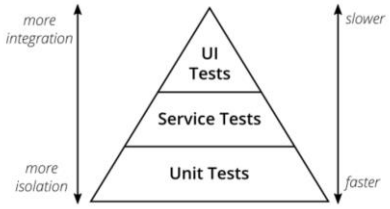
Kemudian terdapat tantangan dalam penerapan TDD. Sebagai contoh, pengembang mengalami kesulitan untuk membuat kode pengujian terhadap suatu fungsi tapi sebenarnya tidak ada yang perlu di uji. Adapun Langkah-langkah yang bisa lakukan apabila mengalami hal tersebut, antara lain :

1. Tentukan input dan output dari fitur spesifik.
2. Tentukan parameter untuk fungsi pengujian.
3. Fokus kepada 1 aspek pengujian.
4. Implementasikan fungsi pengujian dengan sudut pandang *behavior* bukan bagaimana menerapkan kode tersebut.
5. Terapkan kode yang sudah lolos uji satu per satu.

TDD memberikan dampak yang signifikan pada produktivitas pekerjaan pengembangan perangkat lunak. Sebuah penelitian yang dilakukan oleh Vojislav Misic pada tahun 2018 tentang efektivitas TDD terhadap kualitas dan produktivitas, menunjukan bahwa TDD menambahkan sebuah peningkatan kecil pada kualitas tetapi tidak begitu menyakinkan pada produktivitas. Karena, terdapat beberapa perbedaan hasil ditinjau dari konteks dimana TDD tersebut diterapkan. Konteks tersebut terdiri dari perbandingan tempat diterapkan dan siklus pengembangannya. Perbandingan pertama adalah Akademik dengan Industri. Kemudian, perbandingan kedua adalah siklus *waterfall* dengan ITL (*Iterative Test-Last*) [4].

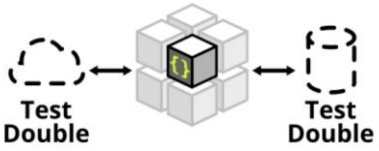
B. Unit Testing

Dalam prateknya, TDD melakukan pengujian yang terisolasi dan berfokus pada kebutuhan serta proses bisnis. Aspek tersebut adalah unit pengujian (*unit testing*). Pengujian unit merupakan bagian dari level pengujian lainnya. Menurut Mike Cohn dalam buku *Succeeding with Agile*, dikemukan tentang *Testing Pyramid*.



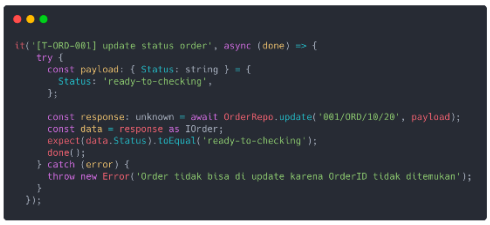
Gambar 3. *Testing Pyramid* [17]

Pada gambar 3, ditunjukan tentang layer pengujian. Secara urutan, dari atas ke bawah menunjukan semakin terisolasi kepada bagian terkecil (unit). Dari bawah ke atas secara waktu pengujian semakin lama. Testing pyramid terdiri dari *UI Tests*, *Service Tests* dan *Unit Tests*. UI Tests dilakukan proses pengujian langsung oleh pengguna. *Service Tests* dilakukan proses pengujian untuk memastikan integrasi antara fungsional sistem. Unit Tests dilakukan proses pengujian terhadap fungsional sistem [17].



Gambar 4. Unit Tests

Definisi *unit tests* apabila dihubungkan dengan bahasa dan paradigma pemrograman, setidaknya akan mempunyai perbedaan definisi. Pada pemrograman berbasis fungsi (*functional programming*), unit adalah suatu fungsi yang dipanggil dan disertai dengan parameter untuk selanjutnya diuji kesesuaian nilai yang dikembalikan (*return value*). Sedangkan pada pemrograman berbasis objek, unit dapat dimaknai sebagai *method* atau keseluruhan *class* [17]. Menurut Vladimir, definisi dari *unit test* adalah suatu pengujian yang di otomatisasi untuk melakukan verifikasi terhadap bagian terkecil (unit) dari kode, dilakukan secara cepat dan terisolasi [18].



Gambar 5. Struktur *Unit Test*

Sebuah *unit test* mempunyai struktur yang membentuknya. Mungkin terdapat beberapa perbedaan dari cara penggunaannya tergantung bahasa pemrograman. Pada penelitian ini menggunakan bahasa pemrograman *Typescript*. Gambar 5 menunjukan struktur dari *unit test update* status *order*. Struktur unit test dapat dibentuk melalui pola AAA yaitu *Arrange*, *Act* dan *Assert*.

Pada *arrange*, dipersiapkan semua data yang dibutuhkan pada fungsi sebagai parameter. Pada *act*, dilakukan pemanggilan fungsi kemudian di lemparkan (*pass*) variabel dari data ke dalam fungsi tersebut. Pada *assert*, dilakukan verifikasi dari output fungsi tersebut. Pola *Arrange*, *Act* dan *Assert* sebaiknya dihindari untuk terjadinya duplikasi di setiap komponennya. Hal ini untuk tetap menjaga sebuah *unit test* tetap sederhana, cepat dan mudah untuk di pahami [18].

Unit testing bergantung kepada bahasa pemrogram yang digunakan. Pada penelitian ini menggunakan *Typescript*. Kemudian, *unit testing* agar lebih efektif menggunakan *library* pendukung, pada penelitian ini menggunakan JEST. Untuk dapat menggunakan JEST pastikan terlebih dahulu lingkungan pengembangan sudah terinstall NodeJS. Proses pengujian unit akan menggunakan perintah dari *command-line*. Pada konfigurasi awal, JEST akan mengenali suatu berkas yang merupakan unit pengujian dengan pola [nama-unit].test.[ext]. Jika pada bahasa pemrograman *Typescript*, maka pola nama berkas yang digunakan adalah order.test.ts.

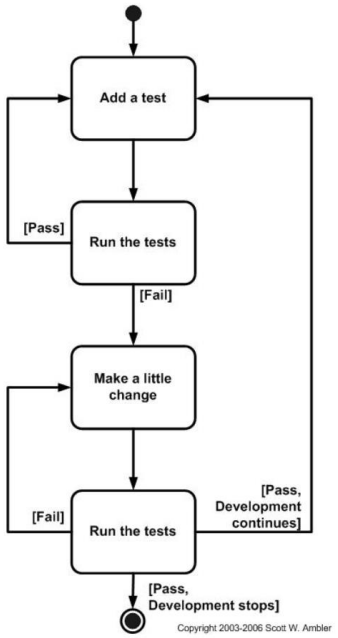
C. *REST API Service*

REST API Service adalah salah satu teknik dan arsitektur dalam *web service*. Webservice merupakan sistem perangkat lunak yang didesain untuk dapat dioperasikan dari mesin ke mesin melalui jaringan. Secara umum, arsitektur web service terdiri dari penyedia, konsumen dan pendaftar. Melalui *web service* dimungkinkan untuk menghubungkan secara dinamis baik perangkat keras, maupun perangkat lunak dalam suatu jaringan [16].

REST API sendiri merupakan arsitektur distribusi data dan proses antara aplikasi dan *service* melalui jaringan internet dengan protokol HTTP. REST API diakses dan dikenali melalui alamat URL yang unik serta metod yang menyertainya seperti GET, POST, PUT, atau DELETE [16]. REST adalah singkatan dari *Representational State Transfer*. REST API mempunyai kebebasan untuk format data yang diterapkan padanya seperti JSON, XML, atau TEXT [15].

D. *Test-First Development*

Penelitian ini menggunakan salah satu teknik dari *Test-Driven Development* (TDD) yaitu *Test-First Development* (TFD). Pada gambar 6, ditampilkan alur pengembangan pengujian aplikasi. Pada tahapan pertama dibuat terlebih dahulu skenario pengujian dan kasus uji (*test case*).



Gambar 6. Test-First Development [19]

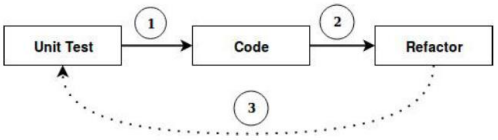
Test case dijalankan untk menguji kegagalannya terlebih dahulu. Apabila sudah dapat diketahui kegagalannya maka dianggap benar (pass). Selanjutnya, dilakukan refactoring pada kode agar proses dioptimalisasi dan kode lebih mudah dibaca. Kemudian, kode yang sudah di *refactor*, dilakukan pengujian kembali dengan harapan uji berhasil. Apabila terjadi kegagalan maka dilakukan tinjauan ulang terhadap kode untuk selanjutnya di perbaiki dan dilanjutkan kembali hingga pengujian dianggap berhasil. Setelah pengujian berhasil, maka kode dibuat dengan lebih tepat secara fungsional dan pola design (*Design Pattern*) [19].

E. *Refactoring*

Teknik TFD pada umumnya akan menghasilkan kode yang tidak rapih. Oleh karena itu, proses *refactoring* menjadi perlu untuk dilakukan. *Refactoring* adalah proses untuk membuat kode lebih mudah di kelola (*Maintain*),

sesuai dengan pola design (*Design Pattern*) dan dapat dipahami oleh pengembang lain [12].

Menurut Martin Fowler, definisi dari *refactoring* adalah segala hal tentang penerapan ubahan kecil pada tiap tiap bagian kode untuk kemudian menjadi ubahan yang cukup besar dengan menggabungkan setiap ubahan kecil tersebut pada suatu urutan yang sistematis [17].



Gambar 7. Proses refactoring [12]

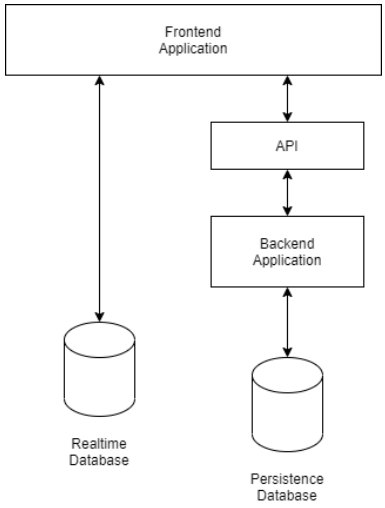
Pada gambar 7, ditampilkan alur proses daripada *refactoring*. Pertama, pengembang membuat terlebih dahulu unit uji (unit test). Unit uji dapat berupa alur fungsional yang sederhana dan sebaiknya mempunyai satu *assertion*. *Assertion* adalah fungsi dari modul pengujian untuk membantu kepastian hasil dengan ekspektasi pada skenario pengujian. Selanjutnya, pengembang membuat kode untuk unit uji tersebut. Apabila sudah sesuai ekspektasi atas hasil uji tersebut, dilanjutkan dengan *refactoring* kode.

F. Sistem Informasi Pelayanan Laboratorium Kalibrasi

Sistem Infomrasi Pelayanan Laboratorim Kalibrasi dirancang untuk pengelolaan permintaan kalibrasi alat berdasarkan lingkup yang sudah ditentukan. Sistem informasi ini bernama Kallon. Kallon mempunyai fitur yang terdiri dari:

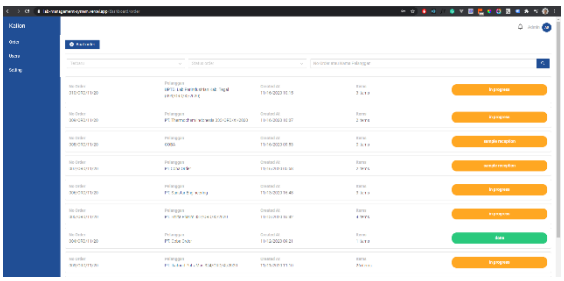
1. Pengelolaan order
2. Pengelolaan penjualan
3. Pengelolaan penagihan

Saat ini sistem informasi tersebut masih dalam pengembangan dan menerapkan metodologi *agile* dengan teknik *prototyping* dan TDD. Fitur yang sudah dapat digunakan adalah pengelolaan *order*. Arsitektur yang digunakan adalah *backend-for-frontend* (BFF), dengan dukungan *REST* API.

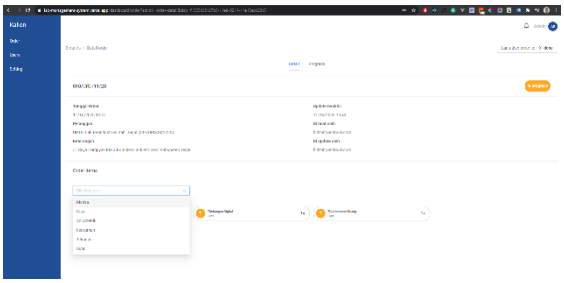


Gambar 8. Arsitektur Kallon

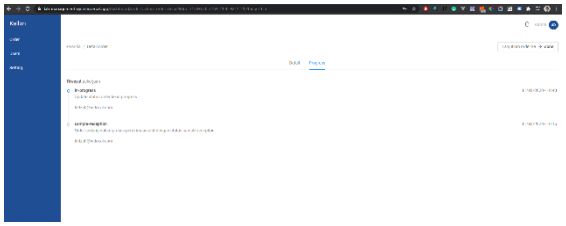
Pada gambar 8 ditunjukan arsitektur Kallon. Kallon terdiri dari aplikasi *frontend*, API, aplikasi *backend* dan dua jenis database yaitu *realtime* dan *persistence*. *Realtime database* menggunakan *Firebase* dan *persistence* menggunakan MongoDB. Aplikasi *frontend* dibangun menggunakan *Typescript* dengan *framework* ReactJS. Untuk aplikasi *backend* dan API dibangun menggunakan *Typescript* dengan *framework* ExpressJS Gambar 9 - 12 menunjukan tampilan dari aplikasi Kallon untuk pengelolaan order kalibrasi.



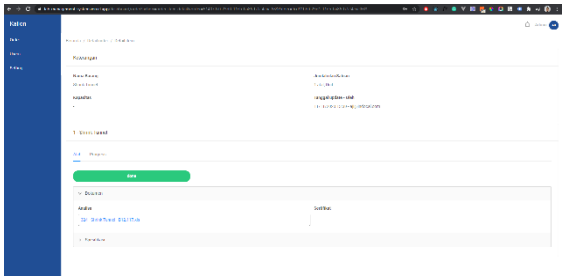
Gambar 9. Daftar Order Kalibrasi



Gambar 10. Detail Order Kalibrasi



Gambar 11. Kemajuan Pelayanan Order Kalibrasi



Gambar 12. Detail Alat pada Suatu Order Kalibrasi

III. HASIL DAN PEMBAHASAN

Hasil dari penelitian ini adalah menyajikan Langkah-langkah sistematis sesuai dengan konsep TDD serta implementasi *unit testing*. Implementasi tersebut terdapat pada layanan *REST* API melalui protokol HTTPS untuk mengelola data berformat JSON. *REST* API digunakan pada layanan pada produk Kallon. Kallon adalah perangkat lunak berbasis web yang mengelola pelayanan kalibrasi. Setiap iterasi TDD, menerapakan pengujian unit.

A. Skenario Pengujian Unit

Pengujian unit pada penelitian ini berfokus pada *endpoint order* dan autentikasi. *Endpoint order* mempunyai *use-case* buat *order*, *update order*, pencarian *order*, dan daftar *order*. *Endpoint* authentikasi mempunyai *use-case* buat akses token dan validasi token. Skenario pengujian terdiri dari *test case*, *expected failed* dan *expected success*. Pada Tabel 2 terdapat skenario pengujian unit pada *endpoint order* sedangkan pada tabel 3 terdapat skenario pengujian unit untuk *endpoint* autentikasi.

TABEL II. SKENARIO PENGUJIAN UNIT ORDER

Test ID	*Test Case*	*Expected Failed*	*Expected Success*
T-ORD-001	*Update status order*	Tidak bisa *update order* karena *OrderID* tidak ditemukan	Berhasil *update order* dengan *OrderID* 001/ORD/10/20
T-ORD-002	*Create order*	Tidak bisa *create order* karena *OrderID* sudah ada	Berhasil *create order*
T-ORD-003	*List order with parameter*	Tidak bisa mendapatkan *list order* karena kesalahan *parameter limit* dan *page*	Berhasil mendapatkan *list order*
T-ORD-004	*Generate Order ID*	Nomor *order* tidak sesuai dengan format yang ditentukan	Berhasil membuat nomor *order* sesuai dengan format yang ditentukan
T-ORD-005	*Get single order by Order ID*	Tidak bisa mendapatkan *order*	Berhasil mendapatkan *order*

TABEL III. SKENARIO PENGUJIAN UNIT AUTENTIKASI

Test ID	*Test Case*	*Expected Failed*	*Expected Success*
T-AUTH-001	*Create basic auth token*	Gagal membuat token	Berhasil membuat token
T-AUTH-002	*Validate basic auth token*	Gagal validasi token	Berhasil validasi token

B. Implementasi Pengujian Unit

Service REST API ini dikembangkan menggunakan bahasa pemrograman *Typescript*. Untuk pengujian unit menggunakan kakas dari JEST. Konfigurasi Jest untuk dapat melakukan pengujian unit pada projek *service* REST API ini terdapat pada *package.json*. Gambar 13 menunjukan konfigurasi project *REST* API yang sudah tersedia JEST.

Gambar 13. Konfigurasi JEST

Struktur unit testing menggunakan pendekatan *Arrange*, *Act*, *Assert*. Pada *arrange* data dipersiapkan, *act* dilakukan pemanggilan fungsi kemudian pada *assert* di validasi ekspektasi hasilnya. Gambar 14 menunjukan *code* dasar untuk membuat *unit testing*. Gambar 15 menunjukan perintah untuk menjalankan *unit testing*.

```
describe('Unit Testing General Topic', () => {
  it('function unit testing', async (done) => {
    /**
     * Arrange
     * Act
     * Assert
     */
  })
});
```

Gambar 14. Code Dasar Pengujian Unit

```
mail-kalibrasionline-id git:(master) ✗ yarn test ./src/__tests__/order.test.ts
```

Gambar 15. Perintah untuk Melakukan Pengujian Unit

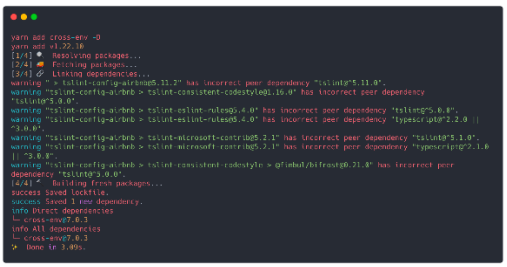
C. *Hasil Pengujian Unit*

Pengujian unit menggunakan *library* dari *node package manager* (NPM) dengan nama *JEST*. Sebelum dilakukan pengujian, JEST perlu untuk dikonfigurasi terhadap projek ini. Tahapan konfigurasi dilakukan dengan cara sebagai berikut:

1. Buka terminal melalui text editor Visual Studio Code, dengan perintah seperti pada gambar 16.

Gambar 16. Perintah untuk *Install Library* JEST

2. Selanjutnya tambahkan *library* pendukung lainnya untuk pengelolaan *environment* variabel. Library tersebut adalah *cross-env*.



Gambar 17. Perintah untuk *install library cross-env*

3. Selanjutnya, modifikasi file *package.json* pada bagian *script*. Modifikasi dilakukan untuk penambahan perintah-perintah pengujian unit.

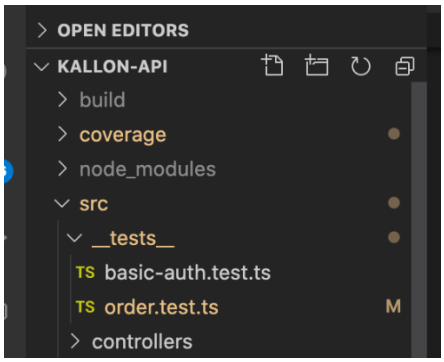
```
"scripts": {
    ...
    "test": "cross-env NODE_ENV=test jest --detectOpenHandles",
    "test:coverage": "cross-env NODE_ENV=test jest --coverage",
    "test:watch": "cross-env NODE_ENV=test jest --watch",
    ...
  },
```

Gambar 18. Konfigurasi Perintah Pengujian pada *package.json*

Perintah *test* digunakan untuk pengujian unit pada setiap *file* pengujian. Perintah *test:coverage* digunakan untuk pengujian unit dengan tambahan informasi lingkup pengujian. Perintah *test:watch* digunakan untuk pengujian unit dengan pemantauan setiap ubahan pada *file* pengujian. Pada JEST, *file* pengujian di identifikasi dengan nama *file* yang telah ditentukan. Nama *file* tersebut mempunyai format sebagai berikut

[nama-pengujian].test.ts

Sebagai contoh, pada pengujian unit pengelolaan order, maka penamaan *file* pengujian menjadi *order.test.ts*. Selain itu, penamaan *folder* untuk penyimpanan *file* pengujian ditentukan menjadi *__tests__*. Pada gambar 19 ditampilkan struktur *folder* dan *file* pengujian.



Gambar 19. Struktur *Folder* dan *File* Pengujian Unit

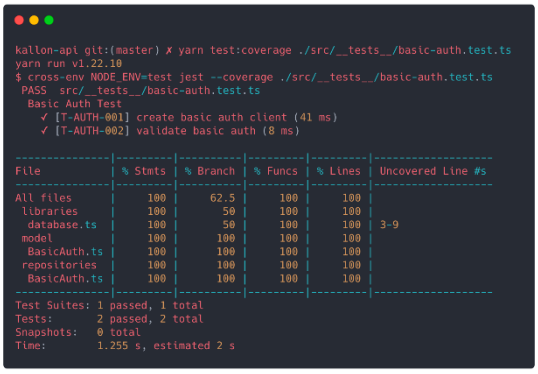
Hasil pengujian unit menggunakan JEST dapat dilihat secara lengkap melalui perintah *test:coverage*. Perintah tersebut selain menjalankan proses pengujian unit di setiap *file* juga menghasilkan matrik pengujian dari masing – masing *file* uji. Matrik yang digunakan terdiri dari *File*, *Statements*, *Branch*, *Functions*, *Lines* dan *Uncovered Lines*. Matrik selain *File* disajikan dalam satuan persen. Matrik ini digunakan untuk mengetahui berapa banyak kode yang diuji dan mengukur apakah kode pengujian sudah cukup baik. Penjelasan tentang matriks pengujian unit terdapat pada Tabel 4.

TABEL IV. MATRIKS PENGUJIAN UNIT

Matriks	Keterangan
File	menunjukan *file* apa saja yang dilibatkan dalam 1 kali eksekusi pengujian unit
Statements	menunjukan persentase kode yang dieksekusi dalam pengujian
Branch	menunjukan persentase struktur kontrol (pengkondisian) yang dieksekusi dalam pengujian
Functions	menunjukan persentase fungsi – fungsi yang di eksekusi pada pengujian
Lines	menunjukan persentase jumlah baris yang dieksekusi dalam pengujian
Uncovered Lines	menunjukan posisi baris yang belum ter-eksekusi oleh pengujian

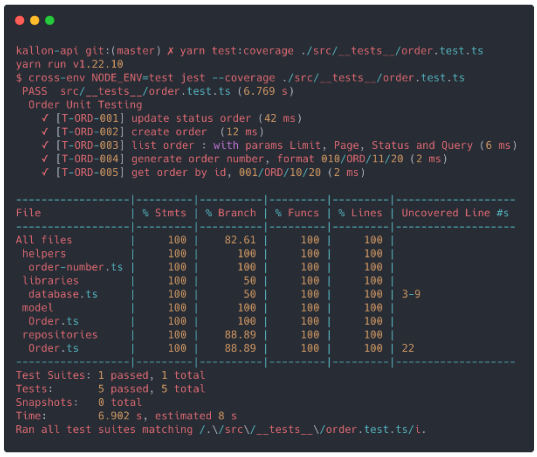
Pengujian unit dilakukan sesuai skenario pengujian yang terdapat pada Tabel 1. Pengujian unit dilakukan pada fitur authentikasi dan order. Unit uji untuk authentikasi terdapat pada *file* basic-auth.test.ts. Unit uji untuk order terdapat pada *file* order.test.ts. Tahapan pengujian unit mengadopsi tahapan *Test-First Development*. Pengujian unit menggunakan JEST dapat menghasilkan laporan *code coverage*. Laporan ini berisi tentang persentase kode yang terlibat dalam eksekusi pengujian unit. Persentase tersebut menunjukan kualitas kode dan *refactoring* yang dilakukan oleh pengembangan (*developer*).

Tahapan pengujian menggunakan JEST hingga dihasilkan laporan *code coverage* dimulai dari eksekusi pengujian unit untuk authentikasi, selanjutnya pengujian unit order dan terakhir pengujian unit untuk keseluruhan fitur. Berdasarkan situs resmi dari JEST (jest.io), persentase standar dari *code coverage* adalah 80% (*threshold*). Pada penelitian ini akan ditemukan *code coverage* 50% pada file database.ts. File tersebut menghasilkan persentase *code coverage* 50% karena tidak terdapat pengkondisian dalam kode yang ditulis. Hasil tersebut tidak mempengaruhi fungsionalitas dari kode tersebut.



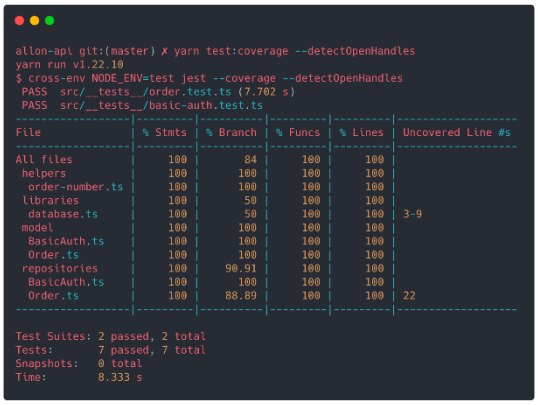
Gambar 20. Pengujian Unit pada Unit Uji Authentikasi

Pada gambar 20 ditunjukan hasil dari matriks pengujian untuk unit uji authentikasi. Hasil tersebut menunjukan nilai 100% pada *Statement, Functions* dan *Lines*. Sementara itu, pada *Branch* menunjukan nilai 62,5%. Hasil ini bermakna bahwa eksekusi pengujian unit sudah secara menyeluruh terjadi pada kode, fungsi dan baris namun pengujian unit belum menyentuh (*cover*) pengkondisian secara menyeluruh.



Gambar 21. Pengujian Unit pada Unit Uji Order

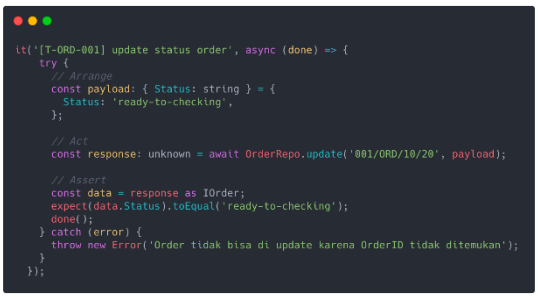
Pada gambar 21 ditunjukan hasil pengujian untuk unit uji order. Hasil tersebut menunjukan nilai 100% pada matriks yang sama seperti pengujian unit authentikasi. Terdapat perbedaan pada matriks *Branch* yaitu 82,61%. Hal ini disebabkan oleh jumlah file uji yang lebih banyak dan terdapat nilai persentase *Branch* sebesar 88,89% pada file Order.ts.



Gambar 22. Perintah Pengujian Unit untuk Unit Uji Order dan Authentikasi

Pada gambar 22 ditunjukan hasil pengujian unit pada seluruh file unit uji. Hasil tersebut menunjukan nilai 100 % pada *Statements*, *Functions* dan *Lines*. Sementara itu, pada *Branch* menghasilkan 84%. Nilai tersebut bermakna pengujian unit yang dijalankan sudah mengeksekusi unit uji secara menyeluruh dan dipastikan bahwa kode sudah mempunyai kualitas yang baik (lebih dari nilai *threshold*).

Peningkatan nilai tersebut dilalui dengan tahapan iterasi dan *refactoring* sesuai tahapan yang ada pada *TFD*. Iterasi pada pengujian unit meliputi penulisan kode unit uji, eksekusi pengujian dengan harapan berhasil (*expected success*), pengujian dengan harapan gagal (*expected failed*), dan *refactoring*.



Gambar 23. Contoh Kode Uji Unit

Pada gambar 23 ditunjukan tahapan pertama dari iterasi pengujian unit. Kode uji unit tersebut digunakan untuk pengujian *requirement update status order*. Fungsi untuk update status *order* terdapat pada *OrderRepo.update()*. Uji unit menggunakan fungsi *assert toEqual()*. Diharapkan hasil dari pengujian ini adalah setelah *order* dengan nomor 001/ORD/10/20 di*update* dengan status *ready-to-checking* maka hasil dari fungsi *update()* memiliki status yang sama yaitu *ready-to-checking*.

Gambar 24. Hasil Pengujian Unit T-ORD-001 *Expected Failed*

Pada gambar 24 ditunjukan hasil pengujian unit pada unit uji *update* status *order*. Hasil menunjukan kegagalan dikarenakan nomor *order* tidak ditemukan. Kemudian pada tahap ini, *developer* melakukan perbaikan kode uji unit. Setelah kode uji unit diperbaiki, maka dilakukan kembali ekseskusi pengujian dengan harapan berhasil.

Gambar 25. Hasil Pengujian Unit T-ORD-001 *Expected Success*

Pada gambar 25 ditunjukan hasil pengujian unit pada unit uji *update* status *order*. Hasil menunjukan bahwa pengujian sudah berhasil. Pada tahapan ini *developer* akan melakukan *refactoring*.

IV. KESIMPULAN DAN SARAN

Unit testing merupakan bagian dari *Test-Driven Development* yang sangat penting. Fungsional aplikasi akan terukur dan terjaga kualitasnya mulai dari bagian terkecil (unit). Sementara itu untuk menjaga validitas fungsi terhadap *requirement* dari penggunanya.

Test-Driven Development dapat diterapkan pada siklus pengembangan sistem informasi yang modern. Salah satu siklus tersebut adalah *Agile*. Pendekatan pengujian yang dilakukan adalah *Test-First Development*. Pendekatan ini melibatkan *test case* dan *test scenario* dari awal, sehingga dapat segera mengantisipasi terjadinya kesalahan baik secara lojik, bisnis maupun kode.

Metode ini memang masih mempunyai tantangan dari penerapannya. Seperti kemampuan

pengembang untuk menentukan aspek pengujian hingga waktu pengembangan yang menjadi lebih lama. Namun, TDD dapat dijadikan seperti budaya dalam suatu projek perangkat lunak. TDD untuk waktu singkat memang memberikan dampak yang belum signifikan dalam merubah kualitas suatu perangkat lunak. Dalam jangka panjang, merupakan suatu kebiasaan yang baik, karena akan melatih pengembang dalam menyesuaikan kebutuhan sistem dengan implementasi kode, kualitas kode yang diterapkan, kualitas fungsional sistem serta dapat berdampak pada efektifitas dan efisiensi waktu projek.

REFERENSI

[1] Bott, M., & Mesmer, B. (2019). An Analysis of Theories Supporting Agile Scrum and the Use of Scrum in Systems Engineering. Engineering Management Journal, 1–10. doi:10.1080/10429247.2019.1659701

[2] Lalband, Neelu & Kavitha, D. (2020). Software Development Technique for the Betterment of End User Satisfaction using Agile Methodology. TEM Journal. 9. 992-1002. 10.18421/TEM93-22

[3] Khanam, Zeba & Ahsan, M.N.. (2017). Evaluating the effectiveness of test driven development: Advantages and pitfalls. International Journal of Applied Engineering Research. 12. 7705-7716

[4] Rafique, Yahya & Misic, Vojislav. (2013). The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. Software Engineering, IEEE Transactions on. 39. 835-856. 10.1109/TSE.2012.28

[5] Besson, Felipe & Moura, Paulo & Kon, Fabio & Milojicic, Dejan. (2014). Bringing Test-Driven Development to Web Service Choreographies. Journal of Systems and Software. 99. 10.1016/j.jss.2014.09.034

[6] Fucci, Davide & Erdogmus, Hakan & Turhan, Burak & Oivo, Markku & Juristo, Natalia. (2016). A Dissection of Test-Driven Development: Does It Really Matter to Test-First or to Test-Last?. IEEE Transactions on Software Engineering. 43. 1-1. 10.1109/TSE.2016.2616877

[7] Karac, Itir & Turhan, Burak. (2018). What Do We (Really) Know about Test-Driven Development?. IEEE Software. 35. 81-85. 10.1109/MS.2018.2801554

[8] Lalband, Neelu & Kavitha, D. (2020). Software Development Technique for the Betterment of End User Satisfaction using Agile Methodology. TEM Journal. 9. 992-1002. 10.18421/TEM93-22

[9] Bhavsar, Krunal & Gopalan, Samir & Shah, Vrutik. (2020). Scrum: An Agile Process Reengineering in Software Engineering. International Journal of Innovative Technology and Exploring Engineering. 9. 10.35940/ijitee.C8545.019320.

[10] Jamaluddin, Herman Yuliansyah, Sri Winiati, Imam Riady. (2018). Implementasi Test Driven Development Pada Pengembangan Aplikasi Android Untuk Mahasiswa Universitas Ahmad Dahlan. JITEKI. Vol. 4, No. 1

[11] Myint Myint Moe. (2019). Comparative Study of Test-Driven Development (TDD), Behavior-Driven Development (BDD) and Acceptance Test–Driven Development (ATDD). IJTSRD. Vol. 3, Issue 4

[12] Thohari, Afandi & Amalia, Andika. (2018). Implementasi Test Driven Development Dalam Pengembangan Aplikasi Berbasis Web. SITECH : Jurnal Sistem Informasi dan Teknologi. 1. 1-10. 10.24176/sitech.v1i1.2255.

[13] Manuaba, Ida Bagus. (2019). Combination of Test-Driven Development and Behavior-Driven Development for Improving Backend Testing Performance. Procedia Computer Science. 157. 79-86. 10.1016/j.procs.2019.08.144.

[14] Suzanti, Ika & Fitriani, Nurhayati & Jauhari, Ahmad & Khozaimi, Ach. (2020). REST API Implementation on Android Based Monitoring Application. Journal of Physics: Conference Series. 1569. 022088. 10.1088/1742-6596/1569/2/022088.

[15] Rulloh, A., Mahmudah, D., & Kabetta, H. (2017). Implementasi REST API pada Aplikasi Panduan Kepaskibraan Berbasis Android.

[16] Perkasa, Muhammad & Setiawan, Eko. (2018). Pembangunan Web Service Data Masyarakat Menggunakan REST API dengan Access Token. Jurnal ULTIMA Computing. 10. 19-26. 10.31937/sk.v10i1.838.

[17] Martin Fowler, Kent Beck. Refactoring Improving the Design of Existing Code Second Edition. Addison-Wesley. 2019: 57 – 58.

[18] Vladimir Khorikov. Unit Testing Principles, Practices, and Patterns. Manning. 2020: 64 – 65

[19] Agile Data, “Introdoction to Test Driven Development”, 2020

[20] Alsaqqa, Samar & Sawalha, Samer & Abdel-Nabi, Hiba. (2020). Agile Software Development: Methodologies and Trends. International Journal of Interactive Mobile Technologies (iJIM). 14. 246. 10.3991/ijim.v14i11.13269.

[21] Dookhun, Avishek & Nagowah, Leckraj. (2019). Assessing The Effectiveness Of Test-Driven Development and Behavior-Driven Development in an Industry Setting. 365-370. 10.1109/ICCIKE47802.2019.9004328.