**REVISI VERSION 0.6**

# SAMPUL DEPAN

# LEMBAR JUDUL

# LEMBAR PENGESAHAN

# LEMBAR PERSETUJUAN DEWAN PENGUJI

# LEMBAR PERNYATAAN KEASLIAN

# ABSTRAK

# ABSTRACT

# KATA PENGANTAR

# DAFTAR ISI

[SAMPUL DEPAN 2](#_Toc166085975)

[LEMBAR JUDUL 3](#_Toc166085976)

[LEMBAR PENGESAHAN 4](#_Toc166085977)

[LEMBAR PERSETUJUAN DEWAN PENGUJI 5](#_Toc166085978)

[LEMBAR PERNYATAAN KEASLIAN 6](#_Toc166085979)

[ABSTRAK 7](#_Toc166085980)

[ABSTRACT 8](#_Toc166085981)

[KATA PENGANTAR 9](#_Toc166085982)

[DAFTAR ISI 10](#_Toc166085983)

[DAFTAR TABEL 12](#_Toc166085984)

[DAFTAR GAMBAR 13](#_Toc166085985)

[DAFTAR LAMPIRAN 14](#_Toc166085986)

[DAFTAR SINGKATAN DAN LAMBANG 15](#_Toc166085987)

[BAB I PENDAHULUAN 1](#_Toc166085988)

[1.1 Latar Belakang 1](#_Toc166085989)

[1.2 Rumusan Masalah 2](#_Toc166085990)

[1.3 Batasan Masalah 3](#_Toc166085991)

[1.4 Tujuan Penelitian 3](#_Toc166085992)

[1.5 Metode Penelitian 3](#_Toc166085993)

[1.5.1 Metode Pengumpulan Data 3](#_Toc166085994)

[1.5.2 Metode Pengembangan Sistem 4](#_Toc166085995)

[1.6 Sistematika Penulisan 7](#_Toc166085996)

[BAB II LANDASAN TEORI 8](#_Toc166085997)

[BAB III ANALISIS DAN PERANCANGAN 11](#_Toc166085998)

[3.1 Analisis Sistem Berjalan 11](#_Toc166085999)

[3.2 Analisis Kebutuhan Software/Sistem 13](#_Toc166086000)

[3.2.2 Spesifikasi Kebutuhan Fungsional Software 15](#_Toc166086001)

[3.2.3 Diagram Use Case 17](#_Toc166086002)

[3.2.4 Skenario Use Case 17](#_Toc166086003)

[3.2.5 *Activity* Diagram 20](#_Toc166086004)

[3.2.6 Diagram Komponen Software 22](#_Toc166086005)

[3.2.7 Diagram Sequence 23](#_Toc166086006)

[BAB IV IMPLEMENTASI DAN PENGUJIAN 26](#_Toc166086007)

[BAB V KESIMPULAN DAN SARAN 28](#_Toc166086008)

# DAFTAR TABEL

[Tabel 3. 1 Kebutuhan Pengguna 16](#_Toc165166394)

# DAFTAR GAMBAR

[Gambar 1. 1 Metode Waterfall 5](#_Toc164681680)

[Gambar 3. 1 Flowmap analisis berjalan 11](#_Toc166085923)

[Gambar 3. 2 Flowmap analisis yang diusulkan 13](#_Toc166085924)

[Gambar 3. 3 Diagram use case penilaian kode sumber 17](#_Toc166085925)

[Gambar 3. 4 Activity diagram input spesifikasi kode sumber 20](#_Toc166085926)

[Gambar 3. 5 Activity diagram input kode sumber 20](#_Toc166085927)

[Gambar 3. 6 Activity diagram proses penilaian kode sumber 21](#_Toc166085928)

[Gambar 3. 7 Activity diagram penyimpanan arsip penilaian 21](#_Toc166085929)

[Gambar 3. 8 Activity diagram pengelolaan data arsip dengan API 22](#_Toc166085930)

[Gambar 3. 9 Activity diagram komponen software 22](#_Toc166085931)

[Gambar 3. 10 Sequence diagram perancangan kode sumber 23](#_Toc166085932)

# DAFTAR LAMPIRAN

# DAFTAR SINGKATAN DAN LAMBANG

# PENDAHULUAN

## Latar Belakang

*Programmer* adalah orang yang bertugas untuk mengembangkan perangkat lunak atau aplikasi. Ketika membuat suatu program, ada bagan alur yang harus diperhatikan kemudian diterjemahkan ke dalam bahasa komputer menggunakan kode tertentu sehingga menjadi sebuah perintah kepada komputer untuk menjalankan tugas tertentu [1].

Kode Sumber adalah komponen dasar dari program komputer yang dibuat oleh seorang *programmer*, yang sering kali ditulis dalam bentuk fungsi, deskripsi, definisi, pemanggilan, metode, dan pernyataan operasional lainnya. Kode sumber ini dirancang agar dapat dibaca manusia dan diformat dengan cara yang dapat dimengerti oleh pengembang dan pengguna lain[2].

Penguji (Tinjauan Kode Sumber) adalah peran yang melakukan pemeriksaan kode sumber aplikasi untuk menemukan kesalahan yang terlewatkan pada tahap pengembangan awal. Penguji menganalisis kode sumber baris demi baris dengan memeriksanya secara manual untuk menemukan kejenuhan dan menghilangkan positif palsu. Positif palsu adalah saat penguji menerima hasil positif untuk suatu tes, padahal programmer seharusnya menerima hasil negatif. Kadang-kadang disebut sebagai “alarm palsu” atau “kesalahan positif palsu”[4]. Jumlah waktu yang dihabiskan penguji untuk meninjau kode sumber bervariasi menurut bahasa pemrograman dan ukuran aplikasi. Misalnya, 1000 baris kode mungkin memerlukan waktu 0,5 – 2 jam untuk dianalisis[3].

Dalam mempelajari sebuah pemrograman *programmer* tentunya harus sering belajar yang harus melibatkan penguji untuk menilai kode yang dibuatnya. Rata-rata penguji sangat kerepotan karena harus menguji beberapa kode sumber dari *programmer* sehingga membutuhkan waktu yang cukup lama dan sangat rentan dengan human *error*. Sebagian dari penguji masih ada yang menilai Kode Sumber secara manual satu per satu, melihat *output* yang sesuai, bahkan jika dijalankan tidak ada *error* maka dinilai berhasil. Kekurangan dalam proses penilaian ini adalah waktu yang diperlukan yang cukup lama karena setiap tugas dari programmer harus diuji satu per satu. Selain itu, terdapat risiko bahwa penguji mungkin kurang fokus saat melakukan pengujian, yang dapat mengakibatkan keterlewatannya kesalahan atau masalah yang penting dalam kode yang diuji.

Sehingga, penguji membutuhkan sistem yang dapat memeriksa dan memperbaiki kode sumber sesuai dengan standar umum yang digunakan oleh programmer. Selain itu, mereka juga membutuhkan fitur penilaian otomatis untuk mengevaluasi kualitas kode sumber secara keseluruhan, yang dapat mempercepat proses penilaian.

Cara kerja sistem yang diusulkan diharapkan penguji membuat terlebih dahulu kode sumber dan TDD (*Test Driven Development*) yang nantinya digunakan untuk referensi jawaban yang tepat, selanjutnya kode sumber akan diproses terlebih dahulu yang akan dicek oleh *library* yang mendukung kode sumber seperti Eslint, Prettier. Jika tidak sesuai prosedur maka akan masuk ke *library* yang dibuat pada penelitian ini agar dimasukan ke sistem yang nanti akan digunakan untuk referensi jawaban dari programmer. Selanjutnya dari sisi programmer melakukan input kode sumber yang akan diproses oleh *library* pada penelitian ini yang nantinya akan dianalisis menggunakan beberapa *library* pendukung dari NPM (*Node Package Manager*) serta bantuan metode *BlackBox* dan sesuai prosedur *test* yang diberikan penguji menggunakan metode TDD (*Test Driven Development*).

## Rumusan Masalah

Adapun rumusan masalah dari penelitian ini, antara lain :

1. Bagaimana perancangan dan pembuatan *library* penilaian kode sumber otomatis?
2. Bagaimana cara praktis menerapkan *library* penilaian otomatis untuk menilai kode sumber JavaScript?
3. Bagaimana cara untuk meningkatkan transparansi dan objektivitas dalam penilaian kode sumber?

## Batasan Masalah

Pada penelitian ini adapun batasan masalahnya, antara lain :

1. Pengujian ini akan dilakukan hanya dengan bantuan *library* yang menggunakan bahasa pemrograman Javascript.
2. Pengujian berfokus pada mengecek kode sumber bahasa pemrograman Javascript.
3. Pembuatan *library* akan didukung dengan metode *BlackBox* dan *Test Driven Development*.

## Tujuan Penelitian

Tujuan dari penelitian ini adalah :

1. Meningkatkan efisiensi dan konsistensi penilaian kode sumber.
2. Memudahkan penguji dan programmer dalam evaluasi kualitas kode sumber.
3. Meningkatkan transparansi dan objektivitas dalam penilaian kode sumber.

## Metode Penelitian

### Metode Pengumpulan Data

Sumber Data

1. Data Primer: Merupakan informasi yang diperoleh dari sumber primer, seperti dari orang atau individu melalui wawancara, survei, dan kuesioner wawancara, yang dilakukan oleh peneliti[5].
2. Data Sekunder: Merupakan informasi yang sudah diolah sebelumnya dan diperoleh oleh peneliti dari berbagai sumber lain untuk memperoleh tambahan informasi. Beberapa sumber data sekunder juga bisa berupa literatur, jurnal, serta publikasi pemerintah dan sumber daya online lainnya[5].

Observasi

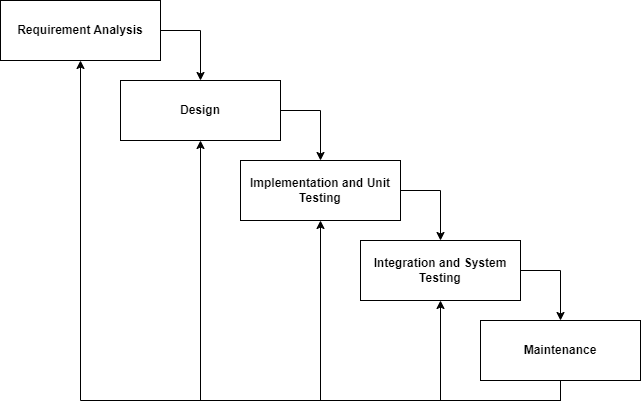
Peneliti melakukan pengamatan secara langsung yang berkaitan dengan masalah yang diteliti untuk dianalisa.

Studi Pustaka

Studi pustaka dikenal menjadi tinjauan pustaka, merupakan proses pengumpulan, analisis, dan buatan literatur yang relevan menggunakan topik atau kasus penelitian tertentu. Ini melibatkan membaca aneka macam asal fakta misalnya jurnal ilmiah, buku, makalah konferensi, artikel daring, dan asal-asal lainnya yang terkait menggunakan subjek yang sedang diteliti[5].

### Metode Pengembangan Sistem

Metode pengembangan sistem yang peneliti gunakan dalam penulisan skripsi ini adalah Metode *Waterfall*. Model *waterfall* adalah model pengembangan perangkat lunak yang terdiri dari analisa kebutuhan perangkat lunak, desain, pembuatan kode sumber, pengujian, pendukung dan pemeliharaan sistem. Model *waterfall* dikerjakan secara berurutan sesuai dengan tahapan awal sampai akhir, sehingga proyek yang dikerjakan memiliki perencanaan yang lebih cermat, dokumentasi lebih rinci dan eksekusi yang berurutan[6].



Gambar 1. 1 Metode Waterfall

#### Requirement Analysis

Tahap awal pengembangan sistem melibatkan pengumpulan dan analisis kebutuhan sistem, termasuk aspek fungsional dan non-fungsional, serta pemahaman yang mendalam tentang kebutuhan pengguna dan tujuan sistem. Setelah itu, desain arsitektur sistem dilakukan dengan menggunakan metode UML (*Unified Modelling Language*). UML adalah suatu metode dalam pemodelan secara tampilan visual yang dipergunakan untuk perancangan sistem yang berorientasi dengan objek.

#### Design

Setelah kebutuhan sistem dipahami dengan baik, langkah berikutnya adalah merancang arsitektur sistem secara keseluruhan dan detail implementasi. Ini mencakup merancang struktur data, algoritma, antarmuka pengguna, dan komponen sistem lainnya. Desain ini harus memastikan bahwa sistem yang akan dibangun akan mencapai tujuan yang telah ditetapkan dan memenuhi kebutuhan pengguna.

#### Implementation and Unit Testing

Pada tahap implementasi, pengembang mulai menerjemahkan desain sistem menjadi kode program menggunakan bahasa JavaScript. Proses ini melibatkan penulisan kode, pembangunan modul atau komponen, dan integrasi secara bertahap. Selama pengembangan, penggunaan library npm memfasilitasi manajemen paket dan dependensi. Alat pengembangan WebStorm digunakan sebagai lingkungan pengembangan terintegrasi (IDE), menyediakan fitur-fitur seperti debugging dan auto-completion untuk mempercepat proses pembangunan. Selain itu, unit tes dilakukan secara terus menerus untuk memastikan setiap komponen atau modul berfungsi sesuai yang diharapkan dan memastikan kualitas keseluruhan dari kode yang dihasilkan.

#### Integration and System Testing

Setelah semua komponen atau modul sistem telah diimplementasikan dan diuji secara individual, tahap selanjutnya adalah mengintegrasikan mereka menjadi satu sistem utuh. Proses ini melibatkan menguji interaksi antara komponen-komponen tersebut dan memastikan bahwa sistem berfungsi dengan baik sebagai satu kesatuan. Pengujian sistem dilakukan untuk memverifikasi apakah sistem memenuhi spesifikasi yang telah ditetapkan dan berperilaku sesuai dengan harapan.

#### Maintenance

Tahap ini terjadi setelah sistem telah dirilis dan digunakan oleh pengguna. Pada tahap ini, perawatan sistem dilakukan untuk memastikan bahwa sistem tetap berjalan dengan baik dan memenuhi kebutuhan pengguna. Ini melibatkan pemecahan bug, peningkatan fitur, dan perbaikan yang mungkin diperlukan seiring waktu.

## Sistematika Penulisan

Sistematika penyusunan laporan tugas akhir ini adalah sebagai berikut:

**BAB I Pendahuluan**

Dalam bab ini akan diuraikan mengenai latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, metodologi penelitian, manfaat penelitian dan sistematika penulisan.

**BAB II Landasan Teori**

Dalam bab ini menjelaskan dari beberapa teori yang digunakan untuk melakukan penelitian dan penulisan laporan (teori apasaja).

**BAB III Analisis dan Perancangan**

Dalam bab ini menjelaskan analisis dan rancangan menggunakan UML sehingga dihasilkan diagram use case, diagram activity, class diagram, squensial diagram, dan Desain Database.

**BAB IV Implementasi dan Pengujian**

Dalam bab ini menjelaskan metode dan pelaksanaan pengujian mencakup kasus uji, prosedur uji, dan hasil uji mengacu pada kebutuan fungsional dan perancangan perangkat lunak.

**BAB V Kesimpulan dan Saran**

Bab ini berisi kesimpulan dari hasil pengujian dan saran-saran untuk perbaikan perangkat lunak dipenelitian ini.

# LANDASAN TEORI

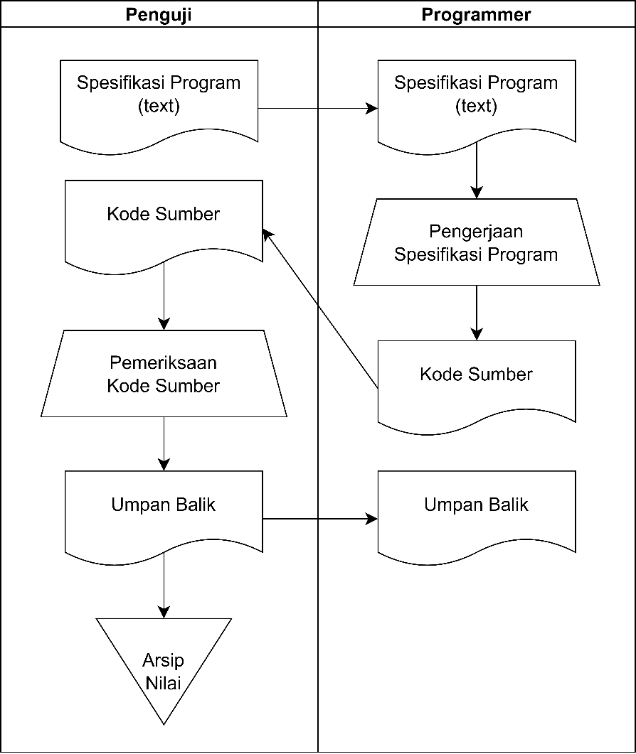
# ANALISIS DAN PERANCANGAN

## Analisis Sistem Berjalan

Aktifitas pengujian *software* bisa terjadi antara programmer dan penguji *software* pada suatu proyek atau antara mahasiswa dan dosen pada mata kuliah pemrograman. Pengujian *software* dalam hal ini adalah pemeriksaan kode sumber untuk menemukan positif palsu. Aktifitas pengujian yang umum dilakukan adalah:

1. Penguji memberikan soal pemrograman dalam bentuk tulisan yang deskriptif kepada programmer untuk melakukan pengerjaan tugas.
2. Programmer membuat kode sumber yang sesuai prosedur yang telah diberikan oleh penguji dan hasilnya nanti akan dikirimkan ke penguji.
3. Penguji melakukan pemeriksaan secara manual dan satu per satu untuk melakukan penilai Kode Sumber dari programmer.

**Flowmap Sistem Berjalan**



Gambar 3. 1 Flowmap analisis berjalan

Berikut proses bisnis yang sedang berjalan dalam pengujian Kode Sumber:

Penerimaan Spesifikasi Program, Penguji memberikan tugas kepada programmer dalam bentuk deskripsi secara rinci. Setiap programmer hanya boleh menerima/memiliki satu spesifikasi program.

Pengerjaan Spesifikasi Program, Programmer membuat kode sumber yang sesuai dengan spesifikasi yang diberikan. Jawaban dari programmer bisa berupa teks atau kode sumber.

Pengiriman Jawaban, Programmer mengirimkan jawaban mereka kepada penguji. Setiap programmer hanya dapat mengirimkan satu jawaban.

Pemeriksaan Kode Sumber, Penguji melakukan pemeriksaan manual terhadap jawaban yang diterima. Penilaian dilakukan terhadap kebenaran, ketepatan (untuk teks) atau kualitas, efektivitas, dan efisiensi (untuk kode) secara manual.

Umpan Balik, Penguji memberikan umpan balik kepada programmer setelah pemeriksaan selesai. Umpan balik dapat berupa evaluasi kinerja, saran perbaikan, atau apresiasi atas hasil kerja.

Arsip Nilai, Penguji akan menyimpan seluruh penilaian kode sumber ke arsip yang sudah disediakan. Programmer tidak dapat mengakses arsip tersebut untuk mencegah manipulasi atau modifikasi yang tidak sah.

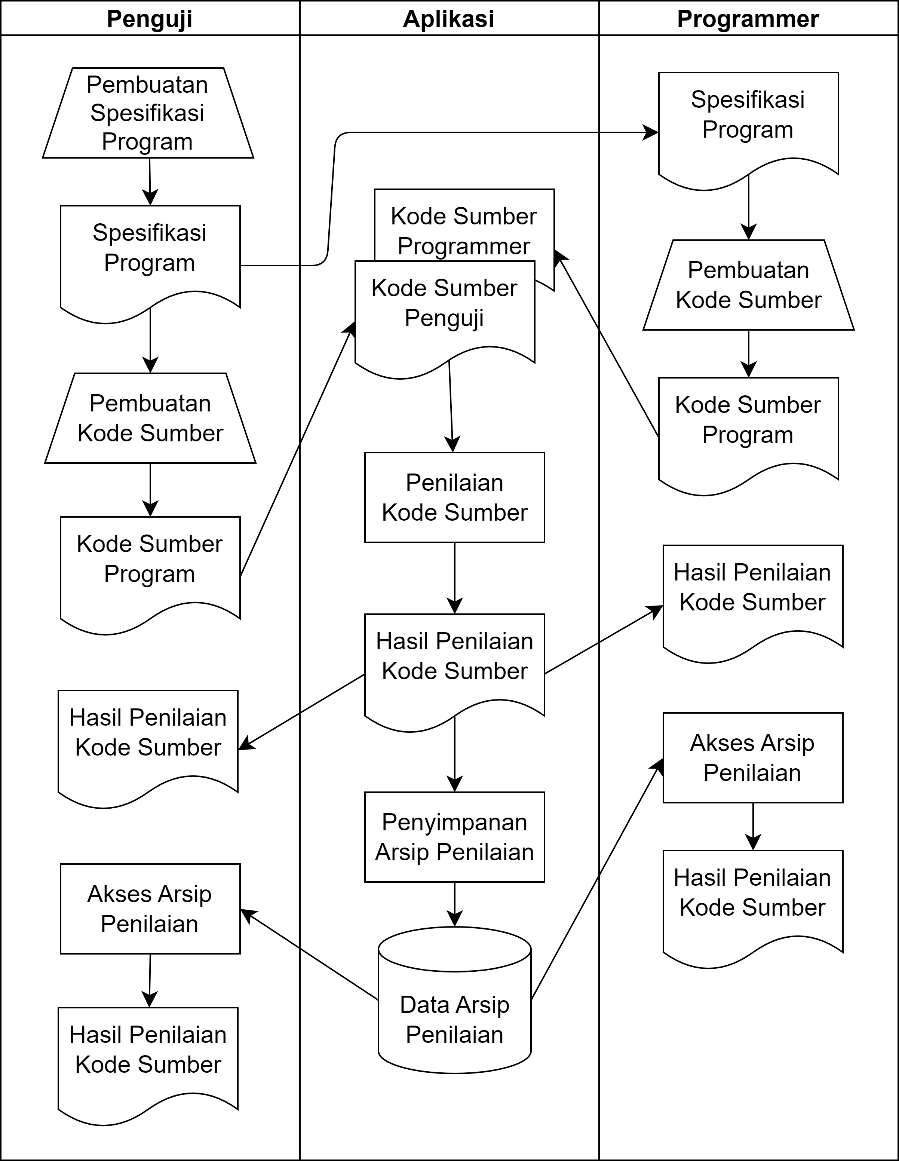
Kelemahan pada sistem yang sedang berjalan diantaranya sebagai berikut:

1. Proses manual memperlambat umpan balik kepada programmer, menghambat kemajuan proyek.
2. Risiko kesalahan manusia dalam penilaian kode sumber karena faktor kejenuhan.
3. Ketergantungan pada penilaian manual, kurangnya integrasi dengan alat pengembangan modern.

## Analisis Kebutuhan Software/Sistem

Untuk memperbaiki kelemahan sistem pengujian pada sistem yang sedang berjalan diusulkan pembuatan *library* yang dapat digunakan pengecekan kode sumber secara otomatis. Proses bisnis pengecekan kode sumber secara otomatis menggunakan *library* yang akan digunakan sebagai berikut:

**Flowmap Sistem Yang Diusulkan**



Gambar 3. 2 Flowmap analisis yang diusulkan

Berikut proses bisnis yang sedang berjalan dalam pengujian Kode Sumber:

Penguji

**Pembuatan Spesifikasi Program**: Penguji bertanggung jawab untuk merumuskan spesifikasi program yang jelas dan sesuai dengan kebutuhan proyek.

**Spesifikasi Program**: Penguji telah membuat spesifikasi program yang akan dikirimkan ke programmer sebagai soal.

**Pembuatan Kode Sumber**: Penguji akan membuatkan kode sumber yang nantinya akan menjadi acuan penilaian untuk kode sumber programmer.

**Kode Sumber Program**: Kode Sumber yang telah dibuat penguji akan dimasukan kedalam aplikasi dan akan diproses untuk penilaian kode sumber secara otomatis.

**Hasil Penilaian Kode Sumber**: Setelah melewati penilaian kode sumber maka akan memberikan umpan balik ke penguji setelah itu akan disimpan didatabase untuk mengarsip data penilaiannya.

**Akses Arsip Penilaian**: Ketika data penilaian kode sumber sudah tersimpan secara komputerisasi. Maka penguji dan programmer bisa mengakses datanya dengan API dengan hak akses yang berbeda.

Programmer

**Spesifikasi Program**: Programmer mendapatkan spesifikasi program dari penguji yang harus dipahami dengan baik agar sesuai dengan harapan penguji.

**Pembuatan Kode Sumber**: Programmer membuat kode sumber sesuai dengan spesifikasi yang telah ditetapkan oleh penguji.

**Kode Sumber Program**: Kode Sumber yang telah dibuat oleh programmer dan akan dikirimkan ke aplikasi untuk dinilai secara otomatis oleh sistem.

**Hasil Penelitian Kode Sumber**: Setelah melewati penilaian kode sumber maka akan memberikan umpan balik ke programmer setelah itu akan disimpan didatabase untuk mengarsip data penilaiannya.

**Akses Arsip Penilaian**: Ketika data penilaian kode sumber sudah tersimpan secara komputerisasi. Maka penguji dan programmer bisa mengakses datanya dengan API dengan hak akses yang berbeda.

Aplikasi

**Kode Sumber Programmer dan Penguji**: Kode sumber yang telah dikirimkan dari penguji dan programmer.

**Penilaian Kode Sumber**: Kode sumber akan diproses untuk dinilai oleh sistem dari library yang dibuat dan sesuai dengan spesifikasi penguji.

**Hasil Penilaian Kode Sumber**: Setelah diproses penilaian maka dapat diakses oleh penguji maupun programmer.

**Penyimpanan Arsip Penilaian**: Proses komputerisasi untuk menyimpan arsip dari hasil peyimpanan yang nantinya akan disimpan dalam database.

**Data Arsip Penilaian**: Data arsip dari semua programmer akan disimpan pada database ini untuk media informasi baik untuk penguji ataupun programmer.

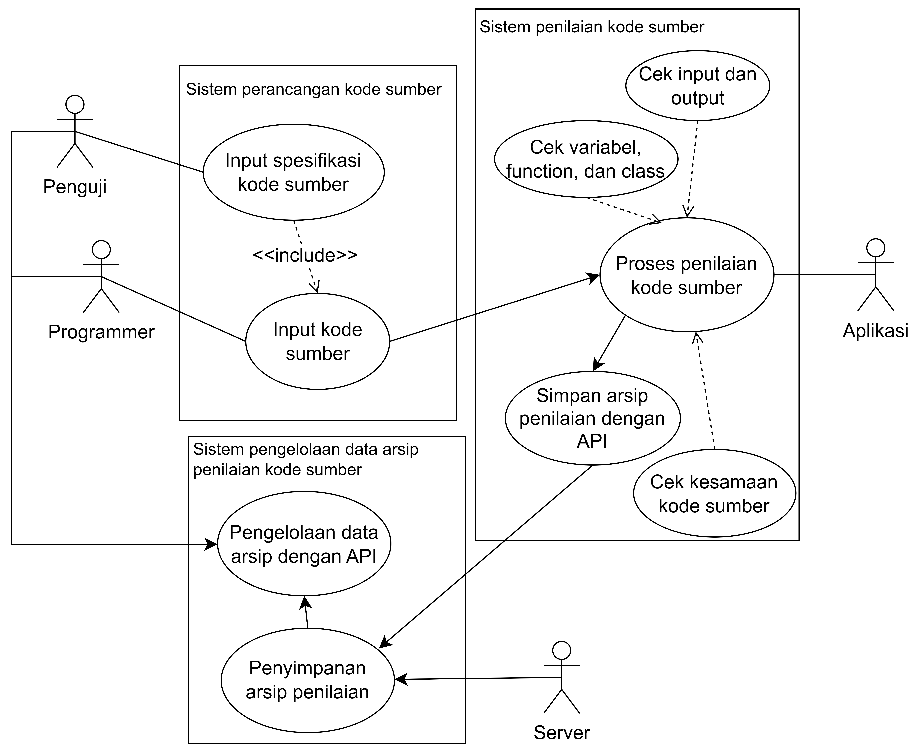
### Spesifikasi Kebutuhan Fungsional Software

Kebutuhan-kebutuhan untuk membuat software yang dapat menilai kode sumber secara otomatis akan dijelaskan pada tabel 3.1 Tabel Daftar Kebutuhan Fungsional, dan tabel 3.2 Spesifikasi kebutuhan pengguna.

Tabel 3. 1 Tabel Daftar Kebutuhan Fungsional

|  |  |  |
| --- | --- | --- |
| Nomor | Kebutuhan Fungsional | Pengguna |
| SKPL-F-01 | Dapat merancang spesifikasi sebuah kode sumber disimpan disistem | Penguji |
| SKPL-F-02 | Dapat mengimplementasikan spesifikasi program ke dalam kode sumber | Penguji, Programmer |
| SKPL-F-03 | Dapat memasukan input data pengujian sebagai acuan penilaian data dalam suatu program yang sesuai dengan spesifikasi kode sumber | Penguji |
| SKPL-F-04 | Dapat menerima input kode dari penguji dan programmer | Aplikasi |
| SKPL-F-05 | Dapat mengetahui kecocokan varibel, function, class untuk melakukan penilaian | Penguji |
| SKPL-F-06 | Memiliki pemahaman untuk mengetahui output yang diharapkan dengan inputan dari penguji | Aplikasi |
| SKPL-F-06 | Mampu Hasil penilaian akan disimpan pada database | Aplikasi |
| SKPL-F-07 | Menyediakan fasilitas untuk mengakses data arsip penilaian dari database dapat diakses melalui API | Programmer, Penguji |
| SKPL-F-08 | Dapat mengetahui spesifikasi dari kode sumber | Aplikasi |
| SKPL-F-09 | Dapat mengetahui kesamaan sebuah kode sumber | Aplikasi |
| SKPL-F-10 | Dapat menampilkan hasil penilaian | Aplikasi |

### Diagram Use Case



Gambar 3. 3 Diagram use case penilaian kode sumber

### Skenario Use Case

1. Input spesifikasi kode sumber

|  |  |
| --- | --- |
| Aktor | Perangkat Lunak |
| **Skenario Normal** |  |
| Memasukan dokumen spesifikasi kode sumber |  |
|  | Mengecek dokumen yang diterima berupa text. |
|  | Spesifikasi tersimpan diaplikasi |
| **Skenario Alternatif** |  |
| 1. Memasukan dokumen spesifikasi kode sumber |  |
|  | 1. Mengecek dokumen yang diterima berupa text |
|  | 1. Menampilkan pesan dokumen tidak valid |
| 1. Memasukan dokumen spesifikasi yang valid |  |
|  | 1. Mengecek dokumen yang diterima berupa text |
|  | 1. Spesifikasi tersimpan diaplikasi |

1. Input kode sumber

|  |  |
| --- | --- |
| Aktor | Perangkat Lunak |
| **Skenario Normal** |  |
| 1. Memasukan kode sumber |  |
|  | 1. Mengecek kode sumber yang diterima berupa file JavaScript |
|  | 1. Kode sumber tersimpan diaplikasi |
| **Skenario Alternatif** |  |
| 1. Memasukan kode sumber |  |
|  | 1. Mengecek kode sumber yang diterima berupa file JavaScript |
|  | 1. Menampilkan pesan kode sumber tidak valid |
| 1. Memasukan kode sumber yang valid |  |
|  | 1. Mengecek kode sumber yang diterima berupa file JavaScript |
|  | 1. Kode sumber tersimpan diaplikasi |

1. Proses penilaian kode sumber

|  |  |
| --- | --- |
| Aktor | Perangkat Lunak |
| **Skenario Normal** |  |
| 1. Menerima kode sumber |  |
|  | 1. Penilaian variabel, function, class yang sesuai dengan spesifikasi |
|  | 1. Penilaian output dengan inputan yang diberikan penguji |
|  | 1. Mengecek kesamaan kode sumber dengan programmer lain |
|  | 1. Hasil penilaian akan dikirimkan ke database melewati API |

1. Penyimpanan arsip penilaian

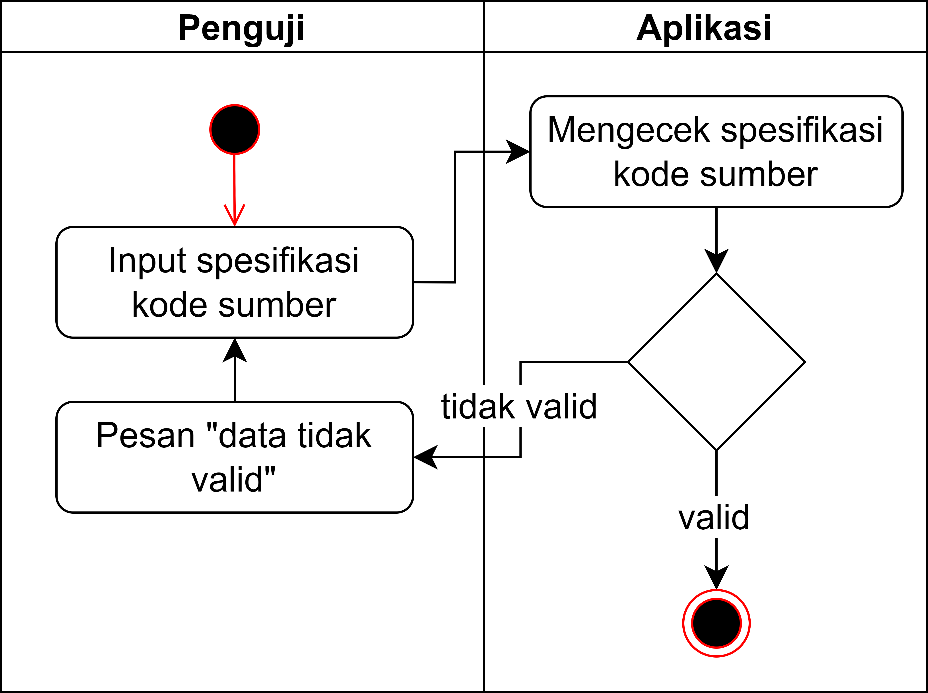
|  |  |
| --- | --- |
| Aktor | Perangkat Lunak |
| **Skenario Normal** |  |
| 1. Menerima input penilaian kode sumber |  |
|  | 1. Mengecek data yang dikirimkan dengan format JSON |
|  | 1. Data penilaian akan tersimpan didatabase |
| **Skenario Alternatif** |  |
| 1. Menerima input penilaian kode sumber |  |
|  | 1. Mengecek data yang dikirimkan dengan format JSON |
|  | 1. Data tidak tersimpan dan mengembalikan umpan balik data yang dikirimkan tidak valid |
| 1. Menerima input penilaian kode sumber yang valid |  |
|  | 1. Data penilaian akan tersimpan didatabase |

1. Pengelolaan data arsip dengan API

|  |  |
| --- | --- |
| Aktor | Perangkat Lunak |
| **Skenario Normal** |  |
| 1. Menerima akses pengelolaan arsip penilaian kode sumber |  |
|  | 1. Mengecek jika yang meminta akses adalah seorang penguji, mereka dapat mengakses seluruh arsip penilaian. |
|  | 1. Mengecek jika yang melakukan permintaan adalah seorang programmer, mereka hanya dapat mengakses data penilaian mereka sendiri. |
|  | 1. Data arsip penilaian berhasil dikirimkan |

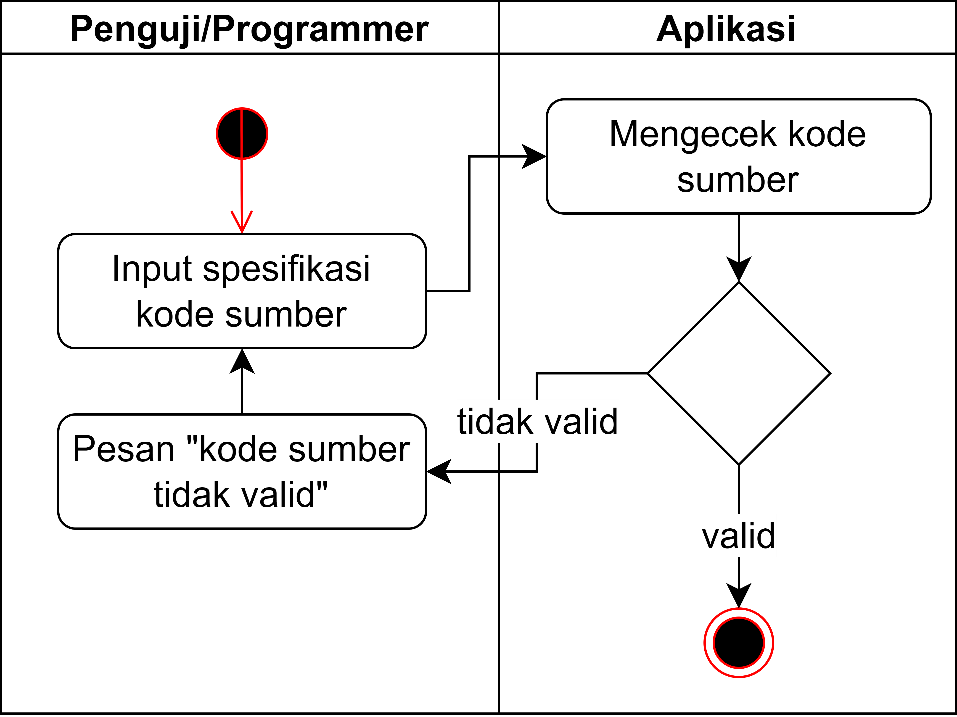
### *Activity* Diagram

1. *Activity* Diagram Input Spesifikasi Kode Sumber



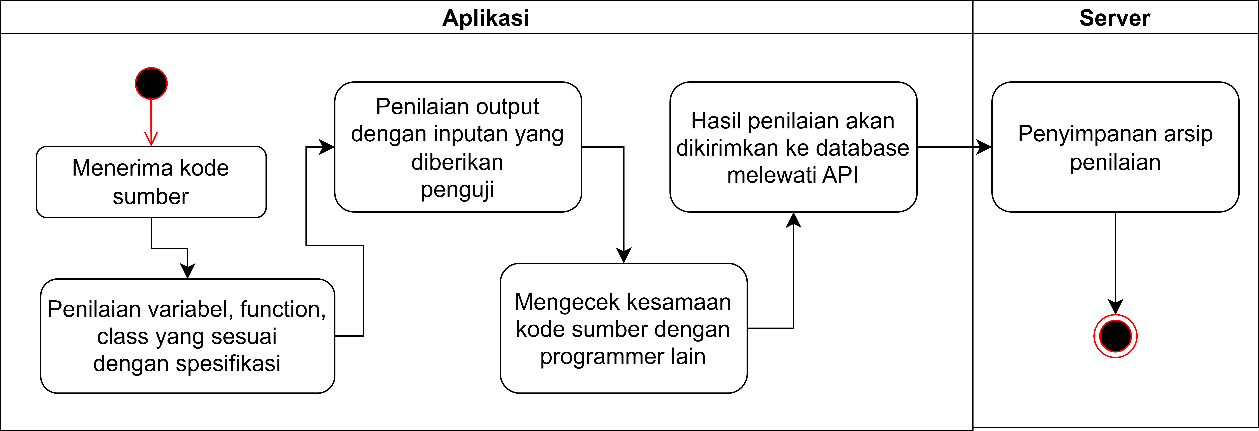
Gambar 3. 4 Activity diagram input spesifikasi kode sumber

1. *Activity* Diagram Input Kode Sumber



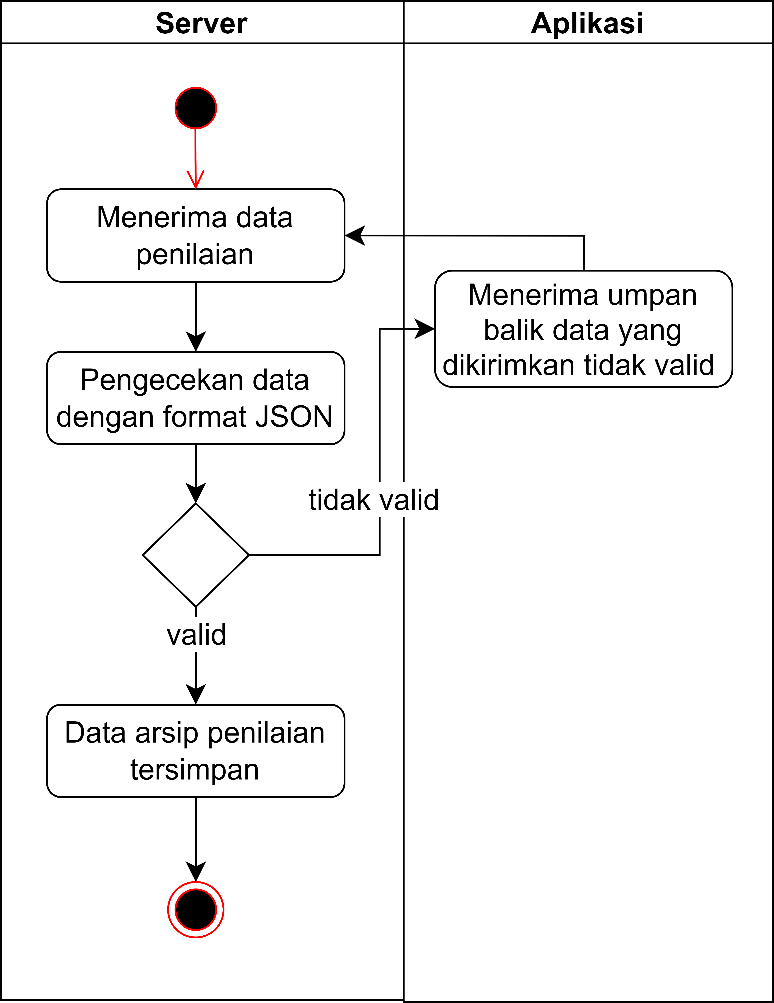
Gambar 3. 5 Activity diagram input kode sumber

1. *Activity* Diagram Proses Penilaian Kode Sumber



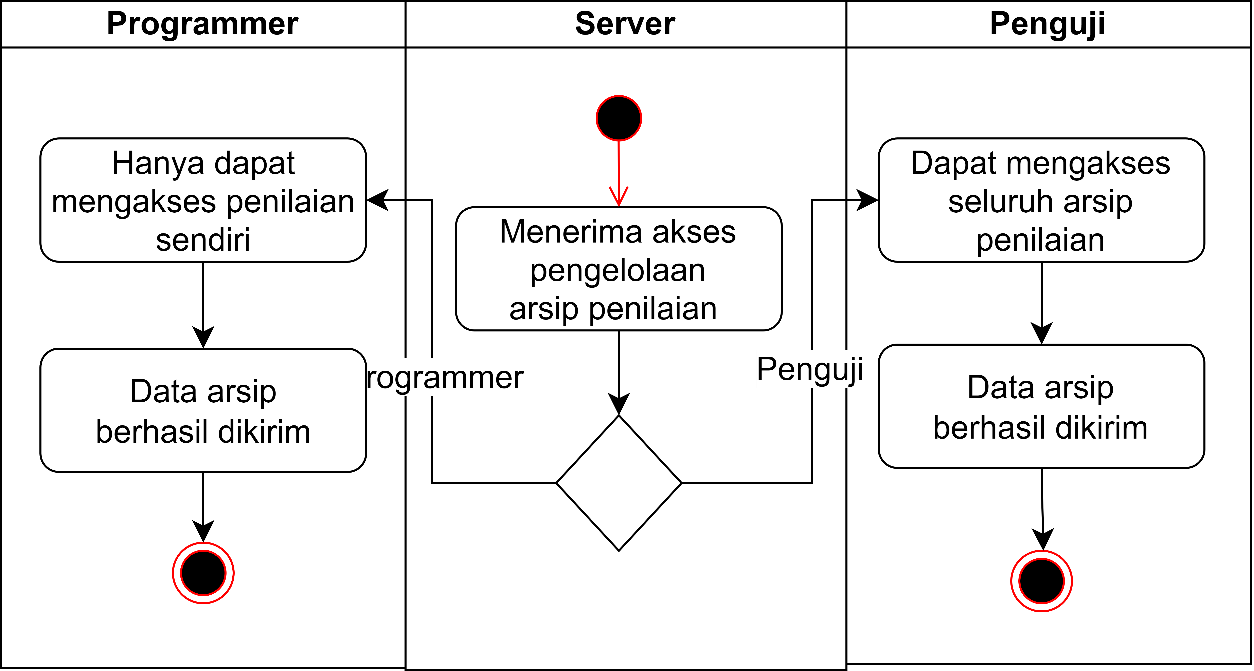
Gambar 3. 6 Activity diagram proses penilaian kode sumber

1. *Activity* Diagram Penyimpanan Arsip Penilaian



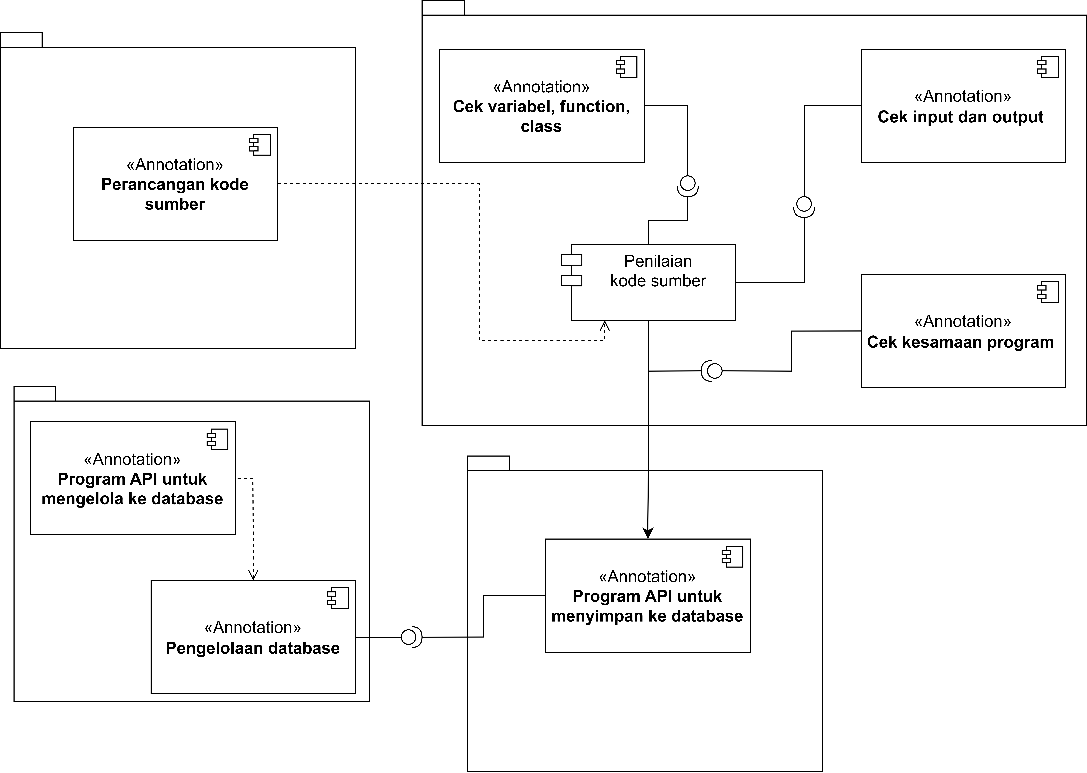
Gambar 3. 7 Activity diagram penyimpanan arsip penilaian

1. *Activity* Diagram Pengelolaan Data Arsip Dengan API



Gambar 3. 8 Activity diagram pengelolaan data arsip dengan API

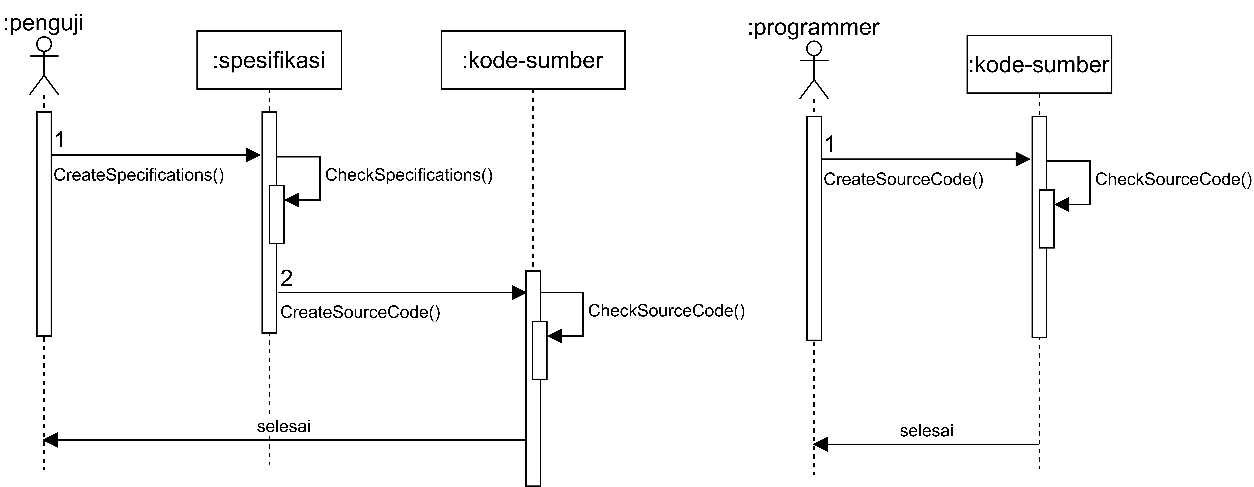
### Diagram Komponen Software



Gambar 3. 9 Activity diagram komponen software

### Diagram Sequence

Sequence diagram perancangan kode sumber



Gambar 3. 10 Sequence diagram perancangan kode sumber

Desain Database111

# IMPLEMENTASI DAN PENGUJIAN

# KESIMPULAN DAN SARAN

**Daftar Pustaka**

[1] Quipper, “Programmer (Software Developer),” Quipper Campus. Accessed: Mar. 08, 2024. [Online]. Available: https://campus.quipper.com/careers/programmer

[2] Scott Wallask, “Kode Sumber,” www.techtarget.com. Accessed: Dec. 13, 2023. [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/source-code

[3] Uladzislau Murashka, “Kode Sumber Review vs. Penetration Testing for Web Application Security,” ScienceSoft. Accessed: Apr. 01, 2024. [Online]. Available: https://www.scnsoft.com/blog/web-applications-security-source-code-review-vs-penetration-testing

[4] Statistics How To, “False Positive dan False Negative: Pengertian dan Contohnya,” www.statisticshowto.com. Accessed: Apr. 01, 2024. [Online]. Available: https://www.statisticshowto.com/false-positive-definition-and-examples/

[5] ADITIA GUSTI ANANDA, “PEMANFAATAN TOOL KATALON STUDIO DALAM PENGUJIAN WEB LEARNING MANAGEMENT SYSTEM (Studi Kasus : Galeri Kejuruan),” 2023.

[6] E. Rahmi, E. Yumami, and N. Hidayasari, “Analisis Metode Pengembangan Sistem Informasi Berbasis Website: Systematic Literature Review,” *ResearchGate*, vol. 7, pp. 821–834, Jan. 2023, doi: 10.33395/remik.v7i1.12177.

Riwayat Hidup

Lampiran