BAB II
LANDASAN TEORI

Pada bab ini akan dijelaskan mengenai dasar-dasar teori dalam literatur yang berkaitan dengan Tugas Akhir. Kemudian akan dijadikan acuan untuk bab selanjutnya, sesuai dengan judul tugas akhir ini, yaitu rancang bangun API *web service* modul Kartu Hasil Studi (KHS) pada sistem informasi akademik STT NF menggunakan RESTful-Spring *Framework*. Berikut konsep teori yang akan di bahas dalam bab ini :

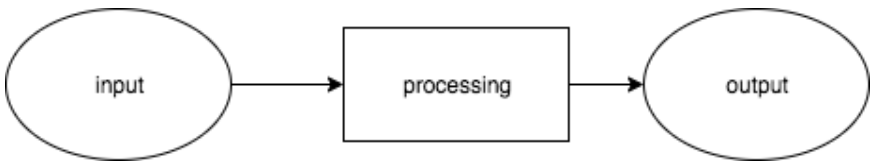
2.1 Tinjauan Pustaka

2.1.1 Sistem Informasi

Sistem informasi adalah seperangkat komponen yang saling terkait yang mengumpulkan, memproses, menyimpan atau mendistribusikan informasi untuk mendukung koordinasi. Informasi adalah data yang berhubungan dengan keputusan.

Sistem informasi selalu menggambarkan, merancang, mengimplementasikan dengan menggunakan proses perkembangan sistematis dan merancang sistem informasi berdasarkan analisis kebutuhan. Jadi, bagian utama dari proses ini adalah mengetahui rancangan dan analisis sistsem. Seluruh aktivitas utama dilibatkan dalam siklus perkembangan yang lengkap (Indrayani, 2011).

Sistem informasi terdiri dari tiga konsep dasar, yaitu: masukan (input), proses (processing) dan keluaran (output). Ketiga elemen dasar ini menghasilkan informasi yang dibutuhkan untuk pengambilan keputusan, pengendalian operasi, analisis permasalahan dan menciptakan produk atau jasa baru.



Gambar 1 - Konsep dasar sistem informasi

Sistem informasi memuat berbagai informasi penting mengenai orang, tempat, dan segala sesuatu yang ada dalam lingkungan sekitar organisasi tersebut. Informasi dapat diartikan suatu data atau input yang telah diolah ke dalam suatu bentuk yang lebih memiliki arti serta dapat dipergunakan dalam pengambilan keputusan. Data yang dimaksud dapat berupa fakta-fakta yang mewakili suatu keadaan, kondisi dan peristiwa yang ada atau terjadi dalam lingkungan organisasi. Data tidak dapat langsung dipergunakan dalam pengambilan keputusan, melainkan harus diolah dan dianalisis agar lebih di pahami, kemudian dimanfaatkan dalam proses pengambilan keputusan.

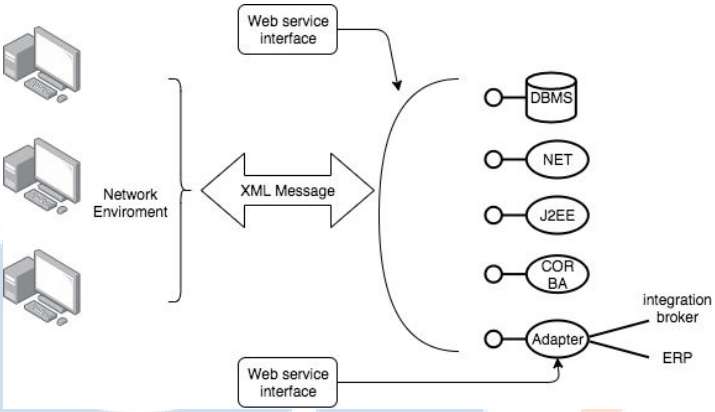
2.1.2 Web Service

1. Web Service

Web Service adalah sebuah software yang dirancang untuk mendukung interoperabalilitas interaksi antar mesin melalui sebuah jaringan. *Web service* secara teknis memiliki mekanisme interkasi antar sistem sebagai penunjang interoperabilitas, baik berupa agregasi (pengumpulan) maupun sindikasi dan data kolaborasi informasi yang bisa diakses melalui internet oleh berbagai pihak menggunakan teknologi yang dimiliki oleh masing-masing pengguna (Andriyanto, 2016).

Alasan menggunakan *web service* adalah kemudahan dalam penggunaan kembali (*reuse*) dan berbagi (*share*) logika yang sama dengan klien yang beragam seperti *mobile*, *desktop*, dan aplikasi web. Jangkauan *web service* yang luas karena *web service* bergantung pada standar yang terbuka, dapat beroperasi pada platform yang berbeda, serta tidak bergantung pada teknologi eksekusi yang mendasarinya. Semua *web service* setidaknya menggunakan HTTP dan format penukaran data standar berupa XML, JSON, atau media lain. Selain itu, *web service* menggunakan HTTP dalam dua cara yang berbeda, yaitu sebagai protokol transportasi untuk menyampaikan data (Daigneau, 2012).

Web service bertujuan untuk meningkatkan kolaborasi antar pemrograman dan perusahaan, yang memungkinkan sebuah fungsi di dalam *web service* dapat dipinjam oleh aplikasi lain tanpa perlu mengetahui detail pemrograman yang terdapat di dalamnya.

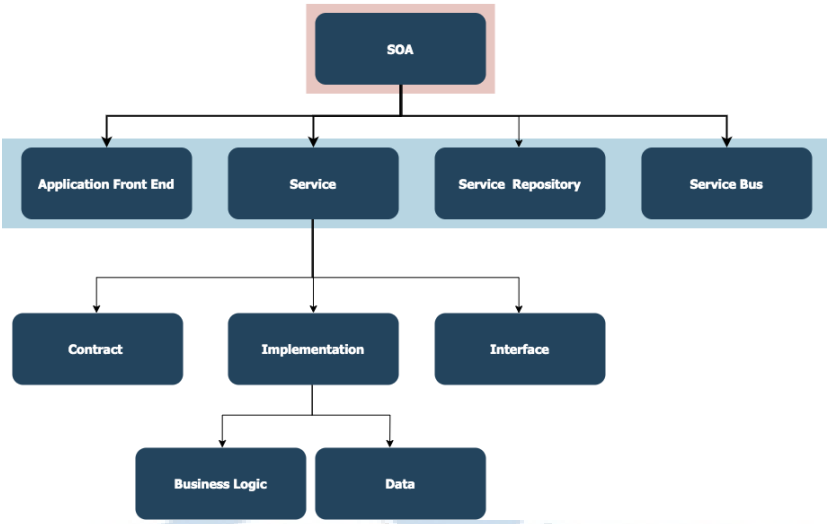


Gambar 2 - Antarmuka *web service* dengan sistem lainya

2. SOA

Service Oriented Architecture atau SOA didefinisikan sebagai kebijakan, praktek, kerangka kerja yang memungkinkan fungsionalitas aplikasi disediakan dan dikonsumsi sebagai seperangkat *service* pada sebuah unit yang sesuai dengan kebutuhan *service customer*. *Service* dapat digunakan, dipublikasikan, ditemukan, dan diabstraksikan menggunakan standar antarmuka (Sprott D & Wilkes, 2004)

Gambar 3 memberikan struktur hirarki dari SOA. SOA adalah sebuah arsitektur perangkat lunak yang didasarkan pada konsep-konsep kunci dari sebuah aplikasi *front-end*, *service*, *service repository*, dan *service bus*. Sebuah *service* terdiri dari sebuah kontrak, satu atau lebih dari dua antar muka, dan sebuah implementasi.



Gambar 3 - Struktur Data Hirarki SOA

3. REST Web Service

REST merupakan singkatan dari *Representational State Transfer*. Istilah REST atau RESTful pertama kali diperkenalkan oleh Roy Fielding pada disertasinya di tahun 2000. RESTful bukanlah sebuah standar protokol *web service,* melainkan hanya sebuah gaya arsitektur. ide dasar dari arsitektur REST adalah bagaimana menghubungkan jalur komunikasi antar mesin atau aplikasi melalui HTTP sederhana. Sebelum adanya REST, mekanisme atau protokol *middleware* yang cukup kompleks seperti DCE, CORBA, RPC, ataupun SOA.

Arsitektur REST mampu mengeksploitasi berbagai kelebihan dari HTTP yang digunakan untuk kebutuhan *web service*. Walaupun SOAP juga dapat menggunakan protokol HTTP, namun hanya terbatas untuk kebutuhan transport saja. HTTP sendiri merupakan sebuah protokol standar di dunia *World Wide Web* yang berbasis *synchronous request/response.* Protokol tersebut sangat sederhana : *client* mengirimkan sebuah *request message* yang mencakup HTTP method yang akan di invokasi, lokasi *resource* dalam format URI, serta pilihan format pesan (pada dasarnya dapat berupa format apa saja seperti HTML, *plain text*, XML, JSON, ataupun data *binary*), kemudian server akan mengirimkan *response* sesuai dengan spesifikasi yang diminta oleh *client.* Selama ini, yang berfungsi sebagai aplikasi *client* adalah sebuah *web browser* yang memfasilitasi komunikasi antara

mesin dengan manusia. Dengan adanya REST, aplikasi *client* dapat berupa aplikasi apa saja hanya dengan memanfaatkan HTTP. Berikut ini beberapa prinsip arsitektur dari REST yang dikutip dari sebuah buku berjudul "RESTful java with JAX-RS" (Burke, 2009) :

1. ***Addressability***

   Addrressability merupakan sebuah ide dimana setiap objek *resource* pada suatu system dapat dicapai hanya dengan melalui sebuah *unique identifier*. Pada dunia REST, *addressability* dikelola dengan penggunaan *Uniform Resource identifier* (URI).

2. ***Constrained & Uniform Interface***

   Ide dari prinsip ini adalah menyediakan berbagai layanan melalui antarmuka atau pemanggilan *method* atau *procedure* yang seragam. Pada system COBRA ataupun SOAP. Pengembangan *client* harus mengetahui *method* apa saja yang disediakan oleh *web service server*. Pemanggilan *method* tersebut dikenal dengan istilah RPC (*Remote Procedure Call).*

   Namun pada system REST, *method* atau *procedure* yang digunakan untuk layanan apapun hanyalah *method-method* yang disediakan pada HTTP. Istilah yang biasa digunakan untuk menyatakan prinsip *uniform interface* pada REST adalah CRUD (*Create, Read, Update, Delete*). Berikut adalah ulasan detail mengenai *method-method* tersebut :

Table 1 - HTTP *method* dan penggunaanya dalam REST

Metode	Deskripsi
GET	Mendapatkan (read) sebuah sumber daya (*resource)* yang diidentifikasi dengan URI (*Uniform Resource Identifier*).
POST	Mengirimkan sumber daya

	(*resource*) ke *server.* Digunakan untuk membuat (*create*) sumber daya baru.
PUT	Mengirimkan sumber daya (*resource)* ke *server*, digunakan untuk memasukkan (*insert*) atau memperbaharui (*update*) sumber daya yang tersimpan.
DELETE	Menghapus sumber daya (*resource*) yang diidentifikasi dengan URI.
HEAD	Mendapatkan metadata (*response header) dari sumber daya* (*resource*) yang diidentifikasi dengan URI.

Sampai sekarang, REST *best practice* dipilih oleh para pengembang dalam melakukan desain REST *web service* sehingga desain antara yang satu dengan yang lain bisa berbeda-beda tergantung masalah yang dihadapi. Beberapa aturan *best practice* tersebut adalah *identifier*, interaksi menggunakan HTTP, serta format balasan harus valid dan konsisten.

Dengan menggunakan protokol HTTP, URI dapat dijadikan sebagai media yang digunakan untuk mengakses *resources* dari *server*. Hal ini disebut dengan URI *tunneling.* URI *tunneling* mempergunakan URI untuk mentransfer informasi pada antar sistem yang dalam jaringan dengan melakukan *encode* pada URI itu sendiri. Dengan mengirim HTTP *method* yang telah disebutkan sebelumnya, server dapat melakukan eksekusi terhadap suatu program yang menghasilkan atau mengambil suatu *resource* dan mengirimnya kembali ke *client.* Dalam proses ini terjadi

proses mapping dan URI menjadi *method call* pada *server* yang dituju (Webber, 2010).

3. ***Stateless Communication***

Pada dunia REST, seperti halnya pada dunia *World Wide Web*, *stateless* berarti tidak ada *client session* data yang disimpan pada *server*. *Server* hanya menyimpan dan mengelola *state* dari *resource* yang digunakan. Jika terdapat kebutuhan informasi yang *session-specific*, hal tersebut seharusnya diatur pada sisi *client*. Karakteristik tersebut memberikan kemudahan dari sisi *scalability* suatu *server* yang berbasis *cluster*. Untuk mengembangkan ukuran *cluster.* Yang perlu dilakukan hanya menambah mesin baru serta tidak perlu memikirkan kepemilikan data terhadap *client* tertentu.

4. **Format Pesan Pertukaran**

Saat ini terdapat dua buah format pesan yang dipertukaran (*data interchange format*) yang digunakan pada *web service*, yaitu XML, dan JSON. XML adalah singkatan dari *Extensible Markup Language* yang merupakan turunan dari format *Standard Generalized Markup Language* (SGML) (J.Bosak, 1997). XML dirancang sebagai alternatif dari SGML untuk mengatasi masalah kompleksitas pada SGML itu sendiri. Pertimbangan fundamental penggunaan XML adalah mencakup unsur *simplicity* dan *human readability*.

Sedangkan JSON adalah singkatan dari *Javascript Object Notation* yang merupakan objek asli bawaan *javascript* JSON didesain sebagai format pesan pertukaran yang *human readable* serta mudah dibaca (parsing) oleh program komputer. Pada aplikasi berbasis *javascript*, penggunaan JSON sebagai format pesan pertukaran pesan akan membawa dampak performa yang cukup signifikan dibandingkan bila menggunakan *library* tambahan untuk membaca data dari *Document Object Model* (DOM).

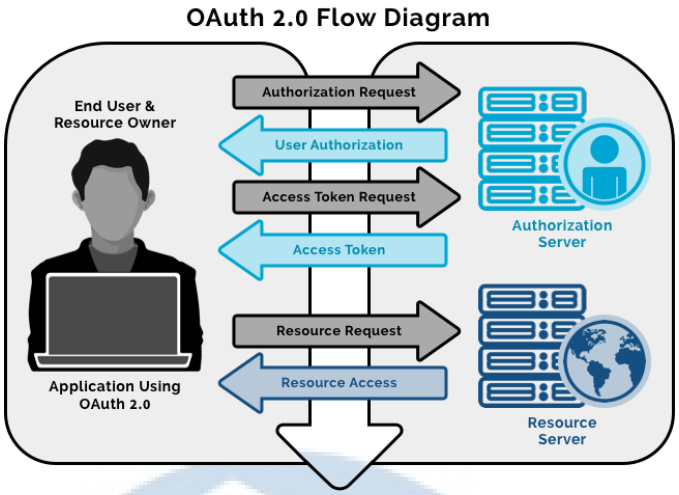
4. OAuth 2

OAuth (Open Authorization) adalah sebuah protokol standar otorisasi standar terbuka yang memungkinkan pengguna mengakses aplikasi tanpa perlu berbagi *password*. Pemilik aplikasi mengintegrasikan *credential* milik pengguna dengan teknologi otentikasi yang berasal dari penerbit *API (Application Programming Interface)*. *OAuth* juga mengizinkan otorisasi API yang sudah terlindungi yang berasal dari desktop ataupun aplikasi web melalui metode sederhana dan standard. Mengatur lalu lintas data antar aplikasi dan digunakan saat pembuat API mengetahui siapa yang terlibat dan berkomunikasi di dalam sistem.

OAuth 1 (juga dikenal sebagai RFC 5849). Diterbitkan pada tanggal 4 Desember 2007, direvisi pada tanggal 24 Juni 2009, dan diselesaikan pada bulan Desember 2010 yang memberikan pengaruh penting pada perkembangan keamanan web API tanpa harus pengguna berbagi *username* dan *password* mereka. Adapun pencipta dan penggagas dari otentikasi *OAuth* berbasis API ini adalah E.Hammer-Lahav, Ed.

OAuth 2 adalah *project* lanjutan dari protokol *OAuth* yang asal mulanya diciptakan pada akhir tahun 2006. *OAuth* 2 lebih menekankan pada kemudahan *client* sebagai pemilik dan pengembang aplikasi dengan memberikan otorisasi khusus di berbagai aplikasi. *OAuth* berada dalam pengembangan *IETF OAuth WG* dan didasarkan pada usulan *WRAP OAuth*. *WRAP (Web Resource Authorization Protocol)* adalah profil *OAuth* yang memiliki sejumlah kemampuan penting yang tidak tersedia di versi sebelumnya. Spesifikasi terbaru dari *OAuth* disumbangkan kepada *IETS OAuth WG* dan merupakan dasar dari terciptanya *OAuth* versi 2.

Dengan banyaknya aplikasi *web* yang mengadopsi kolaborasi penggunaan jaringan sosial, pengembang aplikasi memiliki kesempatan untuk menghubungkan pengguna dengan aplikasi dimanapun mereka berada, yang dapat meningkatkan efektivitas dan efisiensi terjadinya transaksi pengolahan data didalam sistem tersebut. *OAuth* menyediakan kemampuan terhubung dengan sistem secara aman, sehingga pengguna tidak perlu menyerahkan *password*nya.



Gambar 4 - Oauth 2 diagram

Terdapat 4 peran utama dalam mekanisme kerja *OAuth* 2 yakni :

A. *Resource server*

Resource server (Sumber Daya Server) yang digunakan oleh pengguna yang memiliki API *(Application Programming Interface)* dan dilindungi oleh *OAuth*. *Resource server* merupakan penerbit API yang memegang dan memiliki kekuasaan pengaturan data, seperti foto, video, kontak, atau kalender.

B. *Resource Owner*

Memposisikan diri sebagai pemilik sumber daya , yang merupakan pemilik dari aplikasi. Pemilik sumber daya memiliki kemampuan untuk mengakses sumber daya *server* dengan aplikasi yang sudah tersedia.

C. *Client*

Sebuah aplikasi yang membuat permintaan API pada *Resource server* yang telah diproteksi untuk kepentingan pemilik *Resource* dengan melakukan otorisasi.

D. *Authorization Server*

Authorization Server mendapat persetujuan dari pemilik sumber daya (*Resource Owner*) dengan melakukan dan memberikan akses token kepada *client* untuk mengakses sumber daya yang diproteksi yang sudah tersedia pada *Resource Server.*

5. API

API Merupakan *software interface* yang terdiri atas kumpulan instruksi yang disimpan dalam bentuk *library* dan menjelaskan bagaimana agar suatu *software* dapat berinteraksi dengan *software* lain. Penjelasan ini dapat dicontohkan dengan analogi apabila akan dibuangun suatu rumah. Dengan menyewa kontraktor yang dapat menangani bagian yang berbeda, pemilik rumah dapat memberikan tugas yang perlu dilakukan oleh kontraktor tanpa harus mengetahui bagaimana cara kontraktor tersebut menyelesaikan pekerjaan tersebut. Dari analogi tersebut, rumah merupakan *software* yang akan dibuat, dan kontraktor merupakan API yang mengerjakan bagian tertentu dari *software* tersebut tanpa harus diketahui bagaimana prosedur dalam melakukan pekerjaan tersebut.

6. Swagger

Swagger adalah salah satu *tools* untuk kita bisa mendokumentasikan API secara lebih efektif, dan memungkinkan kita mendeskripsikan struktur API kita, sehingga mesin dapat membacanya.

Dengan membaca struktur API, Swagger dapat secara otomatis mendokumentasikan API dengan baik dan rapi, dengan membuat *client libraries* dalam banyak bahasa, juga memungkinkan untuk melakukan pengujian otomatis, dengan cara mengambil API kita dan mengembalikan dalam bentuk JSON atau YAML, yang berisi deskripsi mendetail dari isi API kita, dan pada dasarnya ini adalah OpenAPI. (Swagger support by SMARTBEAR)

2.1.3 Spring Framework

1. Java

Java adalah bahasa pemrograman serbaguna. Java dapat digunakan untuk membuat suatu program sebagaimana kita membuatnya dengan bahasa seperti Pascal atau C++. Yang lebih menarik, java juga mendukung sumber daya internet yang saat ini saat ini popular, yaitu *World Wide Web* atau yang sering disebut *web*

saja. Java juga mendukung aplikasi klien atau server, baik dalam jaringan local (LAN) maupun jaringan berskala luas (WAN).

Java dikembangkan oleh Sun Microsystems pada Agustus 1991, dengan nama semula OAK, konon OAK adalah pohon semacam Jati yang terlihat dari jendela tempat pembuatnya, James Gosling, bekerja. Ada yang mengatakan bahwa OAK adalah singkatan dari *"Object Application Kernel"*, tetapi ada yang menyatakan hal itu muncul setelah nama OAK diberikan pada januari 1995, karena nama OAK dianggap kurang komersial, maka diganti menjadi Java.

Program java bersifat tidak bergantung pada *platform*, artinya java dapat dijalankan pada beberapa jenis dan sistem operasi. Beberapa platform dan sistem operasi yang didukung oleh java dapat dilihat pada tabel 2.

Table 2 - Java pada berbagai sistem operasi

No.	Sistem Operasi	Vendor
1.	AIX	IBM
2.	DG/UX	Data General Corporation
3.	Digital OpenVMS	Digital Equipment Corporation
4.	Digital Unix	Digital Equipment Corporation
5.	HP-UX	Hewlett Packard
6.	IRIX	Silicon Graphics
7.	Linux	Banyak Perusahaan
8.	MacOS	Apple
9.	Netware	Novell
10.	OS/2	IBM
11.	OS/390	IBM
12.	OS/400	IBM
13.	Solaris	Sun Microsystems
14.	Windows family	Microsoft Corporation

Sebagaimana halnya C++, salah satu bahasa yang menginspirasi Java, Java juga merupakan bahasa pemrograman berorientasi objek. Sebagai bahasa pemrograman berorientasi objek, java menggunakan kelas untuk membentuk suatu objek. Sejumlah kelas sudah tersedia dan kita dapat menggunakannya dengan mudah dan bahkan dapat mengembangaknya lebih jauh melalui konsep pewarisan. Pewarisan adalah salah satu sifat yang ada pada bahasa pemrograman berorientasi objek, yang memungkinkan sifat-sifat suatu objek diturunkan dengan mudah ke objek lain.

2. Spring Framework

Spring framework adalah framework *open source* yang menyediakan infrastruktur yang komprehensif dalam mengembangkan aplikasi *java* dengan mudah dan cepat. Spring pertama kali ditulis dan dirilis oleh Rod Johnson dengan lisesi Apache 2.0 pada bulan juni 2003. Spring akan membantu *programmer* dalam pengembangan aplikasi dengan *build* yang sederhana, *portable*, cepat dan sistem berbasis JVM yang fleksibel. Spring dapat digunakan untuk melakukan pengaturan deklarasi manajemen transaksi, *remote access* dengan menggunakan RMI atau *database*. Spring juga memungkinkan kita untuk semua modul spring dalam aplikasi apabila tidak diperlukan.

Spring Framework menggunakan teknik pemrograman yang sederhana, model pemrograman dengan spring cukup mudah, namun lebih tertata. Hal ini memudahkan bagi para pemula untuk mempelajarinya, dikarenakan *framework* spring mendorong untuk membuat kode program yang modular dan independen. Hasilnya, kode program yang dibuat akan lebih rapi, mudah untuk di test, dan terstruktur dengan lebih baik.

Kelebihan spring *framework* dibandingkan *framework* lainya:

- *Build Anything* : Menuliskan kode yang bersih dan rapi, dapat diuji terhadap komponen infrastruktur pilihan dan melakukan tugas apapun.

- *Run Anywhere* : Aplikasi berbasis spring berjalan dimana saja yang menggunakan JVM.
- *Rest Assured* : Spring menyediakan model pemrograman terbuka yang komprehensif, kohesif, dan dipahami secara luas dan didukung dengan baik.
- Pola *MVC* (*Model View Controller*)
- Hemat Waktu : Kita tidak perlu menghabiskan waktu untuk menuliskan kode program, kita bisa menggunakan fungsi atau *class* bawaan dari *framework* yang kita gunakan, seperti :
    a. *Modul generator*, akan menghasilkan modul yang diinginkan menjadi cepat.
    b. *ORM (Object Relational Mapping),* dengan ORM kita tidak perlu lagi untuk menuliskan sintaks SQL yang spesifik untuk *database* tertentu.
    c. Ketersediaan plugin.

3. Data Source

Data Source adalah sumber utama dimana data berasal. Dalam pemrograman komputer, *data source* bisa berasal dari basis data, *data set, spreadsheet*, dan lain-lain. Ketika data ditampilkan dalam halaman web atau aplikasi apapun, data diambil dari sumber data dapat disajikan, dalam format yang sudah didefinisikan dalam pemrograman.

4. Spring Boot

Spring Boot merupakan sebuah *framework* Java yang digunakan untuk membuat aplikasi berbasis web dan aplikasi *enterprise*. Spring Boot merupakan *framework* baru yang dibuat untuk mempermudah *bootstrapping* dan pengembangan pada aplikasi. Kelebihan Spring Boot jika dibandingkan dengan *framework* lain, pada Spring Boot memberikan berbagai macam fitur yang mengutamakan kebutuhan bisnis modern. Pada Spring Boot memberikan fleksibilitas untuk melakukan konfigurasi beans dengan berbagai cara seperti

XML, *Annotations*, dan *JavaConfig*. Selain itu pada Spring Boot mudah digunakan karena kemampuan manajemen transaksi basis data, serta menyederhanakan integrasi dengan *framework* Java lain seperti *JPA* atau *Hibernate ORM*, Struts, JSF, dan lainya.

Spring Boot pada penelitian ini digunakan sebagai bagian *backend* yang digunakan untuk membuat *web service*. Pada Spring Boot sendiri *layer-layer*nya jelas dan dapat digunakan untuk membuat sistem *microservice* yang r*eusable* dan *scalable*.

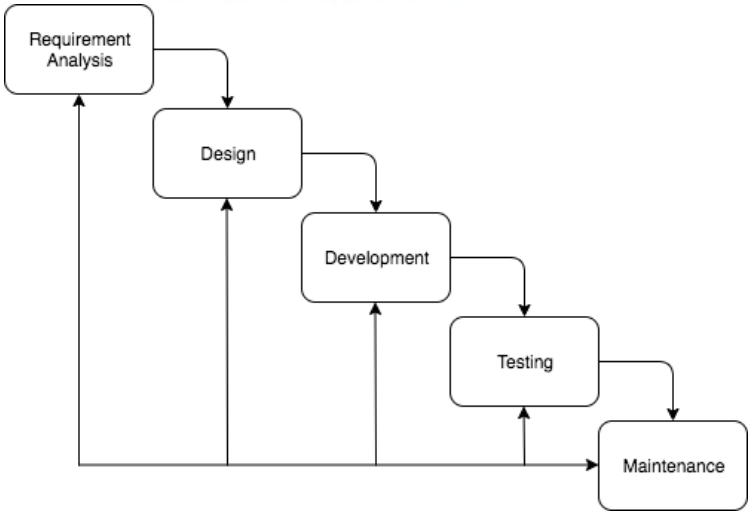
2.1.4 Metode Pengembangan

1. Waterfall

Waterfall adalah suatu metode dalam *Project Management* yang dilakukan secara tahap demi tahap. *waterfall* bertujuan untuk menghasilkan suatu *software* atau sistem yang telah ditentukan dengan kualitas terbaik.

Waterfall mengutamakan interaksi dengan *stakeholders* di awal *project* dikerjakan untuk mendapat gambaran yang jelas, dan saat menggunakan proses *waterfall*, setiap tim menyelesaikan suatu tahap, maka tim harus lanjut ke tahap berikutnya dan diharapkan tanpa mengulang proses sebelumnya.

Tahapan dari metode *waterfall* dapat dilihat pada gambar dibawah ini.



Gambar 5 - Proses Waterfall

Berikut adalah penjelasan dari setiap fase (*workflow)*:

a. *Requirement Analysis* : Dalam fase ini penulis melakukan analisis terhadap kebutuhan sistem, penulis harus betul-betul memahami dan mendefinisikan tujuan dari sistem yang ingin dibangunm yang kemudian hasil analisis tersebut akan dibuat dokumen spesifikasinya untuk dijadikan persyaratan dalam pengembangan aplikasi.
b. *Design :* Pada fase ini penulis merancang suatu arsitektur sistem berdasarkan hasil dari fase sebelumnya.
c. *Development* : Pada fase ini keseluruhan desain sistem yang telah disusun akan diubah menjadi kode-kode program dan modul-modul yang nantinya akan diintegrasikan menjadi sebuah sistem yang lengkap dan sesuai dengan desain yang dirancang sebelumnya.
d. *Testing* : Dalam fase ini keseluruhan sistem yang sudah dibuat akan diuji, apakah telah berfungsi dengan baik, atau tidak, dan yang tidak kalah penting apakah sesuai dengan desain yang dirancang sebelumnya atau tidak.
e. *Maintenance* : Pada fase ini adalah pemeliharaan sistem yang termasuk diantaranya instalasi dan proses perbaikan sistem apabila ditemukan adanya kesalahan atau *bug* yang tidak ditemukan pada tahap *testing*.

Setiap fase di atas harus selesai sebelum masuk ke fase selanjutnya, agar tidak terjadi tumpang tindih dalam setiap fasenya. Hal ini juga bisa disebut sebagai *linear-sequential life model.*

2. UML

Unified Modeling Language (UML) adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk memvisualisasi, menspesifikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis *Object-Oriented*. UML sendiri juga memberikan standar penulisan sebuah sistem *blue print*, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam

bahasa program yang spesifik, skema basis data, dan komponen-komponen yang diperlukan dalam sistem perangkat lunak (Booch, 2005)

UML adalah hasil kerja sama dari konsorisium berbagai organisasi yang berhasil dijadikan sebagai standar baku dalam OOAD (*Object Oriented Analysis & Design)* dan banyak perusahaan yang turut serta untuk berkontribusi untuk UML diantaranya *Oracle, Digital Equipment Corp, Hewlett-Packard Company, i-Logic,* dan masih banyak lagi. Notasi yang digunakan dalam UM sudah dibakukan di dalam pengembangan sistem, UML memiliki serangkaian diagram yang memungkinkan bagi seorang analis sistem untuk membuat suatu cetak biru sistem yang komprehensif kepada klien, *programmer* dan tiap orang yang terlibat dalam proses pengembangan sistem tersebut. Di dalam UML terdapat diagram yang digunakan sebagai alat bantu dalam proses desain sistem, diagram merupakan gambaran atau dokumentasi aspek dari sebuah sistem.

Tipe diagram dalam UML :

5. *Use case diagram*

   Use case diagram memfasilitasi komunikasi di antara analis dan pengguna, serta di antara analis dan klien. Diagram *use case* menunjukkan 3 aspek dari sistem antara lain : *actor, use case,* dan *sub system boundary*. Menggunakan kembali *use case* yang sudah ada merupakan hal yang lumrah, bisa menggunakan <<*include*>> untuk menunjukkan sebuah *use case* adalah bagian dari *use case* yang lain, lalu penggunaan <<*extend*>> untuk membuat *use case* baru dengan melakukan penambahan langkah-langkah pada *use case* yang sudah ada.

6. *Class diagram*

   Class diagram dalam notasi UML digambarkan dengan kotak. Nama *class* menggunakan huruf besar pada awal kalimatnya dan diletakkan di atas kotak. *Class* memiliki atribut maupun operasi, atribut adalah properti dari sebuah *class*. Atribut ini melukiskan batas nilai yang mungkin ada objek dari *class*, sebuah *class* mungkin mempunyai nol atau lebih atribut.

Operasi pada *class* adalah sesuatu yang bisa dilakukan oleh sebuah *class*, salah satu bentuk operasi adalah *function* yang berguna untuk mendapatkan nilai setelah operasi dijalankan.

7. *Sequence diagram*

   Sequence diagram menambahkan dimensi waktu pada interaksi di antara objek-objek. Pada *sequence diagram*, *participant* diletakkan di atas dan waktu ditunjukkan dari atas ke bawah dengan garis putus-putus yang disebut dengan *lifeline*. Kotak kecil yang terdapat pada *lifetime* menyatakan *activation, activation* adalah menandakan adanya *operation* yang diterima dari *participant* lain untuk dijalankan oleh *participant* yang menerimanya. Sedangkan *message* adalah pesan yang berada pada tanda panah yang menghubungkan *lifetime* antara *participant* satu dengan *participant* lainya. Terdapat juga *class boundary* yang mendeskripsikan objek-objek yang mewakili *interface* di antara *actor* dan sistem dan *class entity* yang mendeskripsikan objek-objek yang mewakili entitas yang ada di sebuah domain aplikasi, *class entity* lebih berhubungan dengan struktur yang ada di sistem *database.*

8. *Collaboration diagram*

   Collaboration diagram merupakan bentuk lain dari *sequence diagram*. *Collaboration diagram* merupakan asosiasi di antara objek-objek. *Content message* pada *collaboration diagram* ditunjukkan dengan label, panah di dekat garis asosiasi menandakan objek menerima pesan, dan angka pada *message* menunjukkan urutan *message*.

9. *Activity diagram*

   Activity diagram adalah teknik untuk mendeskripsikan logika yang bersifat prosedural, atau *workflow* pada bisnis proses dengan kasus yang berbeda-beda. Peran *activity diagram* sama halnya seperti *flowchart*, tetapi terdapat perbedaan di antara keduanya yaitu pada *activity diagram* mendukung perilaku parallel sedangkan *flowchart* tidak bisa, hal inilah yang juga menjadi kelebihan dari *activity diagram* disbanding dengan

flowchart. *Activity diagram* berguna untuk menunjukkan siapa mengerjakan apa dengan teknik *partition*.

2.1.5 Metode Pengujian

1. *Blackbox Testing*

Pengujian dengan metode *Blackbox* juga biasa disebut dengan *behaviorial testing* yang berfokus pada *functional requirement* dari sebuah perangkat lunak. Metode pengujian *Blackbox* ini memungkinkan seorang *software engineer* dapat memberikan sekumpulan input guna menguji semua fungsionalitas *requirement* dari sebuah program. *Blackbox testing* ini merupakan pelengkap dari tahapan pengujian sebuah *software* selain metode *white box*. Metode *Blackbox* ini tidak seperti *white box* yang dapat dilakukan di awal tetapi pengujian *Blackbox* ini dilakukan pada tahap akhir, artinya sebelum program secara keseluruhan selesai , program dapat diuji dengan metode *white box* untuk menguji *control structure* dari sebuah program, sedangkan *Blackbox* mengabaikan *control structure* dan lebih berfokus pada domain informasi apakah input yang diberikan penguji menghasilkan keluaran yang diharapkan oleh *user*. Pada pengujian *Blackbox*, kesalahan yang berusaha ditemukan antara lain (Pressman, 2001):

1. Kesalahan performa.
2. Kesalahan *interface*.
3. Kesalahan dalam struktur data atau akses *database* eksternal.
4. Fungsi-fungsi yang salah atau hilang.

Salah satu metode *Blackbox testing* yaitu *equivalence class partitioning* adalah metode uji coba *Blackbox* yang membagi domain *input* dari program menjadi beberapa kelas data dari kasus uji coba yang dihasilkan. *Equivalence class partitioning* berusaha untuk mendefinisikan kasus uji dapat menemukan sejumlah kesalahan, kasus uji yang didesain untuk *equivalence class partitioning* berdasarkan pada evaluasi dari ekuivalensi jenis atau *class* untuk kondisi *input*.

Kelas-kelas yang ekuivalen merepresentasikan sekumpulan keadaan *valid* dan *invalid* untuk kondisi *input*. Kondisi *input* sendiri dapat berupa nilai numeric yang spesifik, kisaran nilai (*range*), sekumpulan nilai yang berhubungan (himpunan), atau kondisi *Boolean* (Pressman, 2001). Terdapat strategi untuk mendefinisikan suatu *equivalence class*, antara lain :

1. Lakukan identifikasi terhadap *input equivalence class*.
2. Berdasarkan pada kondisi seperti apa yang akan di *input*, spesifikasikan jenis *output* apa yang diharapkan.
3. Identifikasi data *valid* maupun *invalid* yang akan dijadikan *input equivalence class.*
4. Definisikan satu atau lebih *test cases* untuk tiap *class,* yaitu test case untuk data *valid* maupun *test case* untuk data *invalid.*

2. Postman

Postman adalah aplikasi yang digunakan untuk pengujian *web service* yang diciptakan oleh Abhinav Astahana, seorang *programmer* dan desainer di Bangalore, India. Postman membuat pengujian, pengembangan dan pendokumentasian API menjadi sangat mudah, yang memungkinkan pengguna dengan cepat, sederhana dalam melakukan HTTP *request*. Postman tersedia paket aplikasi google chrome dan aplikasi *browser* google chrome. Versi aplikasi yang termasuk fitur *advanced* seperti mendukung OAuth 2 dan *bulk uploading* atau *importing* yang belum tersisa dalam versi *browser.* Pada versi *browser* terdapat sedikit fitur tambahan, seperti mendukung *session cookies*, yang belum tersedia pada paket versi aplikasi. Pada saat publikasi, Postman REST *Client* adalah salah satu aplikasi yang mendapatkan *rating* tertinggi dalam *chrome web store*. Lebih dari 348.000 pengguna unik dan lebih dari 63.000 koleksi berbagi via Postman (Hunt, 2006).

2.2 Penelitian Terkait

2.2.1 Penelitian Terkait

Pada penelitian ini, penelitian melakukan studi literatur penelitian terkait dengan permasalahan yang peneliti ambil. Hal ini bertujuan untuk mengetahui tingkat kebutuhan dalam permasalahan untuk penelitian yang sedang dilakukan. Hal ini bertujuan untuk mengetahui posisi penelitian yang sedang dilakukan. Berikut ini daftar penelitian terkait dapat dilihat pada tabel berikut :

Table 3 - Penelitian Terkait

No.	Nama dan Tahun	Judul	Kesimpulan
1.	Mukhammad Agus Arianto, 2016.	ANALISIS DAN PERANCANGAN REPRESENTATIONAL STATE TRANSFER (REST) WEB SERVICE SISTEM INFORMASI AKADEMIK STT TERPADU NURUL FIKRI MENGGUNAKAN YII FRAMEWORK	1. Dihasilkanya rancangan model *web service* sistem informasi akademik STT Terpadu Nurul Fikri berbasis teknologi REST menggunakan Yii Framework 2.0. 2. Rest *web service* yang dirancang dan dibuat dapat diuji dengan baik dengan menggunakan *tools* pengujian.
2.	Watik Adidaya, 2016.	ANALISIS DAN PERANCANGAN SISTEM INFORMASI AKADEMIK SEKOLAH TINGGI TEKNOLOGI TERPADU NURUL FIKRI MODUL ADMINISTRASI KEUANGAN BERBASIS *WEB* MENGGUNAKAN YII *FRAMEWORK*.	1. Aplikasi yang telah dibangun sudah sesuai dan memenuhi kebutuhan. Dan terbantu dengan adanya fitur penambahan pembayaran mahasiswa melalui *upload* data bank. 2. Aplikasi SIAK modul keuangan yang dibangun dengan MVC *framework* Yii, struktur programnya sudah menjadi lebih terorganisir dari aplikasi SIAK sebelumnya.
3.	I Wayan Gede Suma Wijaya.	PENERAPAN WEB SERVICE PADA APLIKASI SISTEM AKADEMIK PADA PLATFORM SISTEM OPERASI MOBILE ANDROID.	Metode *web service* berhasil diterapkan pada aplikasi sistem informasi akademik, dengan memanfaatkan JSON sebagai format pertukaran data, yang memungkinkan lintas platform tanpa tergantung pada jenis aplikasi yang digunakan di sisi *client*.
4.	Budi Santosa, 2008	ANALISA DAN PERANCANGAN WEB SERVICE UNTUK SISTEM INFORMASI UNIVERSITAS	XML *web service* menjadi komponen utama dalam mengembangakan sistem informasi akademik di unversitas.

5.	Sirojul Munir, 2016	Perancangan Sistem Informasi Akademik Berbasis Web menggunakan Framework MVC pada Sekolah Tinggi Teknologi Terpadu Nurul Fikri.	Penelitian ini membahas perancangan sistem SIAK STT-NF menggunakan metode unified process, dengan implementasi perancangan sistem berupa prototype aplikasi web menggunakan web framework dengan desain arsitektur Model View Controller (MVC). Penelitian ini diharapkan menghasilkan aplikasi sistem informasi akademik yang dapat memenuhi kebutuhan STT-NF

2.2.2 Posisi Penelitian

Table 4 - Posisi Penelitian

No.	Sistem Informasi Akademik Kampus	*Web service*	API	*Spring Framework*
1.	**Mukhammad Agus Arianto, 2016.** ANALISIS DAN PERANCANGAN REPRESENTATIONAL STATE TRANSFER (REST) WEB SERVICE SISTEM INFORMASI AKADEMIK STT TERPADU NURUL FIKRI MENGGUNAKAN YII FRAMEWORK.			
2.	**Watik Adidaya, 2016.** ANALISIS DAN PERANCANGAN SISTEM INFORMASI AKADEMIK SEKOLAH TINGGI TEKNOLOGI TERPADU NURUL FIKRI MODUL ADMINISTRASI KEUANGAN BERBASIS *WEB* MENGGUNAKAN YII *FRAMEWORK*.			
3.	**I Wayan Gede Suma Wijaya.** PENERAPAN WEB SERVICE PADA APLIKASI SISTEM AKADEMIK PADA PLATFORM SISTEM OPERASI MOBILE ANDROID.			
4.	**Budi Santosa, 2008** ANALISA DAN PERANCANGAN WEB SERVICE UNTUK SISTEM INFORMASI UNIVERSITAS			
5.	**Sirojul Munir, 2016**			

	PERANCANGAN SISTEM INFORMASI AKADEMIK BERBASIS WEB MENGGUNAKAN FRAMEWORK MVC PADA SEKOLAH TINGGI TEKNOLOGI TERPADU NURUL FIKRI			
6.	**Mohammad Akmaluddin Novianto, 2020.** **RANCANG BANGUN API *WEB SERVICE* MODUL KARTU HASIL STUDI (KHS) PADA SISTEM INFORMASI AKADEMIK STT NF MENGGUNAKAN RESTFUL-SPRING *FRAMEWORK***			