PENERAPAN *CLEAN CODE* PADA PENGEMBANGAN APLIKASI MANAJEMEN DAN MONITORING LAHAN PERTANIAN PADI

SKRIPSI

Sebagai salah satu syarat untuk menyelesaikan
Program Diploma IV Rekayasa Perangkat lunak
Politeknik Negeri Indramayu

Oleh:
ABU MUSHONNIP
NIM 1805001

PROGRAM STUDI REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
POLITEKNIK NEGERI INDRAMAYU
AGUSTUS 2022

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh:

Nama : ABU MUSHONNIP

NIM : 1805001

Program Studi : Diploma IV Rekayasa Perangkat Lunak

Judul : Penerapan *Clean Code* Pada Pengembangan Aplikasi Manajemen dan Monitoring Lahan Pertanian Padi

Pembimbing : I. A. Sumarudin, S.Pd., MT., M.Sc
NIP 198610102019031014 .....

II. Alifia Puspaningrum, S.Pd., M.Kom.
NIP 199305282019032024 .....

Telah berhasil dipertahankan dihadapan dewan penguji pada tanggal 16 Agustus 2022 dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Terapan Program Studi Diploma IV Rekayasa Perangkat Lunak, Jurusan Teknik Informatika, Politeknik Negeri Indramayu.

DEWAN PENGUJI

Ketua Penguji : Fachrul Pralienka Bani Muhamad, S.ST.,M.Kom .....
NIP 199204232018031001

Anggota Penguji I : Iryanto, S.Si.,M.Si
NIP 199008012019031014 .....

Anggota Penguji II : A. Sumarudin, S.Pd., MT., M.Sc
NIP 198610102019031014 .....

Indramayu, September 2022

Ketua Jurusan Teknik Informatika

Iryanto, S.Si.,M.Si
NIP 199008012019031014

PERNYATAAN

Dengan ini saya menyatakan dengan sesungguhnya bahwa Skripsi ini adalah asli hasil karya saya sendiri serta Skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar Sarjana Terapan di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dirujuk dalam naskah ini dan dalam daftar pustaka.

Indramayu, Agustus 2022

Yang menyatakan

Abu Mushonnip

NIM 1805001

ABSTRAK

Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi merupakan sistem yang dapat mengelola dan memantau data lahan pertanian padi. Dugaan masalah muncul bahwa pengembangan aplikasi ini dapat menemui kendala pada saat melakukan pemeliharan atau penambahan fitur. Masalah yang ada yaitu pembacaan kode yang sulit dimengerti, sehingga aplikasi ini dilakukan *code refactoring* agar lebih mudah dalam melakukan pemeliharaan kedepannya. Konsep *clean code* yang diterapkan pada saat proses *code refactoring* yaitu *meaningful names, clean function, clean classess, clean code formating dan clean comments*. Hasil analisis yang dilakukan dengan menggunakan *tools* PhpMetrics menunjukkan bahwa setelah proses *code refectoring*, kode sumber dapat menjadi lebih baik.

Kata Kunci : *clean code*, *refactoring, smart farming*.

ABSTRACT

The Rice Farmland Management and Monitoring Application is a system that can manage and monitor rice farmland data. Alleged problems arise that the development of this application can encounter obstacles when maintaining or adding features. The existing problem is the reading of code that is difficult to understand, so this application is done code refactoring to make it easier to do maintenance in the future. The concept of clean code applied during the code refactoring process is meaningful names, clean functions, clean classess, clean code formatting and clean comments. The results of static analysis conducted using PhpMetrics tools show that after the code refectoring process, the source code can be better.

Keywords: clean code, refactoring, smart farming.

MOTO HIDUP

“melamunkan berbagai angan tak akan selesaikan masalah”

KATA PENGATAR

Puji dan syukur kita panjatkan kepada Allah SWT. yang telah memberikan hidayah-Nya kepada kita semua. Shalawat serta salam senantiasa tercurah untuk Nabi Muhammad SAW. beserta keluarganya, para sahabatnya, dan pengikutnya hingga akhir zaman.

Skripsi ini adalah mata kuliah yang wajib ditempuh oleh mahasiswa tingkat akhir khusunya yaitu diploma 4 dan sebagai salah satu persyaratan memperoleh gelar sarjana pada Program Studi Rekayasa Perangkat Lunak, Jurusan Teknik Informatika Politeknik Negeri Indramayu. Skripsi ini berjudul "Penerapan *Clean Code* Pada Pengembangan Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi".

Penulis juga menyadari dalam pembuatan laporan skripsi ini tidak terlepas dari dukungan, semangat dan bimbingan beberapa pihak, oleh karena itu dalam kesempatan ini penulis ingin mengucapkan rasa terimakasih kepada:

1. Bapak Casiman Sukardi, S.T., MT selaku Direktur Politeknik Negeri Indramayu.
2. Bapak Iryanto, S.Si., M.Si selaku Ketua Jurusan Teknik Informatika Politeknik Negeri Indramayu sekaligus.
3. Bapak A. Sumarudin, S.Pd., M.T., M.Sc selaku pembimbing 1 Skripsi.
4. Ibu Alifia Puspaningrum, S.Pd., M.Kom. selaku pembimbing 2 Skripsi.
5. Dosen Polindra Jurusan Teknik Informatika yang telah banyak memberi saran, ilmu, dan motivasi dalam menyelesaikan pendidikan D4 pada tahun ini.
6. Orang tua penulis yang telah memotivasi penulis secara lahir dan batin untuk semangat menyelesaikan skripsi.
7. Teman–Teman kelas D4 RPL A 2018 yang sudah sama-sama mau berjuang menyelesaikan studi dan sudah sering membantu penulis baik masalah akademik maupun luar akademik.
8. Dan seluruh pihak yang telah membantu dan mendukung.

Penulis menyadari bahwa dalam penyusunan Skripsi ini masih banyak terdapat kekurangan dan kelemahan yang dimiliki penulis baik itu sistematika penulisan maupun penggunaan bahasa. Untuk itu penulis mengharapkan saran dan

kritik dari berbagai pihak yang bersifat membangun demi penyempurnaan laporan ini. Semoga laporan ini berguna bagi pembaca secara umum dan penulis secara khusus. Akhir kata, penulis ucapkan banyak terima kasih.

Indramayu, Agustus 2021

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1. Latar Belakang

Indonesia adalah negara yang dikenal dengan sebutan negara agraris karena sebagian besar penduduknya bermata pencaharian di bidang pertanian. Pada Agustus 2020, Badan Pusat Statistik mencatat penduduk yang bekerja di sektor pertanian sebanyak 38,23 juta orang atau 29,76 persen dari jumlah penduduk bekerja yang jumlahnya 128,45 juta orang (Annur, 2020).

Di Indonesia beras menjadi salah satu makanan pokok yang sangat sulit digantikan, sehingga keberadaan beras menjadi prioritas utama bagi masyarakat dalam memenuhi kebutuhan sehari-hari. Untuk menghasilkan beras yang berkualitas baik, petani harus mengelola tanaman padi sebaik mungkin, mulai dari pemilihan benih, pengelolaan lahan, pemupukan, pengendalian OPT dan lain sebagainya.

Smart farming dapat digunakan para petani karena dapat mempermudah kegiatan pekerjaan para petani salah satunya dalam pemantauan lahan pertanian secara periodik dengan menggunakan sistem akuisisi data. Sistem akuisisi data berfungsi untuk mengambil dan mengumpulkan data dari lingkungan lahan pertanian.

Dengan adanya sistem monitoring tersebut, maka diperlukan adanya sistem manajemen pertanian yang berkelanjutan dan dapat memudahkan petani dalam mengelola lahan beserta perangkat yang tertanam.

Sebuah studi di antara 227 profesional IT menyatakan bahwa keterampilan yang harus dimiliki seorang programmer yaitu kemampuan untuk membaca, memahami, dan memodifikasi kode sumber yang ditulis oleh orang lain. Kualitas kode yang rendah memberikan pengaruh langsung terhadap perawatan sebuah program (Dietz, 2018).

Clean code merupakan suatu konsep atau petunjuk penulisan struktur kode yang bersih (*Clean*), dimana kode yang ditulis dapat dibaca oleh pengembang lain dan dapat diubah dengan mudah (Jackson, 1984).

Dengan didapatkannya dugaan masalah tersebut, penulis bermaksud merancang sebuah platform berupa aplikasi web yang memiliki kode sumber yang terukur baik sehingga aplikasi tersebut dapat dikembangkan dengan mudah oleh programmer lain.

1.2. Rumusan Masalah

Berdasarkan pemaparan latar belakang di atas, maka dapat diidentifikasi masalah pada penelitian ini yaitu:

1. Bagaimana rancangan Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi?
2. Bagaimana mengimplementasikan *clean code* pada pengembangan aplikasi web?
3. Seberapa besar pengaruh dari penerapan *clean code* pada *maintainability* perangkat lunak?

1.3. Batasan Masalah

Untuk memfokuskan pembahasan, dapat diperoleh beberapa batasan masalah, di antaranya:

1. Implementasi *clean code* dilakukan pada bagian *program backend*.
2. Pengukuran *maintainability* perangkat lunak dilakukan menggunakan *static analysis tool* yaitu PhpMetrics.

1.4. Tujuan Penelitian

Berdasarkan permasalahan yang telah dipaparkan, maksud dari penelitian ini yaitu mengimplementasikan *clean code* pada kode sumber dan menguji apakah penerapannya dapat meningkatkan *maintainability* perangkat lunak.

1.5. Manfaat Penelitian

Manfaat yang diperoleh dari penelitian ini adalah memberi gambaran kepada pengembang aplikasi web mengenai bagaimana *clean code* dapat diterapkan.

1.6. Sistematika Penulisan

Sistematika penulisan dalam laporan skripsi ini sebagai berikut.

BAB I : PENDAHULUAN

Pada bab ini penulis menguraikan mengenai latar belakang pemilihan judul untuk skripsi ini, perumusan masalah, batasan masalah, tujuan penelitian dan manfaat penelitian terkait judul yang dipilih oleh penulis serta sistematika penulisan dalam pembuatan laporan skripsi ini. Sehingga permasalahan tersebut memiliki titik fokus dan tidak melebar dari judul yang dibuat.

BAB II : TINJAUAN PUSTAKA

Pada bab ini membahas mengenai referensi penelitian yang pernah dilakukan sebelumnya yang berkaitan secara langsung dengan judul yang dipilih oleh penulis.

BAB III : METODOLOGI PENELITIAN

Pada bab ini penulis menguraikan tentang tahapan dan metode penelitian yang ditempuh untuk mencapai tujuan yang diharapkan, meliputi: metode dan tahapan penelitian, analisis kebutuhan sistem, gambaran sistem, dan perancangan diagram.

BAB IV : HASIL DAN PEMBAHASAN

Pada bab ini penulis menguraikan hasil penelitian, hasil implementasi, dan hasil pengujian.

BAB V : PENUTUP

Pada bab ini penulis memberikan kesimpulan dan saran yang diperoleh berdasarkan hasil pembuatan yang dapat meningkatkan dan mengembangkan penelitian ini di masa yang akan datang.

BAB II
TINJAUAN PUSTAKA

2.1. Studi Literatur

Terdapat beberapa penelitian lain mengenai pengembangan perangkat lunak yang terkait, sehingga penggalian literasi dilakukan dengan tujuan untuk mendapatkan informasi yang memiliki hubungan dengan penelitian terkait. Beberapa studi literatur tersebut sebagai berikut:

1. Penelitian yang dilakukan oleh Ahmad Faisol dari Institut Teknologi Nasional Malang dengan judul *"Pengaruh Pengujian Statis Terhadap Kualitas Perangkat Lunak Sitagih Pada PT. Semesta Mitra Sejahtera (SMS)"* (Faisol, A, dkk. 2021).
2. Penelitian yang dilakukan oleh Daniyal Ahmad Rizaldhi dari Universitas Komputer Indonesia dengan judul *"Implementasi Clean Code Dan Design Pattern Untuk Meningkatkan Maintainability Pada Aplikasi Content Marketing"* pada tahun 2019 (Rizaldhi, 2019).
3. Penelitian yang dilakukan oleh Ilham Prasetyo dari Universitas Komputer Indonesia dengan judul "*Implementasi Clean Code Dan Code Refactoring Untuk Meningkatkan Maintainability Pada Luar sekolah Bot (Lusa Bot)*" pada tahun 2021. Dalam laporannya, peneliti menyimpulkan bahwa konsep *clean code* secara umum memberikan petunjuk kepada para pengembang untuk menulis kode sumber yang mudah dibaca agar pengembang lain dapat memahami maksud dari kode yang ditulis tersebut (Prasetyo, 2021).

2.2. *Software Quality Assurance* (SQA)

Software Quality Assurance (SQA) merupakan serangkaian kegiatan untuk memastikan kualitas dalam proses rekayasa perangkat lunak. Ini memastikan bahwa perangkat lunak yang dikembangkan memenuhi dan sesuai dengan spesifikasi yang ditentukan atau standar.

Ada beberapa model faktor kualitas perangkat lunak, diantaranya yaitu yang dikemukakan oleh McCall, yang juga direkomendasikan oleh banyak peneliti selama bertahun-tahun (D. Galin, 2004). Model ini mengklasifikasikan semua kebutuhan perangkat lunak ke dalam 11 faktor kualitas perangkat lunak. 11 faktor

dikelompokkan menjadi tiga kategori, yaitu *Product Operation (Correctness, Reliability, Efficiency, Integrity, Usability), Product Revision (Maintainability, Flexibility, Testability), dan Product Transition (Portability, Reusability, Interoperability).*

2.3. *Maintainability*

Maintainability merupakan salah satu faktor pada SQA yang berfokus pada bagaimana perangkat lunak agar mudah dipelihara. Menurut (D. Galin, 2004), faktor ini mengacu pada struktur modul perangkat lunak, dokumentasi program internal, dan panduan *programmer.* Salah satu contoh kebutuhan *maintainability* yaitu pengembang mengikuti standar dan pedoman penulisan kode dari perusahaan.

2.4. *Maintainability Index*

Maintainability Index adalah *software metric* yang mengukur sebuah perangkat lunak mudah atau sulit untuk mengalami perawatan atau perubahan di masa mendatang. *Maintainability Index* terdiri atas metrik *Halstead Volume* (HV), metrik *Cyclomatic Complexity* (CC), rata-rata jumlah baris kode per modul (LOC), dan persentase jumlah komentar per modul (COM) (Stapelberg, 2009). Adapun formula *Maintainability Index* ditujukan pada persamaan berikut.

$$MI = 171 - 5.2 * \ln(\mathrm{HV}) - 0.23 * (\mathrm{CC}) - 16.2 * \ln(\mathrm{LOC}) + 50 * \sin(\sqrt{2.4 * \mathrm{LOC}})$$

Semakin tinggi nilai *maintainability index* tersebut menandakan tingginya *maintainability* dari kode sumber yang diukur. Nilai tersebut terbagi menjadi tiga kategori yang dapat dilihat pada Tabel berikut:

Table 2. 1 Tabel Nilai *Maintainability Index*

Nilai MI	Keterangan
MI > 85	Dapat dipelihara dengan baik
65 < MI ≤ 85	Cukup untuk dapat dipelihara

MI ≤ 65	Sulit untuk dipelihara

2.5. *Clean Code*

Clean code adalah kode sumber dalam perangkat lunak dengan format yang benar dan disusun dengan rapi dan baik sehingga programmer lain dapat membaca dan memodifikasi kode sumber tersebut tanpa harus menanyakan terlebih dahulu kepada programmer sebelumnya (Arhandi, Pramitarini, & Alviandra, 2019). *Clean code* juga bertujuan untuk mengatasi penurunan tingkat produktivitas pengembangan perangkat lunak akibat dari struktur kode yang berantakan. Adapun hal inti yang dibahas dalam c*lean code* menurut (Martin, 2008) diantaranya sebagai berikut:

2.5.1. *Meaningful Names*

Pada konsep ini terdapat petunjuk dalam melakukan pemberian nama terhadap variabel, *method*/fungsi, dan *class*/modul agar lebih mudah untuk dipahami. Berikut merupakan petunjuk yang dapat digunakan dalam melakukan pemberian nama:

2.5.1.1. Nama Yang Dapat Dieja Dan Memiliki Arti

Menggunakan nama yang dapat dieja dan secara eksplisit menerangkan kegunaan dari kode yang ditulis dapat memudahkan pengembang lain untuk memahaminya. Contoh panduan ini ditunjukkan pada Gambar 2.1

```
Buruk
$yyyymmdstr = date('Y/m/d');

Baik
$currentDate = date('Y/m/d');
```

Gambar 2. 1 Contoh Nama Yang Dapat Dieja Dan Memiliki Arti

2.5.1.2. Nama Yang Mudah Untuk Dicari

Petunjuk ini digunakan untuk mempermudah pencarian kode. Dengan menggunakan nama yang mudah ditemukan, pengembang tidak perlu memahami keseluruhan program. Hal ini sering digunakan untuk nama konstanta. Contoh panduan ini dapat dilihat pada Gambar 2 .2

```
Buruk
setTimeout(blastOff, 86400000);

Baik
define("MILLISECONDS_IN_A_DAY", 86400000);
setTimeout(blastOff, MILLISECONDS_IN_A_DAY);
```

Gambar 2. 2 Contoh Nama Yang Mudah Untuk Dicari

2.5.1.3. Menghindari *Mental Mapping*

Menggunakan singkatan seperti **`i`** dan **`n`** untuk menamai iterator dan counter, atau menggunakan singkatan lain yang umumnya tidak diketahui orang dapat memperumit proses pemahaman. Oleh karena itu, pengembang harus menghindari penggunaan singkatan ini. Contoh panduan ini dapat dilihat pada Gambar 2 .

```
Buruk
const locations = ["Bandung", "Cimahi", "Sumedang"];
locations.forEach(l => {
    doStuff();
    // ...
    dispatch(l);
});

Baik
const locations = ["Bandung", "Cimahi", "Sumedang"];
locations.forEach(location => {
    doStuff();
    // ...
    dispatch(location);
});
```

Gambar 2. 3 Contoh Menghindari *Mental Mapping*

2.5.1.4. Kata Benda Untuk *Class*/Modul

Dalam melakukan pemberian nama terhadap *class*/modul, disarankan untuk menggunakan kata benda dan menghindari penggunaan kata kerja. Hal tersebut didasari oleh penggunaan *class*/modul yang biasanya digunakan mewakili suatu entitas. Adapun contohnya seperti penggunaan kata *User, LoginController,* dan *MessageParser.*

2.5.1.5. Kata Kerja Untuk *Method*/Fungsi

Penamaan pada method/fungsi disarankan untuk menggunakan kata kerja. Hal tersebut dilakukan untuk mendeskripsikan tujuan dari pekerjaan yang dilakukan oleh *method*/fungsi yang ditulis secara eksplisit. *Method*/fungsi

accessors, *mutators*, dan *predicates* biasanya memiliki prefiks *set*, *get*, dan *is*. Contoh panduan ini dapat dilihat pada Gambar 2.4

```
$name = employee.getName();
customer.setName("Mike");
if (paycheck.isPosted()){
    printInvoice();
}
```

Gambar 2. 4 Contoh Kata Kerja Untuk *Method*/Fungsi

2.5.1.6. Menghindari Penggunaan Konteks Yang Tidak Diperlukan

Ini biasa terjadi di dalam *class*/modul atau pada objek, dimana atributnya mengandung nama dari *class*/modul atau objek tempat atribut tersebut berada. Hal tersebut tidak disarankan, karena bersifat repetitif. Contoh panduan ini dapat dilihat pada Gambar 2.5

```
Buruk
class Car {
    public $carWheels;
    public $carEngine;
    public $carDoors;
    public $carColor;

    function paintCar($newColor) {
        $this->carColor = $newColor;
    }
}

Baik
class Car {
    public $wheels;
    public $engine;
    public $doors;
    public $color;

    function paintCar($newColor) {
        $this->color = $newColor;
    }
}
```

Gambar 2. 5 Contoh Penggunaan Konteks

2.5.2. *Clean Functions*

Konsep ini memiliki petunjuk untuk menulis *method*/fungsi yang bersih agar lebih mudah dipahami. Di bawah ini adalah petunjuk untuk digunakan saat menulis *method*/fungsi.

2.5.2.1. Jumlah Parameter

Konsep ini memiliki panduan untuk menulis *method*/fungsi yang bersih agar lebih mudah dipahami. Di bawah ini adalah panduan untuk digunakan saat menulis *method*/fungsi.

```
Buruk
function createMenu(title, body, buttonText, cancellable) {
    // ...
}

Baik
function createMenu(Menu $menu) {
    // ...
}
```

Gambar 2. 6 Contoh *Clean function* (Jumlah Parameter)

2.5.2.2. Jumlah Pekerjaan

Suatu *method*/fungsi disarankan untuk melakukan hanya satu pekerjaan saja. Suatu *method*/fungsi yang melakukan lebih dari satu pekerjaan akan sulit untuk dibuat, diuji, dan dipahami. *Method*/fungsi yang lebih kecil akan lebih mudah untuk dimodifikasi. Apabila *method*/fungsi yang ditemukan melakukan lebih dari satu pekerjaan, pecah pekerjaan-pekerjaan tersebut ke dalam *method*/fungsi yang berbeda. Contoh petunjuk ini dapat dilihat pada Gambar 2.7

```
Buruk
function emailClients(clients) {
    clients.forEach(client => {
        const clientRecord = database.lookup(client);
        if (clientRecord.isActive()) {
            email(client);
        }
    });
}

Baik
function emailActiveClients(clients) {
    clients.filter(isActiveClient).forEach(email);
}
function isActiveClient(client) {
    const clientRecord = database.lookup(client);
    return clientRecord.isActive();
}
```

Gambar 2. 7 Contoh *Clean function* (Jumlah Pekerjaan)

2.5.3. *Clean Classes*

Pada konsep ini terdapat petunjuk dalam membuat *class*/modul yang lebih bersih dan terorganisir. Berikut merupakan petunjuk yang dapat dalam membuat *class*/modul:

2.5.3.1. *Class Organization*

Pembuatan *class*/modul yang baik dapat dilakukan dengan mengikuti code *convetion/code styling* dari bahasa pemrograman yang digunakan seperti PSR-12 pada bahasa pemrograman PHP, dan ES6 pada bahasa pemrograman Javascript.

2.5.3.2. *Single Responsibility Principle*

Dengan menjaga ukuran *class*/modul untuk tidak terlalu besar, dapat mempermudah proses modifikasi. Ukuran suatu *class*/modul diukur berdasarkan jumlah *responsibility* yang ditanggung oleh class/modul tersebut. *Class*/modul yang bersih merupakan *class*/modul yang hanya memiliki satu *responsibility* saja. Hampir sama dengan *clean function*, apabila *class*/modul tersebut memiliki lebih dari satu *responsibility*, pisahkan *responsibility* ke dalam *class*/modul yang berbeda.

2.5.4. *Clean Comments*

Pada konsep ini terdapat petunjuk dalam melakukan pemberian komentar yang baik dan efisien, agar komentar yang ditulis dapat memberikan informasi yang tidak tersampaikan oleh kode sumber yang sudah ditulis. Akan tetapi, pemanfaatan *clean comment* berhubungan dengan pemanfaatan *meaningful names*, di mana semakin jelas penamaan dari suatu variabel, *method*/fungsi, maupun *class*/modul semakin sedikit pula komentar yang harus diberikan.

2.5.5. *Clean Code Formatting*

Dalam melakukan penulisan kode sumber diperlukan pemanfaatan format penulisan, hal ini meliputi penggunaan indentation, spasi, tab dan penempatan kode berdasarkan penggunaannya. Penentuan format penulisan dapat mengikuti *code convention/code styling* dari bahasa pemrograman yang digunakan atau mengikuti kesepakatan dari tim pengembang.

2.6. *Code Refactoring*

Code refactoring adalah proses mengubah sistem perangkat lunak tanpa mengubah perilaku eksternal dari kode tetapi ditujukan untuk memperbaiki struktur internal dari kode tersebut (Fowler dan Beck, 2002).

2.7. *Static Code Analysis*

Static code analysis merupakan konsep untuk mengaudit kode sumber untuk diidentifikasi kualitasnya dan mengidentifikasi potensi kerentanan keamanan. Tidak seperti analisis kode sumber dinamis yang mengevaluasi perilaku kode sumber selama eksekusi kode.

Kelebihan dari s*tatic code analysis* pada aplikasi adalah kemungkinan jumlah cacat yang dapat ditemukan jauh lebih cepat sehingga biaya perbaikan bisa lebih murah jika dibandingkan dengan melakukan pengujian dinamis.

2.8. Teknologi Terapan

2.8.1. Laravel

Laravel adalah sebuah framework PHP yang dirilis dibawah lisensi MIT dan dibangun dengan konsep MVC (*Model View Controller*). Laravel adalah pengembangan website berbasis MVP yang ditulis dalam PHP yang dirancang untuk meningkatkan kualitas perangkat lunak dengan mengurangi biaya pengembangan awal dan biaya pemeliharaan, serta untuk meningkatkan pengalaman bekerja dengan aplikasi dengan menyediakan sintaks yang ekspresif, jelas, dan menghemat waktu. Sebagai sebuah framework PHP, Laravel itu sendiri bersifat open source yang dibuat menarik di bagian sintaknya yang ekspresif dan elegan dan dirancang khusus untuk memudahkan dan mempercepat proses *web developmnet* (Yudhanto, 2018).

2.8.2. PostgreSQL

PostgreSQL adalah database relasional open source yang mendukung kueri SQL (relasional) dan JSON (non-relasional). PostgreSQL adalah sistem manajemen basis data yang sangat stabil, didukung oleh lebih dari 20 tahun pengembangan oleh komunitas *open source*. PostgreSQL digunakan sebagai database utama untuk banyak aplikasi web serta aplikasi seluler dan analitik.

2.8.3. PhpMetrics

PhpMetrics adalah perangkat lunak *static code analysis* yang berfungsi untuk menghitung *software metric* pada bahasa pemrograman PHP. PhpMetrics merupakan perangkat lunak open source yang dikembangkan oleh Jean-François Lépine dan mengakomodir beberapa metrics yang mengukur tingkat kompleksitas aplikasi, jumlah *code* atau variabel tertentu, data terkait *object oriented*, maupun tingkat *maintainability*.

2.8.4. PHP CodeSniffer

PHP CodeSniffer atau PHPCS merupakan alat yang berfungsi untuk mendeteksi *violation* atau pelanggaraan terhadap *coding standard.*

2.9. Pemodelan Perangkat Lunak

2.9.1. *Unified Modeling Languange (UML)*

2.9.1.1. *Use Case Diagram*

Use Case menggambarkan *external view* dari sistem yang akan kita buat modelnya (Widodo, 2012). Model *Use Case* dapat dijabarkan dalam diagram *Use Case*, tetapi perlu diingat, diagram tidak indentik dengan model karena model lebih luas dari diagram (Pooley, 2003:15). *Use Case* harus mampu menggambarkan urutan aktor yang menghasilkan nilai terukur. (Widodo, 2012).

2.9.1.2. *Class Diagram*

Class Diagram sebagai suatu objek yang memiliki atribut dan perilaku yang sama kelas kadang disebut kelas objek. Class memiliki tiga area pokok yaitu Nama, Atribut dan Operasi. (Haviluddin, 2011).

2.9.1.3. *Activity Diagram*

Activity Diagram juga dapat menggambarkan proses lebih dari satu aksi dalam waktu bersamaan. Menurut Haviluddin (2011) “Diagram *Activity* adalah aktivitas-aktivitas, *object*, *state*, transisi *state* dan *event*. Dengan kata lain kegiatan diagram alur kerja menggambarkan perilaku sistem untuk aktivitas”.

2.9.2. *Entity Relationship Diagram (ERD)*

Entity Relationship Diagram merupakan teknik yang digunakan untuk memodelkan kebutuhan data dari suatu organisasi, biasanya oleh Sistem analis dalam tahap analisis persyaratan proyek pengembangan sistem. *ERD*

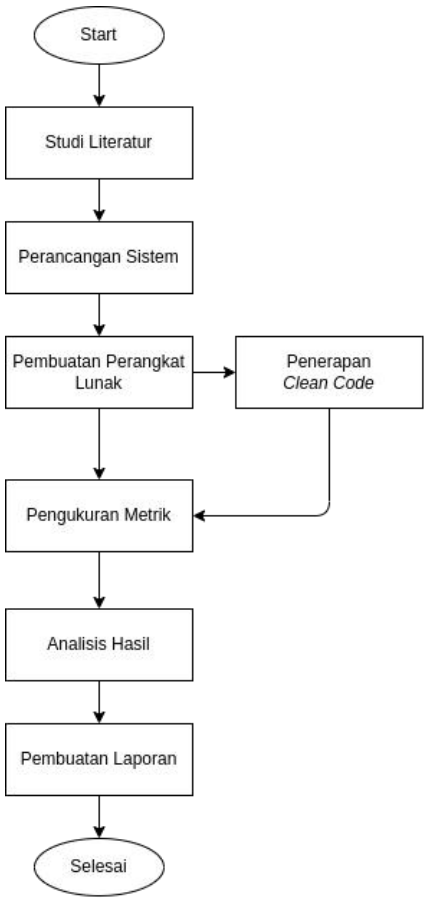
bersama-sama dengan detail pendukung merupakan model data yang pada gilirannya digunakan sebagai spesifikasi untuk *Database*. Dalam pembentukan *ERD* terdapat tiga komponen yang akan dibentuk yaitu Entitas, Hubungan, dan Atribut. (Mulyani, 2016).

BAB III
METODOLOGI PENELITIAN

Pada bab ini akan membahas metode penelitian yang digunakan penulis dalam penelitian perancangan dan implementasi *clean code* pada Aplikasi Manajemen dan Monitoring Lahan Pertanian Padi.

3.1. Metode Penelitian

Penelitian ini secara umum memiliki tahapan-tahapan diantaranya (1) Studi Literatur, (2) Perancangan Sistem, (3) Pembuatan Perangkat Lunak, (4) Penerapan *Clean Code,* (5) Pengukuran Metrik, (6) Pembuatan Laporan. Ilustrasi alur penelitian dapat dilihat pada gambar di bawah ini.



Gambar 3. 1 Tahapan Penelitian

3.2. Perancangan Sistem

Analisa kebutuhan diperlukan untuk pembuatan perangkat lunak, adapun untuk mendapatkan gambaran sistem yang dibuat, proses dan data dimodelkan dengan *Unified Modeling Languange (UML)* dan *Entity Relationship Diagram (ERD).*

3.2.1. Kebutuhan Fungsional

Dalam merancang Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi, diuraikan beberapa kebutuhan fungsional (Tabel 3.1) sebagai berikut.

Tabel 3. 1 Kebutuhan Fungsional

No.	Kebutuhan Fungsional	*Role*
1.	Login	Admin
2.	Mengelola data Petani	
3.	Mengelola data Lahan	
4.	Mengelola data Perangkat	
5.	Register	Farmer
6.	Login	
7.	Mengelola data Lahan	
8.	Mengelola data Perangkat	
9.	Memonitoring data Sensor	
10.	Mendapatkan rekomendasi pengairan	
11.	Mendapatkan rekomendasi pemupukan	
12.	Melihat kualitas tanaman padi	

3.2.2. Kebutuhan Non-Fungsional

Dalam merancang Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi, diuraikan beberapa kebutuhan non-fungsional (Tabel 3.2) sebagai berikut.

Tabel 3. 2 Kebutuhan Non-Fungsional

No.	Kebutuhan Non Fungsional
1.	Sistem menggunakan template *Bootstrap CSS*
2.	Sistem memiliki tampilan yang *responsive*

3.2.3. Lingkungan Pengembangan

Adapun lingkungan pengembangan dalam merancang Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi, dapat dilihat pada Tabel 3.3.

Tabel 3. 3 Kebutuhan Perangkat Lunak

No	Jenis Software	Kebutuhan Software
1.	Sistem Operasi	Windows 10/ Linux Kernel 5.10
2.	Bahasa Pemrograman	PHP dengan *framework* Laravel 9.2
3	*Text Editor*	Visual Studio Code
4.	DBMS	PostgreSQL 14.3

3.2.4. Kebutuhan Perangkat Keras

Adapun kebutuhan perangkat keras dalam merancang Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi, dapat dilihat pada Tabel 3.4.

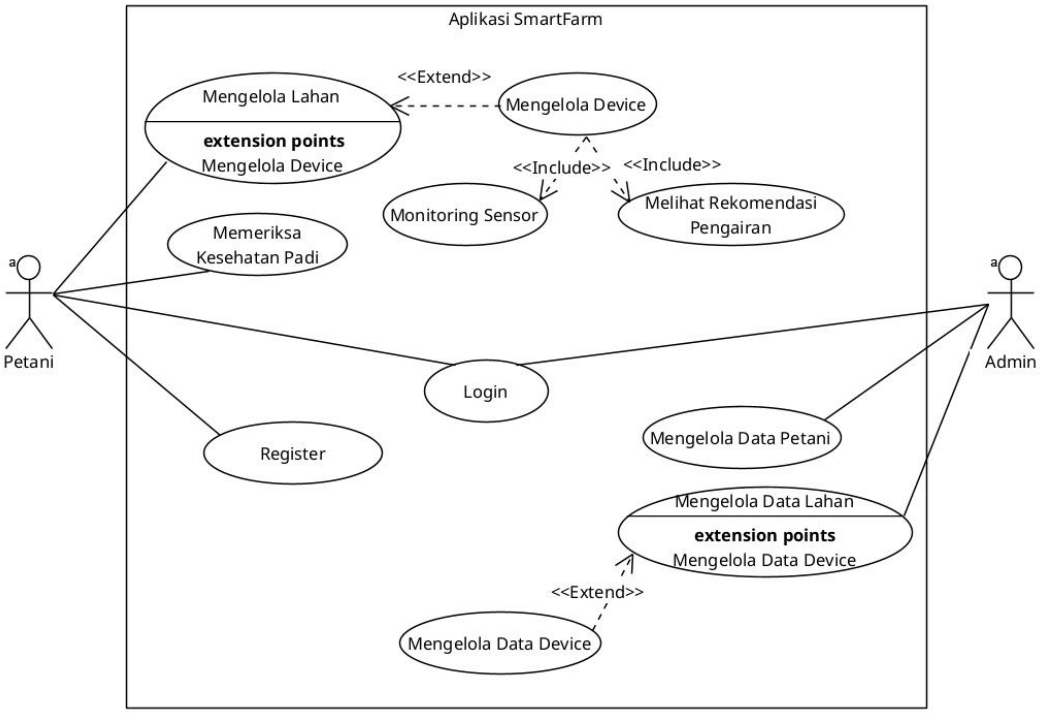
Tabel 3. 4 Kebutuhan Perangkat Keras

No.	Jenis Hardware	Kebutuhan Hardware
1.	*Processor*	AMD Ryzen 5 5500U @ 2.100GHz
2.	*Memory*	12 GB
3.	*Storage*	512 GB
4.	*Display*	Full HD 1920x1080

3.2.5. *Unified Modeling Languange (UML)*

3.2.5.1. *Use Case Diagram*

Pada Gambar 3.2 merupakan *use case diagram* dari Aplikasi Manajemen dan Monitoring Lahan Pertanian Padi.



Gambar 3. 2 Use Case Diagram

Adapun penjelasan dari gambar 3.2 *Use Case Diagram* adalah pada tabel-tabel berikut ini:

Tabel 3. 5 Penjelasan *Use Case Diagram* Admin

Aktor	Nama *Use Case*	Keterangan
Admin	*Login*	Admin dapat masuk ke sistem menggunakan akun yang telah dibuat pada database
	Mengelola Data Petani	Admin dapat mengelola data akun petani
	Mengelola Data Lahan	Admin dapat mengelola data lahan yang dimiliki petani

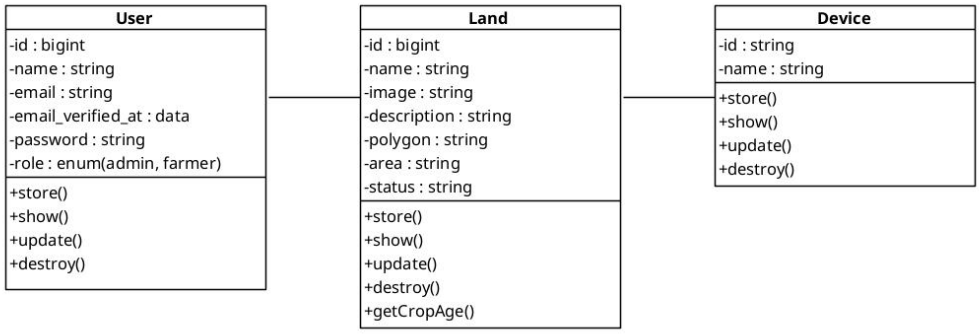
Tabel 3. 6 Penjelasan *Use Case Diagram* Petani

Aktor	Nama *Use Case*	Keterangan
Petani	*Register*	Petani dapat mendaftarkan akun untuk bisa mendapatkan akses masuk ke aplikasi
	Login	Petani dapat masuk ke aplikasi dengan memasukkan *email* dan

		password
	Mengelola Lahan	Petani dapat mengelola lahan beserta deskripsinya
	Mengelola *Devices*	Petani dapat mengelola *device* yang tertanam pada lahan tertentu

3.2.5.2. *Class Diagram*

Adapun struktur *class diagram* pada *Unified Modeling Languange (UML)* dapat dilihat pada Gambar 3.3.



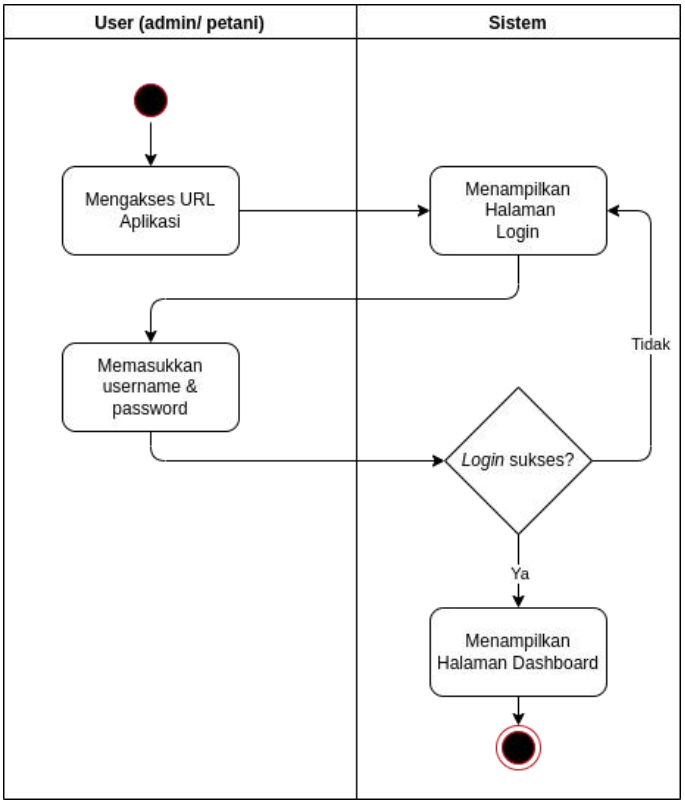
Gambar 3. 3 Class Diagram

3.2.5.3. *Activity Diagram*

Activity Diagram menunjukkan urutan aktivitas atau penggunaan yang dilakukan oleh setiap kebutuhan fungsional yang ada pada sistem. *Activity Diagram* juga dapat menggambarkan proses lebih dari satu aksi dalam waktu bersamaan. Berikut ini merupakan Activity Diagram yang mewakili setiap kebutuhan fungsional sistem.

3.2.5.3.1. Login

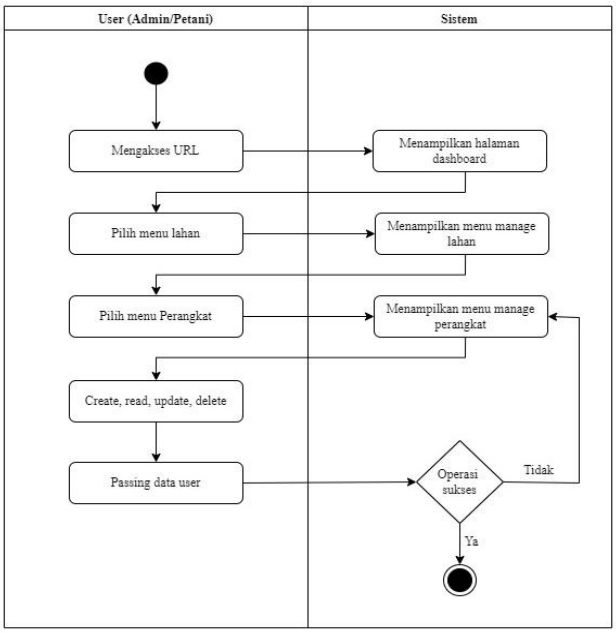
Adapun urutan aktivitas pada kebutuhan *Login* admin dapat dilihat pada Gambar 3.3.



Gambar 3. 4 *Activity Diagram Login*

3.2.5.3.2. Mengelola Data Petani

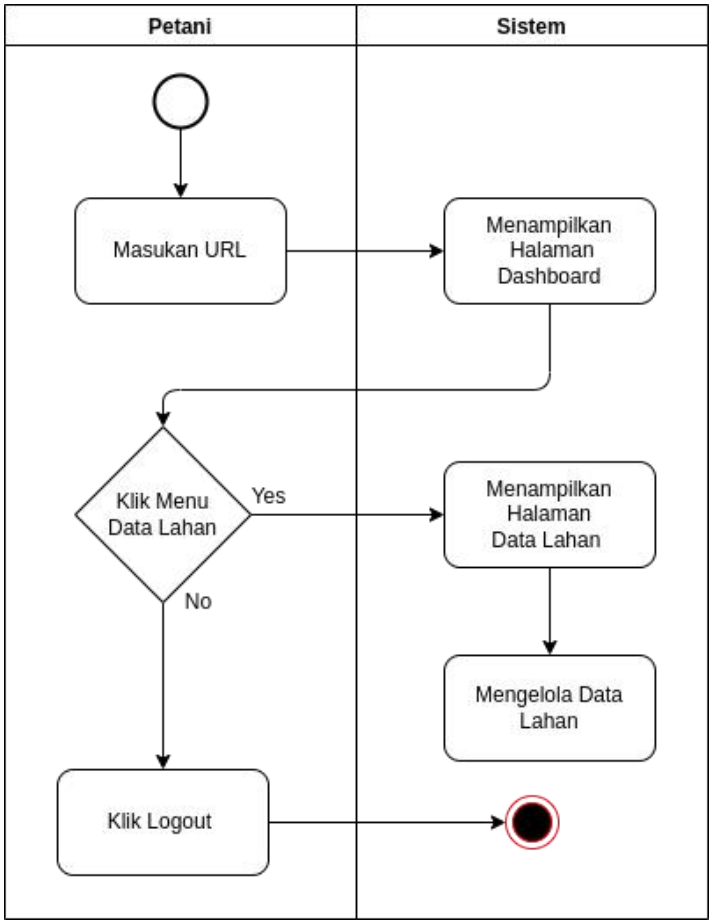
Adapun urutan aktivitas pada kebutuhan mengelola data petani dapat dilihat pada Gambar 3.4.



Gambar 3. 5 *Activity Diagram* Mengelola Data Petani

3.2.5.3.3. Mengelola Data Lahan

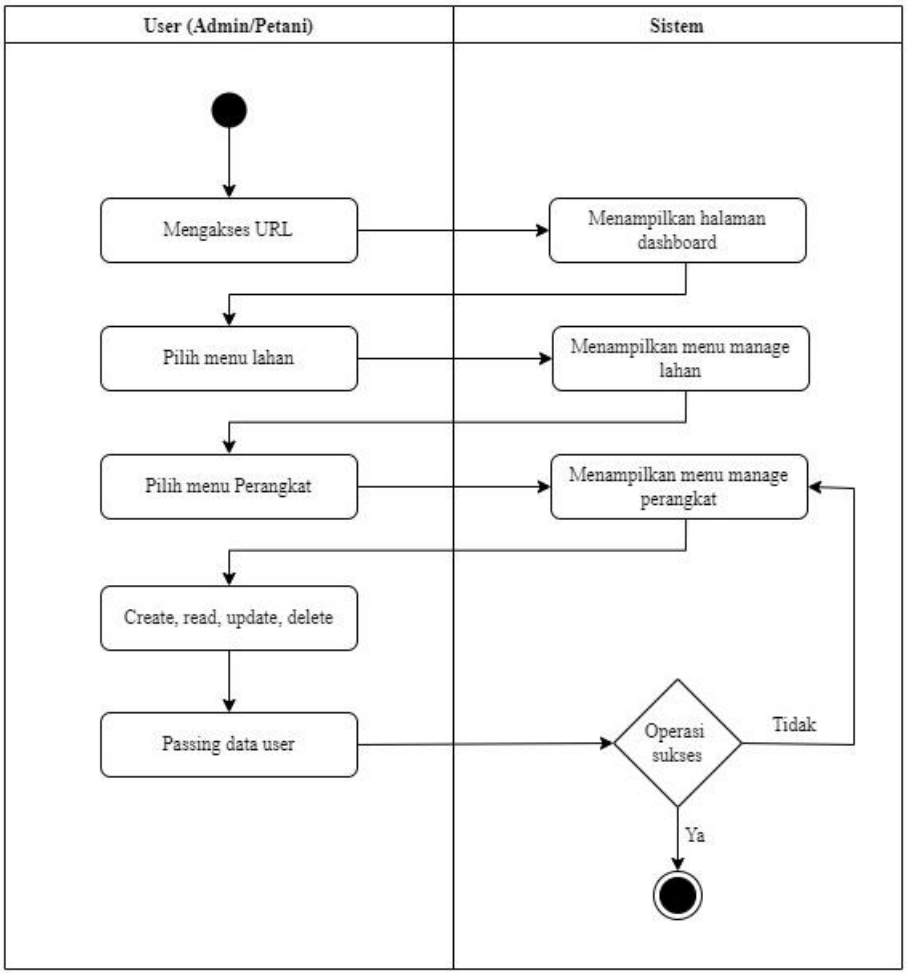
Adapun urutan aktivitas pada kebutuhan mengelola data lahan dapat dilihat pada Gambar 3.5.



Gambar 3. 6 *Activity Diagram* Mengelola Data Lahan

3.2.5.3.4. Mengelola Data Perangkat

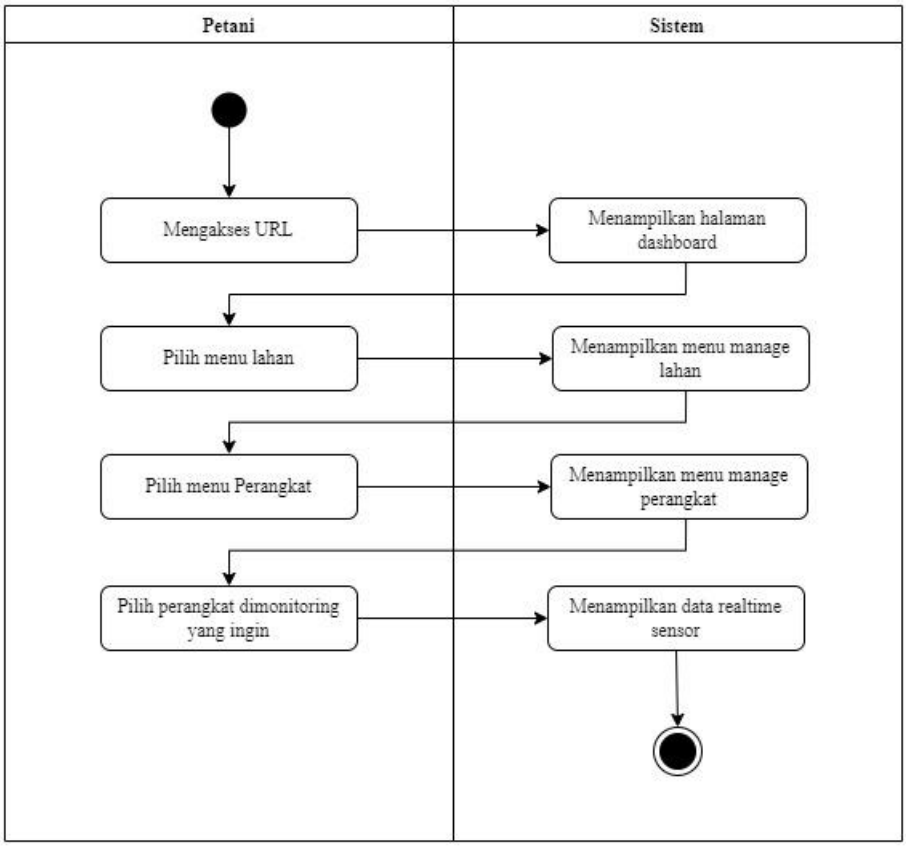
Adapun urutan aktivitas pada kebutuhan mengelola data perangkat dapat dilihat pada Gambar 3.6.



Gambar 3. 7 *Activity Diagram* Mengelola Data Perangkat

3.2.5.3.5. Monitoring Data Sensor

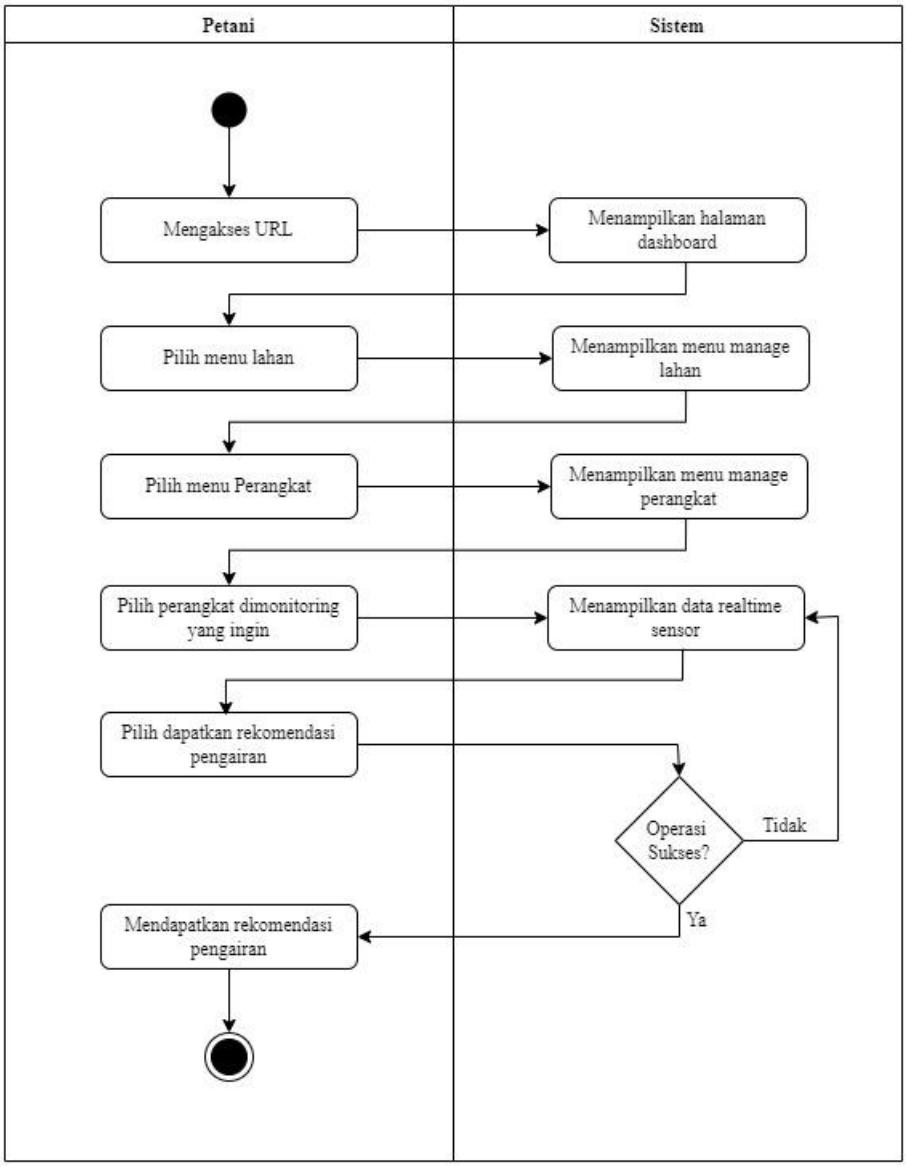
Adapun urutan aktivitas pada kebutuhan mengelola data sensor dapat dilihat pada Gambar 3.7.



Gambar 3. 8 *Activity Diagram* Monitoring Data Sensor

3.2.5.3.6. Mendapatkan Rekomendasi Pengairan

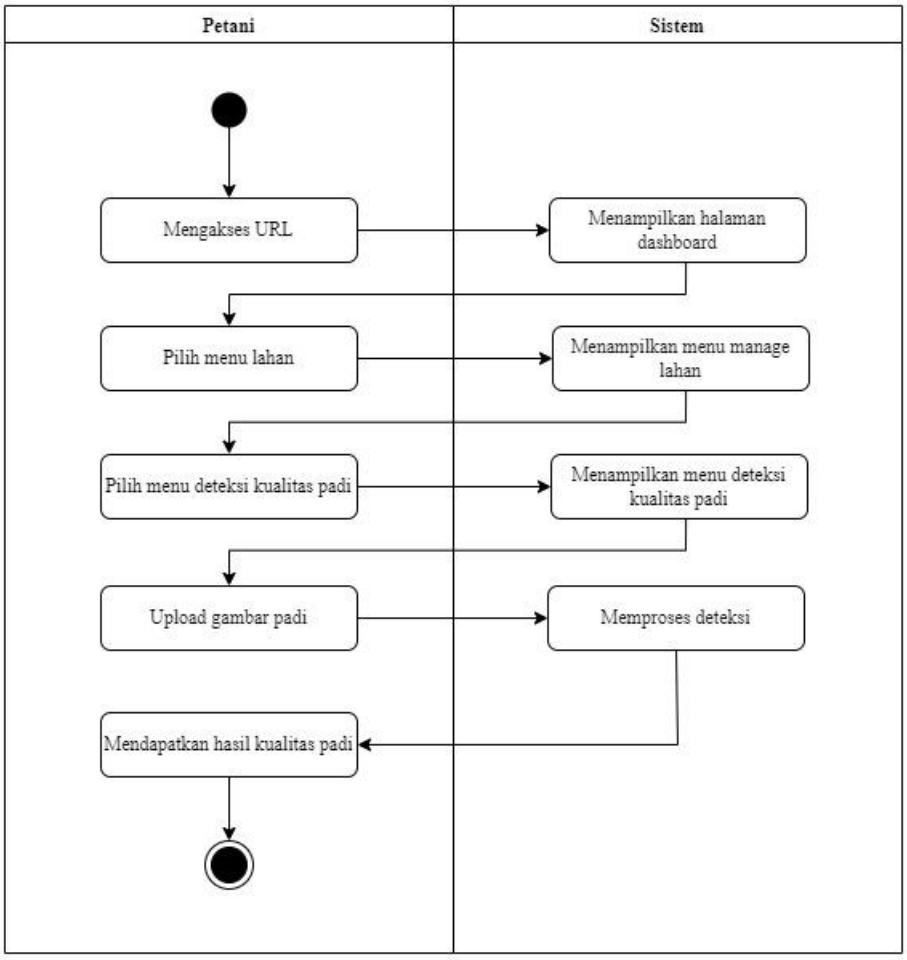
Adapun urutan aktivitas pada kebutuhan mendapatkan rekomendasi pengairan dapat dilihat pada Gambar 3.8.



Gambar 3. 9 *Activity Diagram* Rekomendasi Pengairan

3.2.5.3.7. Melihat Kualitas Tanaman Padi

Adapun urutan aktivitas pada kebutuhan melihat kualitas tanaman padi dapat dilihat pada Gambar 3.9.



Gambar 3. 10 *Activity Diagram* Melihat Kualitas Tanaman Padi

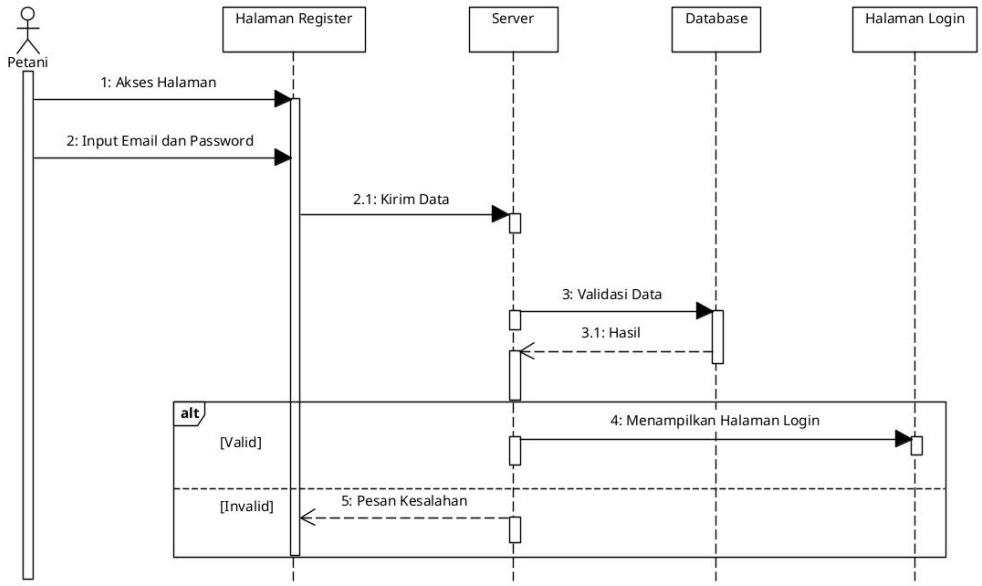
3.2.5.4. *Sequence Diagram*

Sequence Diagram mengambarkan aliran fungsional urutan interaksi yang terjadi di dalam sistem Admin dan Petani, Diagram ini menunjukan serangkaian pesan yang dipertukarkan oleh objek-objek yang melakukan suatu tugas atau aksi tertentu. Objek-objek tersebut kemudian diurutkan dari kiri ke kanan, aktor yang menginisiasi interaksi tersebut berada di paling kiri diagram.

3.2.5.4.1 *Sequence Diagram Register*

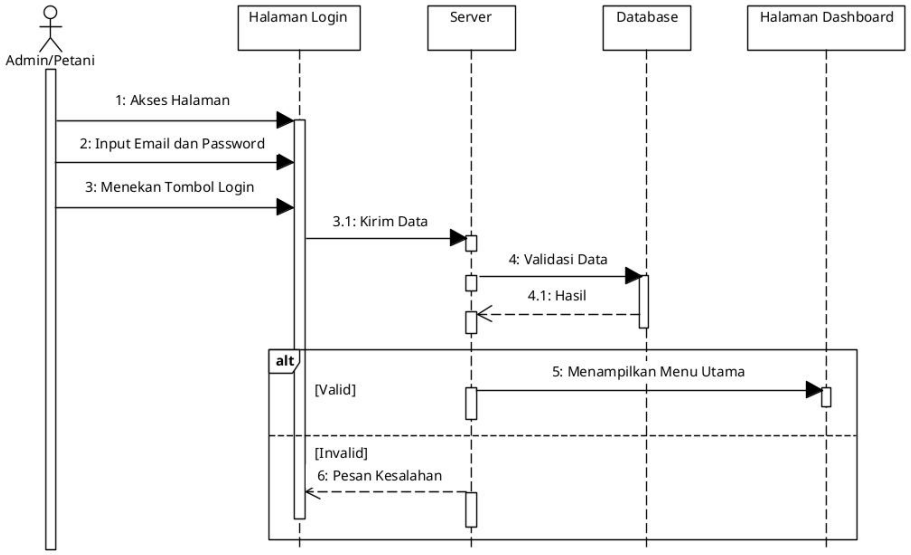
Sequence Diagram berikut menggambarkan proses registrasi untuk petani. Prosesnya dimulai dengan mengakses halaman registrasi, kemudian memasukkan data email dan password. Data tersebut kemudian dikirim ke *server* dan

melakukan validasi untuk email yang digunakan apakah sudah digunakan pada *database.*



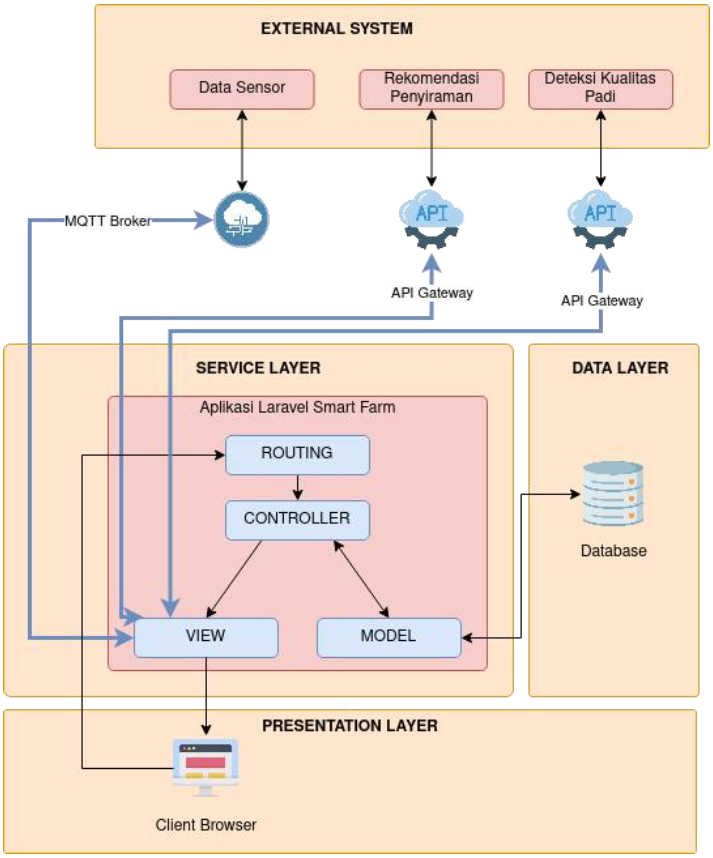
3.2.5.4.2 *Sequence Diagram Login*

Sequence Diagram berikut menggambarkan proses login untuk admin maupun petani. Prosesnya dimulai dengan mengakses halaman login, kemudian memasukkan data email dan password. Data tersebut kemudian dikirim ke *server* dan melakukan validasi untuk kombinasi email dan password.



3.2.6. Arsitektur Aplikasi

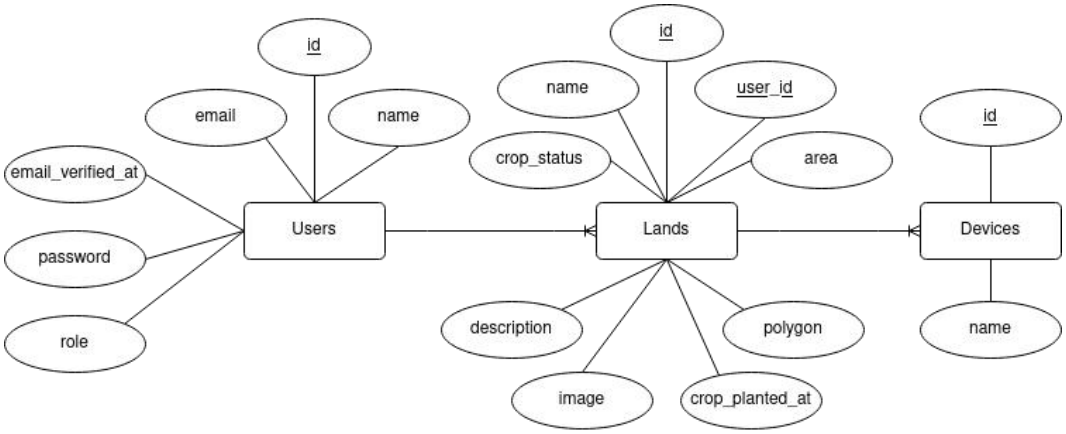
Secara umum sistem yang dibuat memiliki 4 layer: (1) *Service Layer* berisikan perangkat lunak utama *frontend* dan *backend* dari Laravel, (2) *Data Layer* berisikan PostgreSQL *database server* yang digunakan untuk menyimpan data pada perangkat lunak. (3) *External System* merupakan dependensi perangkat lunak yang berisikan *service* untuk pengambilan data sensor dan rekomendasi penyiraman, dan deteksi kualitas padi. (4) *Presentation Layer* dewasa ini perangkat lunak yang dikembangkan dapat diakses melalui *web browser*.



Gambar 3. 11 Rancangan Arsitektur Sistem

3.2.7. Perancangan ERD (*Entity Relationship Diagram*)

Pada Gambar 3.4 merupakan gambar *Entity Relationship Diagram* (ERD) yang digunakan untuk proses aplikasi *Smart Farm*. Adapun penjelasan dari Gambar 3.4 adalah pada Tabel 3.3 berikut ini:



Gambar 3. 12 Perancangan ERD

Tabel 3. 7 Penjelasan ERD

Tabel 3. 8 Penjelasan ERD

Nama Tabel	Keterangan
Users	Tabel yang digunakan untuk menyimpan data user
Lands	Tabel yang digunakan untuk menyimpan data lahan petani
Device	Tabel yang digunakan untuk menyimpan data device

Pada Aplikasi *Smart Farm* ini terdiri dari 3 tabel yaitu *Users, Lands, Devices.*

3.5 Perancangan Database

Berikut ini merupakan tabel dari perancangan Aplikasi *Smart Farm*.

3.5.1 Tabel *Users*

Tabel users digunakan untuk meyimpan data Pengguna Pengguna, terdiri dari *field id, name, email, email_verified, password, role, remember_token, deleted_at, created_at, update_at*. Adapun tabel *Users* dapat dilihat pada Tabel 3.9.

Tabel 3. 9 Tabel *Users*

No	Nama Field	Tipe Data	Keterangan
1	*id*	Bigint(255)	Untuk menyimpan id pengguna
2	*name*	Varchar(255)	Untuk menyimpan nama pengguna

3	*email*	Varchar(255)	Untuk menyimpan email pengguna
4	*email_verified_at*	Timestamp	Untuk menyimpan waktu email terverifikasi
5	*password*	Varchar(255)	Untuk menyimpan password pengguna
6	*role*	Varchar(255)	Untuk menyimpan *role* pengguna (admin/ *farmer*)
7	*deleted_at*	Timestamp	Untuk menyimpan waktu dihapus
8	*created_at*	Timestamp	Untuk menyimpan waktu dibuat
9	*update_at*	Timestamp	Untuk menyimpan waktu diupdate

3.5.2 Tabel Device

Tabel *device* digunakan untuk menyimpan Data *device* terdiri dari *id, land_id, name, created_at, updated_at*. Adapun tabel *device* dapat dilihat pada Tabel 3.5.

Tabel 3. 10 Tabel *Devices*

No	Nama Field	Tipe Data	Keterangan
1	*id*	Varchar(255)	Untuk menyimpan id Perangkat
2	*land_id*	Bigint(255)	Untuk menyimpan id Lahan
3	*name*	Varchar(255)	Untuk menyimpan nama Device
4	*created_at*	Timestamp	Untuk menyimpan waktu dibuat
5	*update_at*	Timestamp	Untuk menyimpan waktu diupdate

3.5.3 Tabel *Lands*

Tabel *lands* digunakan untuk menyimpan Data lahan yang petani punya, terdiri dari *id, user_id, name, image, description, polygon, area, crop_planted_at, created_at, updated_at, deleted_at*. Adapun tabel *lands* dapat dilihat pada Tabel 3.11.

Tabel 3. 11 Tabel Lands

No	Nama Field	Tipe Data	Keterangan
1	*id*	Bigint(255)	Untuk menyimpan id pengguna
2	*name*	Varchar(255)	Untuk menyimpan nama Lahan
3	*image*	Varchar(255)	Untuk menyimpan URL gambar Lahan
4	*description*	Varchar(255)	Untuk menyimpan deskripsi Lahan
5	*polygon*	Text	Untuk menyimpan data area *polygon* Lahan
6	*area*	Varchar(255)	Untuk menyimpan data luas area Lahan
7	*crop_status*	Varchar(255)	Untuk menyimpan status tanaman pada lahan (sudah panen/ belum panen)
8	*crop_planted_at*	Date	Untuk menyimpan data waktu kapan tanaman ditanam
9	*deleted_at*	Timestamp	Untuk menyimpan waktu dihapus
10	*created_at*	Timestamp	Untuk menyimpan waktu dibuat
11	*update_at*	Timestamp	Untuk menyimpan waktu diupdate

3.4. Pembuatan Perangkat Lunak

Pada tahap ini penulis melakukan pengkodean perangkat lunak, dalam hal ini aplikasi dikembangkan menggunakan bahasa pemrograman PHP dan *framework* Laravel menggunakan arsitektur *Model, View, Controller* (MVC), sesuai dengan arsitektur bawaan yang ada pada *framework* Laravel. Selain itu, dilakukan juga integrasi dengan *API* pada fitur rekomendasi pengairan dan fitur deteksi kualitas padi.

Di bawah ini merupakan daftar *file* kode sumber pada *backend* Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi, hasil implementasi dari desain yang telah dibuat pada tahap sebelumnya.

Tabel 3. 12 Tabel Kode Sumber

Tipe	File Kode Sumber
Model	App\Models\User

Model	App\Models\Land
Model	App\Models\Device
Controller	App\Http\Controllers\AuthController
Controller	App\Http\Controllers\LandController
Controller	App\Http\Controllers\DashboardController
Controller	App\Http\Controllers\UserController
Controller	App\Http\Controllers\LandDeviceController
Controller	App\Http\Controllers\QualityDetectorController
Controller	App\Http\Controllers\Controller
Livewire Component	App\Http\Livewire\Components\LandCard
Helper	App\Helpers\DatatableHelper

3.5. Penerapan *Clean Code*

Pada tahap ini dilakukan *refactoring* pada kode sumber aplikasi dan juga diterapkannya konsep *clean code* pada kode sumber sebelumnya. Adapun konsep *clean code* yang diterapkan penulis yaitu (1) *Meaningful Names,* (2) *Clean Functions,* (3) *Clean Classes,* (4) *Clean Comments*, dan (5) *Clean Code Formatting.*

3.6. Pengukuran Metrik

Pada tahap ini dilakukan *static code analysis* untuk mendapatkan *software metric maintainability index* menggunakan alat PhpMetrics. Kemudian dilakukan perbandingan hasil sebelum dan sesudah penerapan *clean code.*

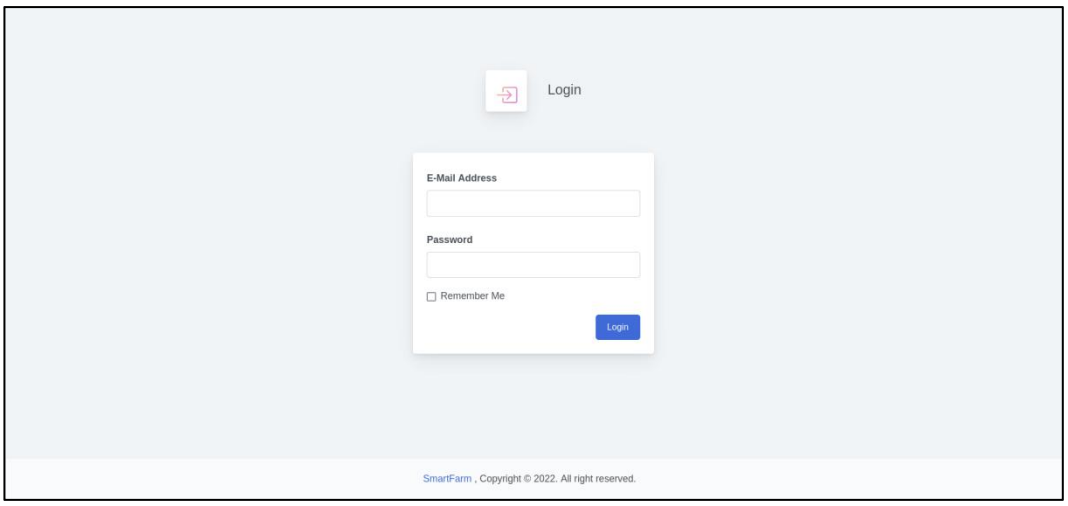
BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil Implementasi Sistem

Pada bagian ini akan membahas hasil implementasi sistem yang telah dirancang dan dibuat, yaitu Aplikasi Manajemen dan Monitoring Lahan Pertanian Padi.

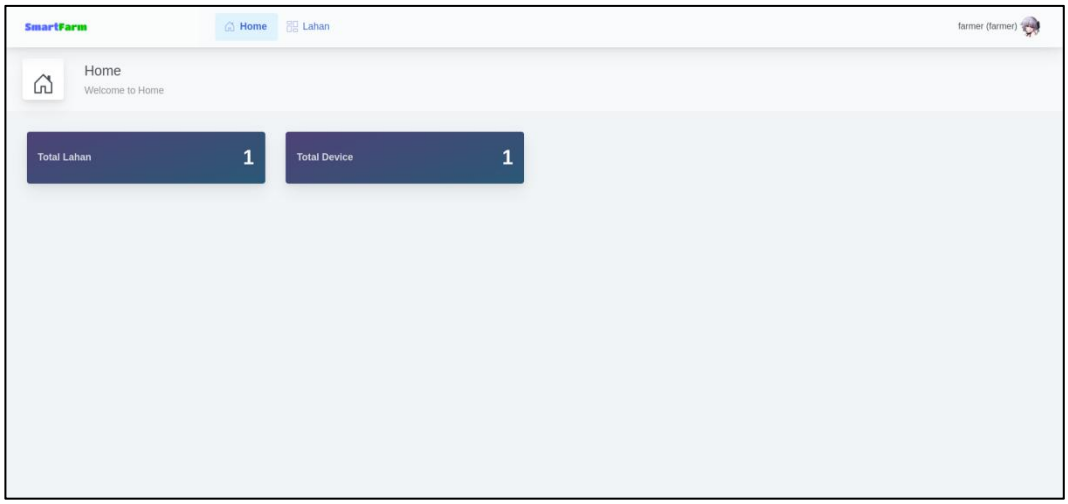
4.1.1. *Web Dashboard*

Pada bagian ini akan dibahas fitur-fitur yang ada pada *Web Dashboard*. Fitur tersebut diantaranya, halaman *Login*, halaman *Dashboard*, halaman kelola Lahan, halaman kelola Perangkat, halaman Deteksi Kualitas Padi, dan halaman Pengaturan. Adapun penjelasan lengkapnya sebagai berikut.



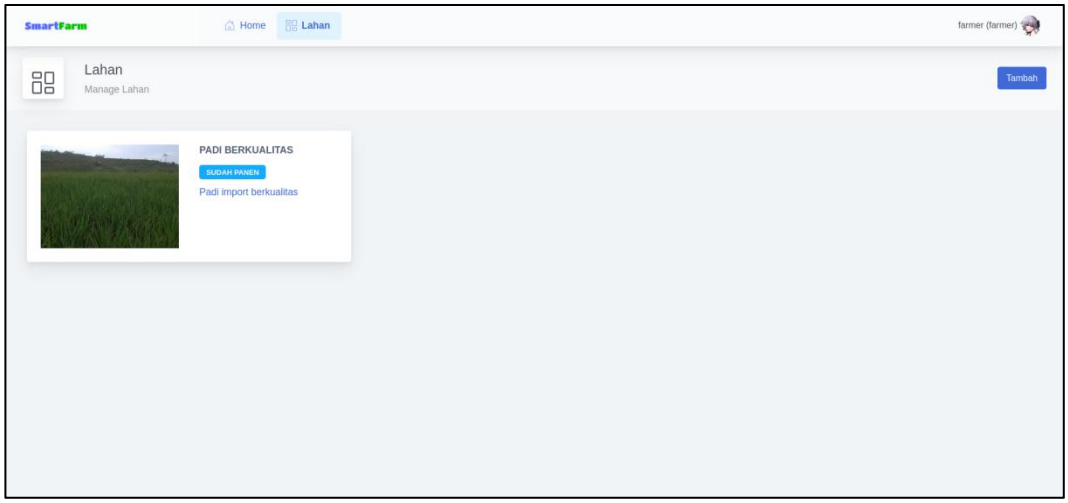
Gambar 4. 1 Halaman *Login*

Pada fitur halaman *login* (Gambar 4.1), admin maupun petani dapat melakukan otorisasi agar dapat masuk ke halaman *dashboard* dan menggunakan fitur lainnya.



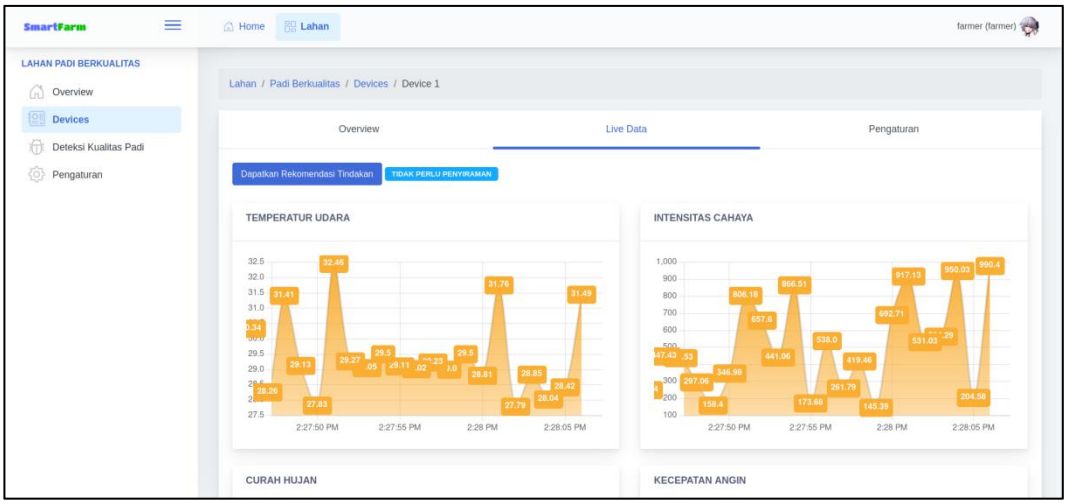
Gambar 4. 2 Halaman *Dashboard* Petani

Pada fitur halaman *dashboard* petani (Gambar 4.2), petani dapat melihat total lahan dan perangkat yang dipunyai.



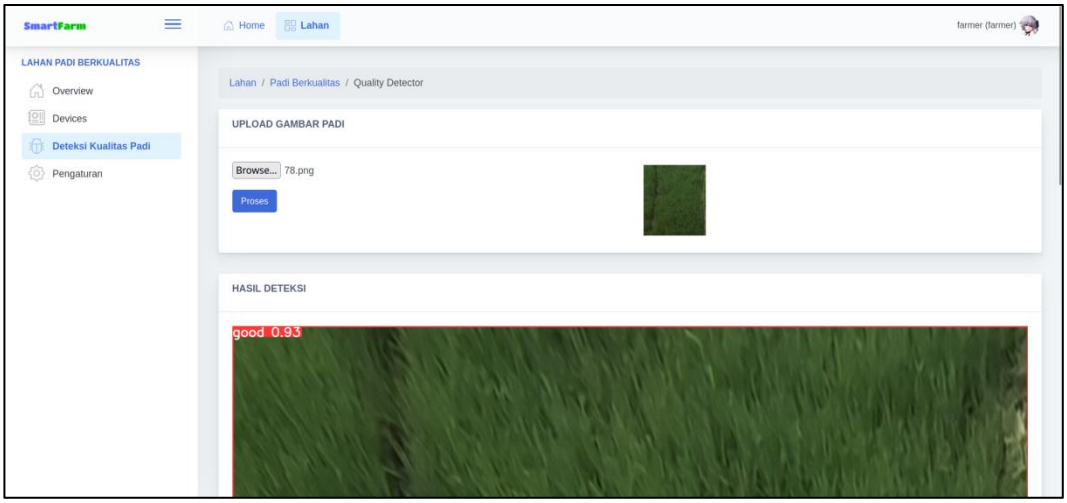
Gambar 4. 3 Halaman Menu Lahan

Pada menu lahan petani (Gambar 4.3), petani dapat melihat daftar Lahan yang telah ditambahkan.



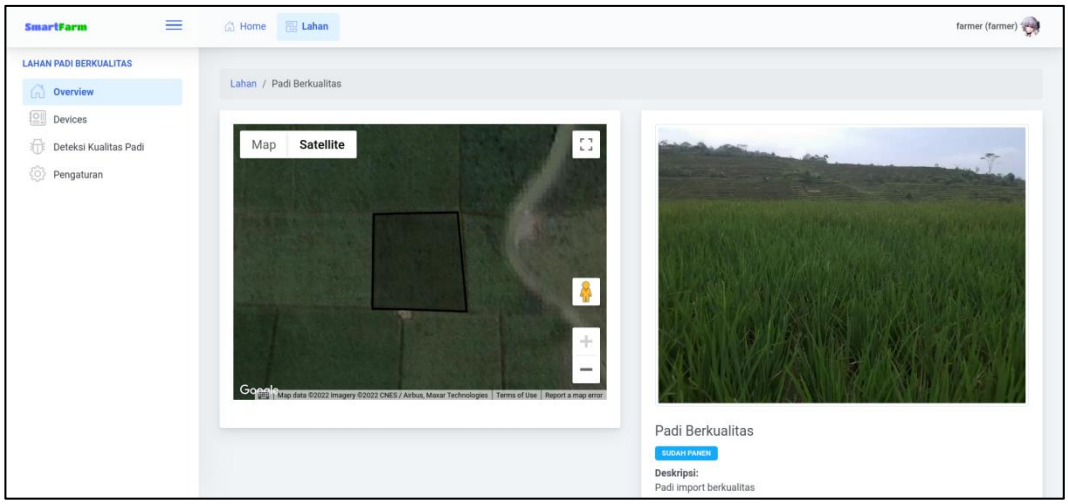
Gambar 4. 4 Halaman *Live Data*

Pada halaman *live data* (Gambar 4.4), petani dapat melihat data sensor yang terpasang pada perangkat secara *realtime* dengan grafik.



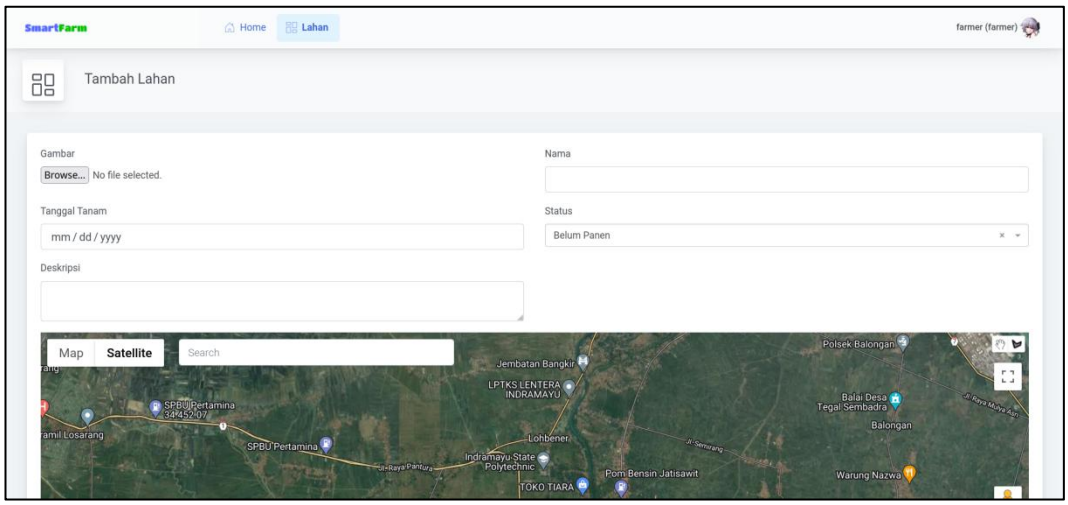
Gambar 4. 5 Deteksi Kualitas Padi

Pada halaman menu deteksi kualitas padi (Gambar 4.5), petani dapat mengunggah gambar lahan padi dan mendapatkan hasil deteksi kualitasnya.



Gambar 4. 6 *Menu Overview*

Pada halaman *overview* (Gambar 4.6), petani dapat melihat detail lahan dan lokasi lahan pada *maps*.



Gambar 4. 7 Halaman Tambah Lahan

Pada halaman tambah lahan (Gambar 4.7), petani dapat menginputkan detail lahan yang ingin ditambahkan.

4.2. Hasil Penerapan *Clean Code*

Pada bagian ini dijelaskan hasil implementasi konsep *clean code* yang telah diterapkan pada kode sumber aplikasi.

4.1.2.1. *Meaningful Names*

Pada bagian ini, peneliti menerapkan konsep *meaningful names* dengan memperhatikan penamaan nama *class, method,* maupun variabel. Contoh hasil dari penerapan konsep *meaningful names* dapat dilihat pada gambar berikut.

```php
public function login(LoginRequest $request)
{
    $validatedRequest = $request->validated();
    try {
        Auth::attempt($validatedRequest);
        return $this->handleRedirectByRole();
    } catch (\Exception $e) {
        return redirect()->back()->withErrors(['error' => $e->getMessage()]);
    }
}
```

Gambar 4. 8 *Meaningful Names* Pada Fungsi *Login*

Pada fungsi *login* (Gambar 4.8) nama fungsi tersebut *"login"* telah memiliki arti, kemudian pada variabel `$validatedRequest` nama tersebut telah memiliki arti.

4.1.2.2. *Clean Functions*

Pada bagian ini, peneliti menerapkan konsep *clean functions* dengan memperhatikan jumlah parameter dan jumlah pekerjaan pada suatu *method*/ fungsi. Contoh hasil dari penerapan konsep *clean functions juga* dapat dilihat pada Gambar 4.8, fungsi *login* tersebut memiliki keterkaitan dengan konsep *clean function* berikut

1. Jumlah Parameter, fungsi *login* tersebut hanya memiliki satu parameter saja.
2. Jumlah Pekerjaan, fungsi *login* tersebut telah memisahkan logika untuk memvalidasi *request* dan memisahkan logika untuk mengatasi *redirect* halaman berdasarkan *role* user.

4.1.2.3. *Clean Classes*

Pada bagian ini, peneliti menerapkan konsep *clean classess* dengan memperhatikan *class organization* dan *responsibility* pada suatu *class*. Contoh hasil dari penerapan konsep *clean classess* dapat dilihat pada gambar berikut.

```php
class LoginController extends Controller
{
    /*
    * Show the login form
    */
    public function index()
    {…
    }

    /*
    * Handle the redirect after login by user role
    */
    private function handleRedirectByRole()
    {…
    }

    /*
    * Login
    */
    public function login(LoginRequest $request)
    {…
    }
}
```

Gambar 4. 9 *Clean Classes* Pada Login Controller

Pada *class LoginController* (Gambar 4.9) class tersebut sudah dikatakan tepat karena hanya melayani hal yang berhubungan dengan *login.*

4.1.2.4. *Clean Comments*

Pada bagian ini, peneliti menerapkan konsep *clean comments* dengan memberikan komentar pada hal-yang yang diperlukan saja, seperti sintaks yang rumit.

4.1.2.5. *Clean Code Formatting*

Pada bagian ini, peneliti menerapkan konsep *clean code formatting* dengan menerapkan *coding standard* PSR-12. Berikut merupakan hasil pengukuran PHP Codesniffer terhadap kode sumber untuk coding standard PSR-12 dengan menjalankan perintah pada *terminal* "`./vendor/bin/phpcs --standard=PSR12 --report=summary app/`".

```
PHP CODE SNIFFER REPORT SUMMARY
----------------------------------------------------------------------------------------------------
FILE                                                                                  ERRORS  WARNINGS
----------------------------------------------------------------------------------------------------
/home/abu/repository/smart-farm/app/Console/Kernel.php                                2       0
/home/abu/repository/smart-farm/app/Helpers/DatatableHelper.php                       2       0
/home/abu/repository/smart-farm/app/Http/Controllers/AuthController.php               4       0
/home/abu/repository/smart-farm/app/Http/Controllers/Controller.php                   1       0
/home/abu/repository/smart-farm/app/Http/Controllers/DashboardController.php          1       0
/home/abu/repository/smart-farm/app/Http/Controllers/LandController.php               1       1
/home/abu/repository/smart-farm/app/Http/Controllers/LandDeviceController.php         1       1
/home/abu/repository/smart-farm/app/Http/Controllers/UserController.php               1       0
/home/abu/repository/smart-farm/app/Models/Land.php                                   2       1
/home/abu/repository/smart-farm/app/Models/User.php                                   2       0
/home/abu/repository/smart-farm/app/Providers/AppServiceProvider.php                  0       1
/home/abu/repository/smart-farm/app/View/Components/device/live-data.php              4       0
----------------------------------------------------------------------------------------------------
A TOTAL OF 21 ERRORS AND 4 WARNINGS WERE FOUND IN 12 FILES
----------------------------------------------------------------------------------------------------
```

Gambar 4. 10 PHPCS Sebelum Refactoring

Pada Gambar 4.10 merupakan hasil analisa PHPCS pada kode sumber sebelum refactoring, terdapat 21 errors dan 4 warning yang menandakan masih bisa dilakukan perbaikan.

```
PHP CODE SNIFFER REPORT SUMMARY
----------------------------------------------------------------------------------------------------
FILE                                                                                  ERRORS  WARNINGS
----------------------------------------------------------------------------------------------------
/home/abu/repository/smart-farm-clean/app/Http/Controllers/LandController.php         0       1
/home/abu/repository/smart-farm-clean/app/Http/Middleware/RedirectIfAuthenticated.php 0       1
/home/abu/repository/smart-farm-clean/app/Models/Land.php                             0       1
/home/abu/repository/smart-farm-clean/app/Providers/AppServiceProvider.php            0       1
----------------------------------------------------------------------------------------------------
A TOTAL OF 0 ERRORS AND 4 WARNINGS WERE FOUND IN 4 FILES
----------------------------------------------------------------------------------------------------
```

Gambar 4. 11 PHPCS Setelah Refactoring

Pada Gambar 4.11 Setelah dilakukan perbaikan kode sumber dengan menerapkan coding standard PSR-12, kode sumber memiliki lebih sedikit errors dan warning.

4.3. Hasil Perhitungan *Software Metric*

Hasil pengukuran *Maintainability Index* sebelum penerapan *refactoring* dapat dilihat pada Tabel 4.1, dan Tabel 4.2 setelah *refactoring* dan penerapan *clean code*

Tabel 4. 1 Nilai *Maintainability Index* Sebelum Refactoring

File Kode Sumber	Nilai MI
App\Models\User	118.88
App\Models\Device	82.13
App\Models\Land	54.08
App\Http\Controllers\AuthController	47.36
App\Http\Controllers\LandController	50.66

App\Http\Controllers\DashboardController	63.72
App\Http\Controllers\UserController	64.23
App\Http\Controllers\LandDeviceController	67.46
App\Http\Controllers\QualityDetectorController	74.55
App\Http\Controllers\Controller	171
App\Http\Livewire\Components\LandCard	67.56
App\Helpers\DatatableHelper	66.78
Rata-Rata	**77.37**

Tabel 4. 2 Nilai *Maintainability Index* Sebelum Refactoring

File Kode Sumber	**Nilai MI**
App\Models\User	118.88
App\Models\Device	82.13
App\Models\Land	89.6
App\Http\Controllers\LandController	74.66
App\Http\Controllers\DashboardController	62.63
App\Http\Controllers\UserController	64.23
App\Http\Controllers\LandDeviceController	72.48
App\Http\Controllers\QualityDetectorController	74.55
App\Http\Controllers\Auth\RegisterController	95.79
App\Http\Controllers\Auth\LogoutController	76.94
App\Http\Controllers\Auth\LoginController	79.45
App\Http\Controllers\Controller	171
App\Http\Livewire\Components\LandCard	67.56
App\Http\Requests\RegisterRequest	110.35
App\Http\Requests\Auth\RegisterRequest	111.55
App\Http\Requests\Auth\LoginRequest	111.55
App\Helpers\DatatableHelper	66.78
Rata-Rata	**86.53**

BAB V
PENUTUP

5.1. Kesimpulan

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, dapat diperoleh kesimpulan bahwa penerapan *clean code* dapat menghasilkan nilai *maintainability index* sebesar 86.53 pada *backend* Aplikasi Manajemen Dan Monitoring Lahan Pertanian Padi, sehingga dapat dikatakan bahwa kode sumber tersebut tergolong dapat di*maintenance* dengan baik.

5.2. Saran

Saat mengembangkan perangkat lunak, salah satu bentuk penggunaan konsep *clean code* adalah mengikuti konvensi pengkodean bahasa pemrograman maupun *framework* yang digunakan. Pada dasarnya *clean code* hanyalah panduan umum penulisan source code sehingga penggunaan *clean code* dapat disesuaikan dengan kebutuhan *developer*. Adapun saran terhadap penelitian ini ke depannya dapat sebagai berikut:

1. Penilaian *clean code* dilakukan oleh *reviewer*.
2. Penerapan *clean code* dilakukan menyeluruh, termasuk pada bagian *frontend* aplikasi.

DAFTAR PUSTAKA

Annur, C. M. (2020). *Sektor Pertanian paling Banyak Menyerap Tenaga Kerja Indonesia: Databoks*. Databoks Pusat Data Ekonomi dan Bisnis Indonesia. Retrieved August 9, 2022, from https://databoks.katadata.co.id/datapublish/2020/11/12/sektor-pertanian-paling-banyak-menyerap-tenaga-kerja-indonesia

Dietz, L. W., Manner, J., Harrer, S., & Lenhard, J. (2018). Teaching Clean Code. *Proceedings of the 1st Workshop on Innovative Software Engineering Education*, 24-27.

Jackson, M. (1984). ISO/IEC.. Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)-System and softwre quality models. *Switz. ISOIEC Towar. a Syst. Syst. Methodol. J. Oper. Res. Soc*, 473-486.

Faisol, A., Orisa, M., Ibrahim Ashari, M., & Putu Agustini, N. (2021). Pengaruh Pengujian Statis Terhadap Kualitas Perangkat Lunak Sitagih Pada pt. Semesta Mitra sejahtera (SMS). *Jurnal Mnemonic*, *4*(2), 45–49. https://doi.org/10.36040/mnemonic.v4i2.4114

Rizaldhi, D. A. (2019). *Implementasi Clean Code Dan Design Pattern Untuk Meningkatkan Maintainability Pada Aplikasi Content Marketing.*

Prasetyo, I. (2021). *Implementasi Clean Code Dan Code Refactoring Untuk Meningkatkan Maintainability Pada Luarsekolah Bot (Lusa Bot).*

Galin, D. (2004). *Software quality assurance from theory to implementation*. Pearson.

Stapelberg, R. F. (2009). *Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design.* London: Springer.

Arhandi, P. P., Pramitarini, Y., & Alviandra, R. (2019). Desain Prototype Frontend Auto Generator Based On REST API. *Seminar Informatika Aplikatif Polinema*, 389-393.

Martin, R. C. (2008). *Clean code: A Handbook of Agile Software Craftsmanship*. Pearson Education.

Fowler, M., & Beck, K. (2002). *Refactoring improving the design of existing code*. Addison-Wesley.

Prasetyo Adi Helmi, Yudhanto Yudho. 2018. *Panduan Mudah Belajar Framework Laravel*. PT. Elex Media Komputindo, Jakarta.

Widodo, P. P., & Prabowo, H. (2011). Menggunakan uml. *Bandung: Informatika*, *19*, 393-403.

Pooley, R., & Wilcox, P. (2003). *Applying UML: Advanced Applications*. Butterworth-Heinemann.

Mulyani, Sri. 2016. *Sistem Informasi Manajemen*. Bandung: Abdi Sistematika.

Haviluddin. (2011). Memahami Penggunaan UML (Unified Modelling Language). *Memahami Penggunaan UML (Unified Modelling Language)*, *6*(1), 1–15.

Chen, C., Alfayez, R., Srisopha, K., Boehm, B., & Shi, L. (2017). Why Is It Important to Measure Maintainability and What Are the Best Ways to Do It? *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*.

LAMPIRAN

Lampiran 1 Biodata Penulis

Nama Mahasiswa	: Abu Mushonnip
Tempat Tanggal Lahir	: Indramayu, 16 Agustus 1999
Jenis Kelamin	: Laki–Laki
Golongan Darah	: -
No. Induk Mahasiswa	: 1805001
Alamat	: Dusun Lempuyang, RT 05 RW 01, Kec. Anjatan, Kab. Indramayu
No Handphone	: +62 812 8625 3315
Email	: mushonnip@protonmail.com
Nama Orang Tua	: Junaedi
Pekerjaan	: Wiraswasta
Alamat	: Dusun Lempuyang, RT 05 RW 01, Kec. Anjatan, Kab. Indramayu

Lampiran 2 Kode