BAB IV

HASIL DAN PEMBAHASAN

4.1 Pengembangan Sistem

Pengembangan *Web Service* dari *website* program *Pengembangan Web api Pada Sistem Assesmen Dan Berbasis Tag Sebagai Pembantu Penyusunan Strategi Pembelajaran* berbasis *web* menggunakan Bahasa pemrograman Visual Studio C# dan *Web api Using Entity Framework. Web api Entity Framework* menggunakan metode *MVC* di mana terdapat tiga komponen yaitu *folder Models* yang akan menyimpan *file* ADO.Net yang akan digunakan untuk mengakses *database*, *folder View* yang akan menyimpan semua *file* yang berhubungan dengan *interface website*, dan *folder Controllers* yang akan menjadi penghubung antara *views* dan *models*.

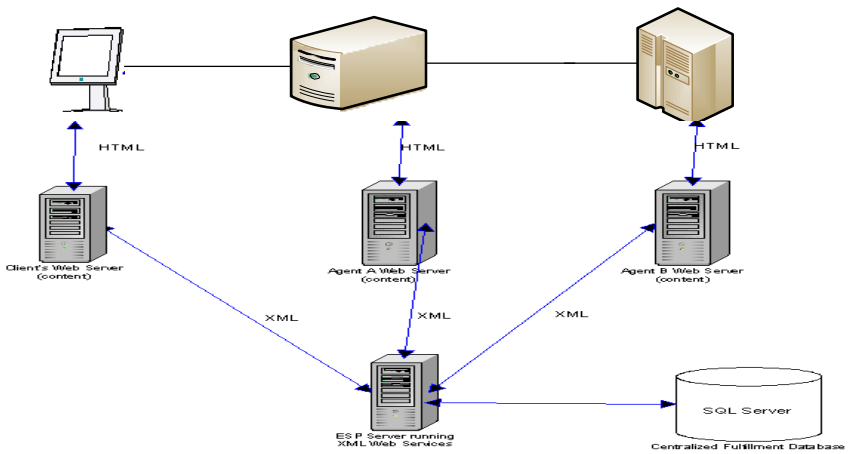
4.1.1 Tujuan Pengembangan Sistem

Adapun tujuan dari pengembangan *website* tersebut pada *Pengembangan Web api Pada Sistem Assesmen Dan Berbasis Tag Sebagai Pembantu Penyusunan Strategi Pembelajaran* ialah :

1. Untuk memudahkan user dalam penggunaan *web api* dan *toolsnya*.
2. Untuk membuat pengembangan *web services* pada *website* tersebut.
3. Untuk mengetahui hasil *web services* yang terdapat pada *website* tersebut.
4. Untuk mengetahui analisis dan pengujian *web services* yang di gunakan pada *website.*

4.1.3 Gambaran Umum Sistem

Pada *website ini*, bisa digambarkan sebagai media penggunaan sistem untuk para siswa dan guru sekolah yang menyediakan data pembuatan event ujian, soal ujian, hasil ujian, serta perekapan nilai-nilai untuk para siswa yang akan melakukan tes untuk masuk ke perguruan tinggi. Adapun fitur-fitur yang terdapat dalam *web services* tersebut yaitu guru yang dapat melakukan penginputan soal-soal ujian serta mata pelajaran yang akan di ujikan untuk para siswanya, guru dapat melihat hasil ujian siswanya, serta siswa juga dapat melihat hasil ujian yang telah di laksanakan dengan berbentuk program *website* dalam bentuk presentase, angka dan grafik. Semua data yang ditampilkan pada *website* telah tersimpan dalam *database*, dan *server*. Database server berfungsi sebagai layanan akses data-data yang terdapat pada isi *website* tersebut. Layanan apapun yang tersedia pada *website* tersebut dapat di akses dengan koneksi internet dengan menggunakan sistem *XML* messaging atau *JSON*. untuk melakukan akses data ke server *web* harus melewati *web services* yang fungsinya sebagai jembatan antara aplikasi *website* dengan *database server* untuk lebih jelasnya bisa dilhat pada gambar dibawah ini :



Gambar 4. 1 Gambaran Umum Sistem

Penjelasan yang ditunjukkan pada gambar 4.2 adalah :

Sebuah *website, web service,* dan *database server* yang saling berhubungan dalam melakukan layanan-layanan data sesuai *request.* Kemudian hasil *request* akan di tampilkan dengan *response* bentuk format data *JSON.*

1. **Aplikasi *Webssite***

   Aplikasi *Website* dalam hal ini merupakan analisis fungsional aplikasi untuk mengakses data dari database server melalui *web service*. Aplikasi ini dibangun oleh *Web api Entity Framework MVC* dari Visual Studio C# dan bekerja dengan memparsing data dari *web service* yang bertipe *JSON* untuk diolah pada *website* tersebut.

2. ***Web Service***

   Web Service pada penelitian ini berfungsi sebagai jembatan antara aplikasi *mobile* dengan data base server. Cara kerja *web service* ini yaitu dengan mengambil data dari database server dan kemudian mengkonversikan data tersebut ke format pertukaran data *JSON*.

3. **Protocol *HTTP***

   Protokol *HTTP* merupakan protokol l*api*san jaringan aplikasi yang digunakan untuk sistem informasi terdistribusi, kolaboratif dan menggunakan *hypermedia*. Dalam penelitian ini protocol *HTTP* digunakan sebagai protokol yang dapat mendistribusikan data dari database server.

4.2 Arsitektur Jaringan

Arsitektur jaringan yang dibangun dari *web api* ini berfungsi sebagai antarnuka penguna dengan sistem. Kemudian dengan Akses koneksi jaringan yang digunakan dari *web api* adalah jaringan IIS Server sebagai localhost yang terhubung ke *web service* yang bertujuan menghubungkan suatu *website* dengan server ke database untuk mengambil sebuah data.

4.3 Implementasi

Tujuan Implementasi sistem adalah untuk menjelaskan tentang penggunaan *web api* kepada *user* yang akan menggunakan sistem. Sehingga *user* tersebut dapat merespon apa yang ditampilkan di sistem dan memberikan masukan kepada pembuat sistem untuk melakukan perbaikan agar sistem lebih baik lagi.

4.3.1 Batasan Implementasi

Adapun batasan implementasi pada *web api* ini, diantaranya ialah :

1. Implementasi *website* ini hanya menampilkan *request* dan *response* Login dengan akses pengguna, mengambil hasil *response* yang akan di lanjutkan ke *postman* untuk mengambil sebuah access_token dari setiap pengguna yang login, kemudian melihat hasil keseluruhan data dengan format bentuk data *JSON*, selain itu menampilkan data guru dan data siswa dengan sebuah tampilan dashboard yang cukup memudahkan *users.*
2. Implementasi data pergerakan *web api* tersebut yang di ambil dari database server.

4.3.2 Implementasi Perangkat Keras

Prangkat Keras yang digunakan selama pembangunan *web api* ini memiliki spesifikasi sebagai berikut :

a. Perangkat Keras Laptop Dell.
b. Processor Intel Core i5 2.20 Ghz.
c. RAM 4 GB.
d. VGA 2.0 GB.
e. Hardisk 500 GB.

4.3.3 Implementasi Basis Data (Sintaks SQL)

Implementasi Basis Data diambil berdasarkan perancangan basis data yang dibuat sebelumnya.

1. Tb_Da*Tag*uru

Tabel 4. 1 Tabel Da*Tag*uru

```
create table Da Taguru (
Guru_Id nvarchar (128) not null,
primary key clustered (Guru_Id),
foreign key (Guru_Id) references[AspNetUsers](Id) on update
    cascade on delete cascade,
Nama_Guru varchar (50),
MP_Id int foreign key references MataPelajaran(MP_Id),
Jenis_kelamin char(1) constraint checkJenis_kelamin
    check(Jenis_kelamin in('L','P')),
Alamat varchar (255),
NIP varcahr (18),
Sekolah varchar (20));
```

Penjelasan Tabel 4.1 Data Guru :

Dari tabel diatas dijelaskan dalam pembuatan tabel data guru sebagai tempat untuk menyimpan data. Tabel diatas terdiri dari field dan record. Database diatas dibuat dengan menggunakan aplikasi desktop yaitu SQL Server 2014 Management.

2. Tb_DataSiswa

Tabel 4. 2 Tabel DataSiswa

```
create table DataSiswa (
Siswa_Id nvarchar (128) not null,
primary key clustered (Siswa_Id),
foreign key (Siswa_Id) references[AspNetUsers](Id) on update
    cascade on delete cascade,
Nama_Siswa varchar (50),
Jenis_kelamin varchar(1),
Sekolah varchar (30),
Alamat varchar (255),
NISN varchar (10), Kelas varchar(10));
```

Penjelasan Tabel 4.2 Data Siswa :

Dari tabel diatas dijelaskan dalam pembuatan tabel data siswa sebagai tempat untuk menyimpan data dan tempat dimana seorang developer dapat mengambil data tersebut melalui database yang telah dibuat. Tabel diatas terdiri dari field dan record. Tabel diatas dibuat dengan menggunakan aplikasi desktop yaitu SQL Server 2014 Management.

3. Tb_MataPelajaran

Tabel 4. 3 Tabel MataPelajaran

```
create table MataPelajaran (
MP_Id int identity(1,1) primary key,
Nama_MP varchar (15)
);
```

Penjelasan Tabel 4.3 Mata Pelajaran :

Dari tabel diatas dijelaskan dalam pembuatan tabel mata pelajaran sebagai tempat untuk menyimpan data dan tempat dimana seorang developer dapat mengambil data tersebut melalui database yang telah dibuat. Tabel diatas terdiri dari field dan record. Tabel diatas dibuat dengan menggunakan aplikasi desktop yaitu SQL Server 2014 Management.

4. Tb_*Tag*

Tabel 4. 4 Tabel *Tag*

```
create table Tag (
Tag_Id int identity(1,1) primary key,
MP_Id int foreign key references MataPelajaran(MP_Id) on update
cascade on delete cascade,
Tag varchar (20), Tanggal datetime );
```

Penjelasan Tabel 4.4 *Tag* :

Dari tabel diatas dijelaskan dalam pembuatan tabel *Tag* sebagai tempat untuk menyimpan data dan tempat dimana seorang developer dapat mengambil data tersebut melalui database yang telah dibuat.

Tabel diatas terdiri dari field dan record. Tabel diatas dibuat dengan menggunakan aplikasi desktop yaitu SQL Server 2014 Management.

5. Tb_Soal

Tabel 4. 5 Tabel Soal

```
create table Soal (
Soal_Id int identity(1,1) primary key,
Createdby nvarchar (128) not null,
foreign key (Createdby) references[Da Tag uru](Guru_Id) on update cascade on delete cascade,
Tag_Id int foreign key references Tag(Tag_Id) on update cascade on delete cascade,
Soal varchar (max),
A varchar (max),
B varchar (max),
C varchar (max),
D varchar (max),
E varchar (max),
Jawaban varchar (1),
Tanggal datetime (Date) ,
);
```

Penjelasan Tabel 4.5 Soal :

Dari tabel diatas dijelaskan dalam pembuatan tabel soal sebagai tempat untuk menyimpan data dan tempat dimana seorang developer dapat mengambil data tersebut melalui database yang telah dibuat. Tabel diatas terdiri dari field dan record. Tabel diatas dibuat dengan menggunakan aplikasi desktop yaitu SQL Server 2014 Management.

6. Tb_Nilai

Tabel 4. 6 Tabel Nilai

```
create table Nilai (
Nilai_Id int identity(1,1) primary key,
EventUjian_Id int foreign key references EventUjian(EventUjian_Id) on update cascade on delete cascade,
Jawaban_Siswa char (1),
Nilai int
);
```

Penjelasan Tabel 4.6 Nilai :

Dari tabel diatas dijelaskan dalam pembuatan tabel nilai sebagai tempat untuk menyimpan data dan tempat dimana seorang developer dapat mengambil data tersebut melalui database yang telah dibuat. Tabel diatas terdiri dari field dan record. Tabel diatas dibuat dengan menggunakan aplikasi desktop yaitu SQL Server 2014 Management.

4.4 Fungsi – Fungsi *Service* Pada Aplikasi *Web api*

Pada penjelasan kali ini, saya akan menjelaskan beberapa fungsi-fungsi *web service* atau disebut sebagai *web api* pada aplikasi *Web api Tag*BasedExam. Fungsi *web service* yang terdapat didalamnya sebagai berikut :

a. *API*_Da*Tag*urus*Controller*.cs
b. *API*_DataSiswas*Controller*.cs
c. *API*_EventUjians*Controller*.cs
d. *API*_MataPelajarans*Controller*.cs
e. *API*_Nilais*Controller*.cs
f. *API*_Soals*Controller*.cs
g. *API_Tags*Controller.cs

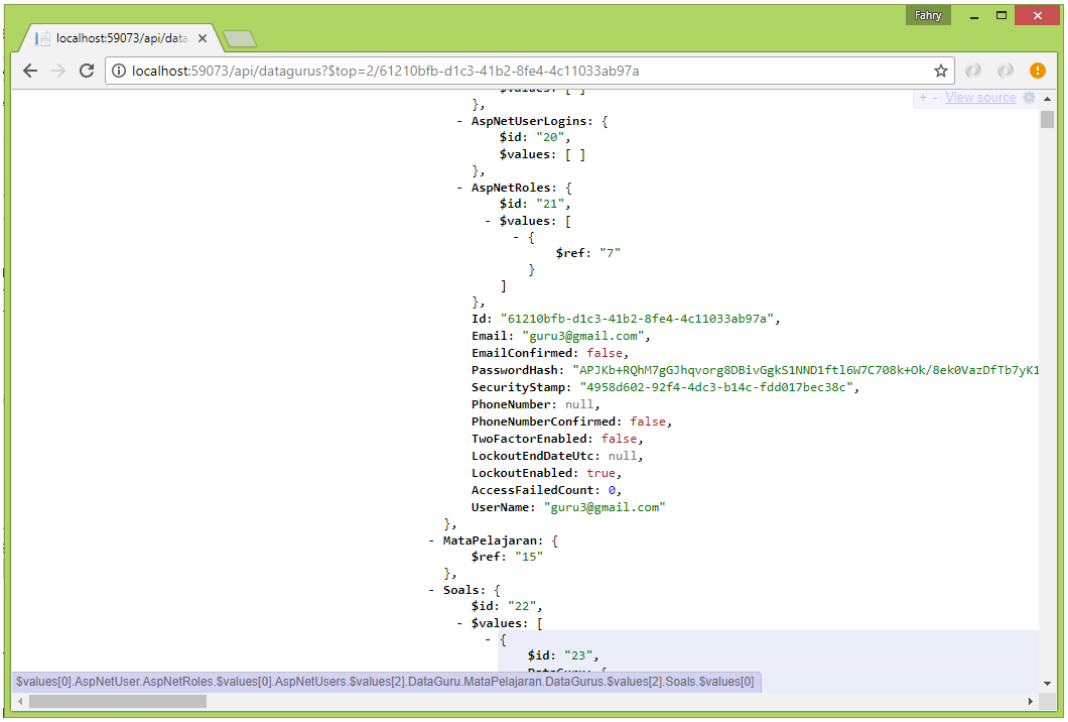
h. Da*Tag*urus*Controller*.cs
i. DataSiswas*Controller*.cs
j. EventUjians*Controller*.cs
k. MataPelajarans*Controller*.cs
l. Nilais*Controller*.cs
m. Soals*Controller*.cs
n. *Tags**Controller*.cs
o. *Tag*Data*Service*.cs
p. *Tag*DataPoint.cs
q. *WebApi*Config.cs
r. Startup.Auth.cs
s. Global.asax

```
    // WebAPI_Routes - DataGuru
    config.Routes.MapHttpRoute(
    name: "DataGurus",
    routeTemplate: "api/datagurus/{id}",
    defaults: new { controller = "datagurus", id = RouteParameter.Optional },
    constraints: new { id = "/d+" }
);
    //WebAPI_Routes - DataSiswa
    config.Routes.MapHttpRoute(
    name: "DataSiswas",
    routeTemplate: "api/datasiswas/{id}",
    defaults: new { controller = "datasiswas", id = RouteParameter.Optional },
    constraints: new { id = "/d+" }
);

    //WebAPI_Routes - MataPelajaran
    config.Routes.MapHttpRoute(
    name: "MataPelajarans",
    routeTemplate: "api/matapelajarans/{id}",
    defaults: new { controller = "matapelajarans", id = RouteParameter.Optional },
    constraints: new { id = "/d+" }
);

    //WebAPI_Routes - Nilai
    config.Routes.MapHttpRoute(
    name: "Nilais",
```

Gambar 4. 2 *Routing and Action Selection In Web api*

Pada gambar 4.3 memperlihatkan *coding* yang digunakan untuk menentukan nama route atau parameter *query* rute pertama di tabel rute yang sesuai dengan URI dengan tujuan untuk menampilkan data *API*. Jika nama route tersebut menemukan kecocokan dengan URI, maka akan menampilan hasil untuk setiap placeholder. Kuncinya adalah nama

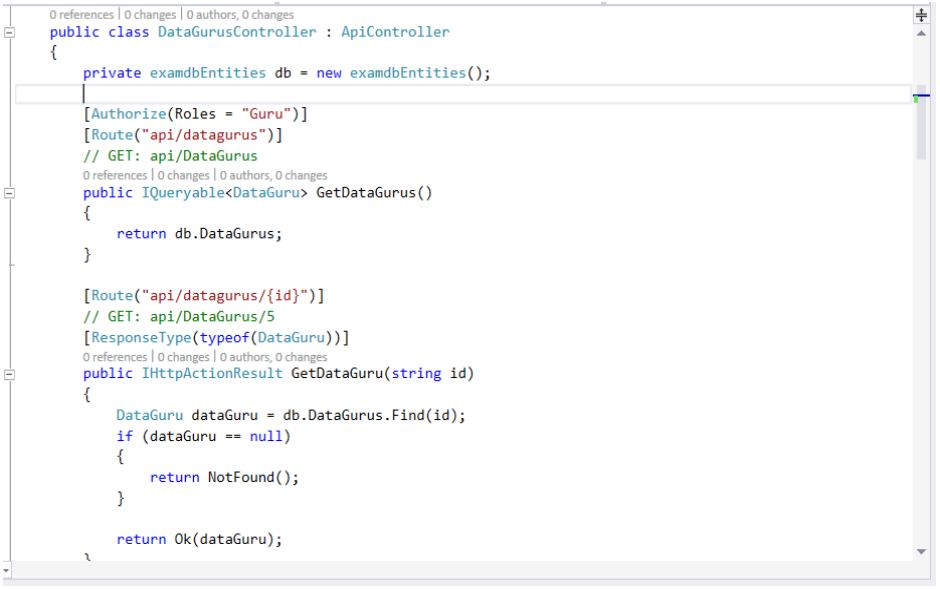
placeholder, Hasilnya diambil dari jalur URI atau dari default, dan data tersebut telah tersimpan di I*Http*RouteData.



Gambar 4. 3 *Route API* Guru

Pada gambar 4.4 memperlihatkan *coding* yang digunakan untuk menampilkan hasil route atau parameter *query* rute yang sesuai dengan URI dengan tujuan untuk menampilkan data *API*. Untuk jalur URI "*api*/da*Tag*urus?$top=2/id", hasil rute akan berisi :

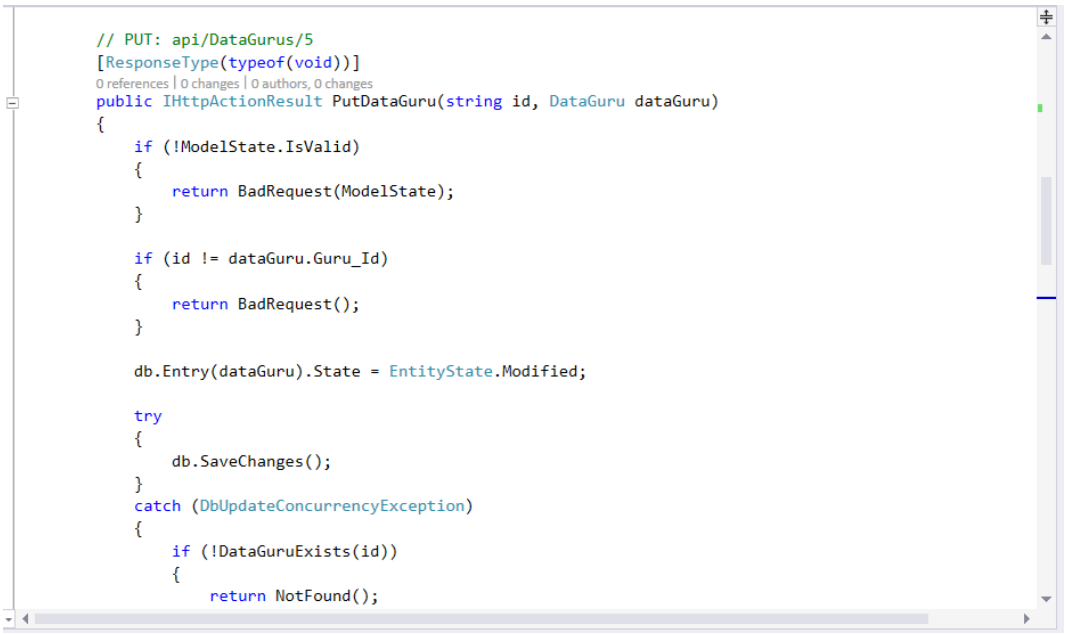
- *Controller* : “Da*Tag*urus”
- Id :“?$top=2/61210bfb-d1c3-41b2-8fe4 4c11033ab97a”
- Filter : *api*/*api*_da*Tag*urus?$expand=Soals&$expand=StandarNilai &MataPelajaran
- Select by : *api*/*api*_da*Tag*urus?$select=Guru_Id, Nama_Guru



```
0 references | 0 changes | 0 authors, 0 changes
public class DataGurusController : ApiController
{
    private examdbEntities db = new examdbEntities();

    [Authorize(Roles = "Guru")]
    [Route("api/datagurus")]
    // GET: api/DataGurus
    0 references | 0 changes | 0 authors, 0 changes
    public IQueryable<DataGuru> GetDataGurus()
    {
        return db.DataGurus;
    }

    [Route("api/datagurus/{id}")]
    // GET: api/DataGurus/5
    [ResponseType(typeof(DataGuru))]
    0 references | 0 changes | 0 authors, 0 changes
    public IHttpActionResult GetDataGuru(string id)
    {
        DataGuru dataGuru = db.DataGurus.Find(id);
        if (dataGuru == null)
        {
            return NotFound();
        }

        return Ok(dataGuru);
    }
```

Gambar 4. 4 ***Coding Get API* Data Guru**

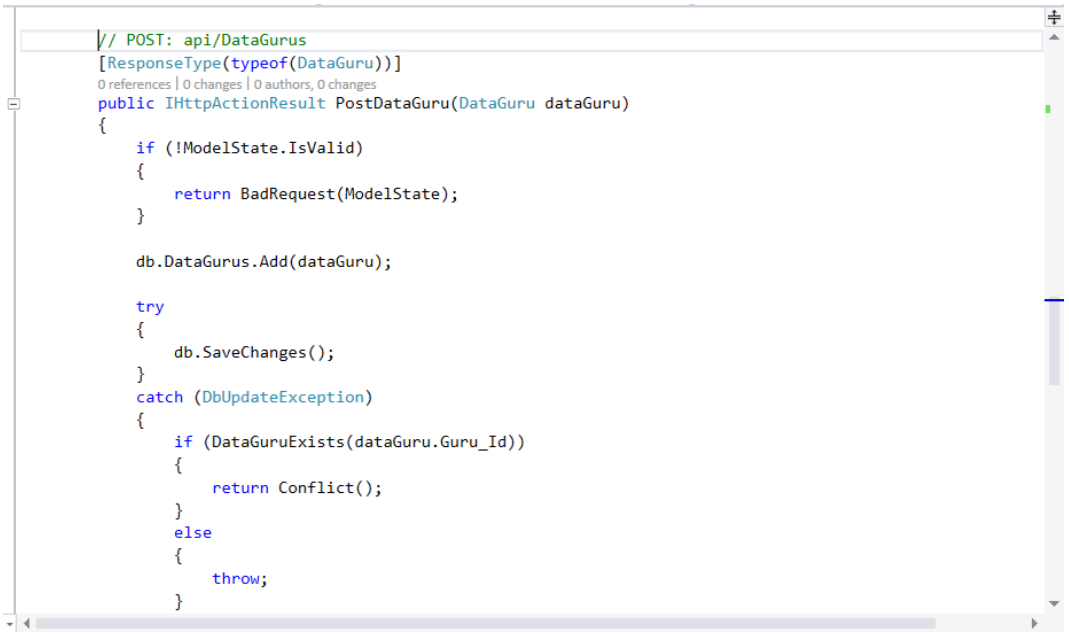
Pada gambar 4.5 memperlihatkan *coding* yang digunakan untuk mengambil atau membaca data request route *API* atau parameter *query* dari data guru yang sesuai dengan URI dengan tujuan untuk menampilkan data *API*.



```
// PUT: api/DataGurus/5
[ResponseType(typeof(void))]
0 references | 0 changes | 0 authors, 0 changes
public IHttpActionResult PutDataGuru(string id, DataGuru dataGuru)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    if (id != dataGuru.Guru_Id)
    {
        return BadRequest();
    }

    db.Entry(dataGuru).State = EntityState.Modified;

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!DataGuruExists(id))
        {
            return NotFound();
```

Gambar 4. 5 *Coding Put API* Data Guru

Pada gambar 4.6 memperlihatkan *coding* yang digunakan untuk method dalam kelompok *HTTP* untuk mengupdate item/resource yang telah ada. (balik ke Gambar 4.5).



```
// POST: api/DataGurus
[ResponseType(typeof(DataGuru))]
0 references | 0 changes | 0 authors, 0 changes
public IHttpActionResult PostDataGuru(DataGuru dataGuru)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    db.DataGurus.Add(dataGuru);

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateException)
    {
        if (DataGuruExists(dataGuru.Guru_Id))
        {
            return Conflict();
        }
        else
        {
            throw;
        }
```

Gambar 4. 6 *Coding Post API* Data Guru

Pada gambar 4.7 memperlihatkan *coding* yang digunakan untuk method dalam mengembalikan keluaran/output dari *request URI* /*api*/da*Tag*urus/{id} yang disebut procedure.

```
0 references | 0 changes | 0 authors, 0 changes
public class DataSiswasController : ApiController
{
    private examdbEntities db = new examdbEntities();

    //[Authorize(Roles ="Siswa")]
    [Route("api/datasiswas")]
    // GET: api/DataSiswas
    0 references | 0 changes | 0 authors, 0 changes
    public IQueryable<DataSiswa> GetDataSiswas()
    {
        return db.DataSiswas;
    }


    // GET: api/DataSiswas/5
    [ResponseType(typeof(DataSiswa))]
    0 references | 0 changes | 0 authors, 0 changes
    public IHttpActionResult GetDataSiswa(string id)
    {
        DataSiswa dataSiswa = db.DataSiswas.Find(id);
        if (dataSiswa == null)
        {
            return NotFound();
        }

        return Ok(dataSiswa);
    }
```

Gambar 4. 7 ***Coding Get API* Data Siswa**

Pada gambar 4.8 memperlihatkan *coding* yang digunakan untuk mengambil atau membaca data request route *API* atau parameter *query* dari data siswa yang sesuai dengan URI dengan tujuan untuk menampilkan data *API*.

```
    // PUT: api/DataSiswas/5
    [ResponseType(typeof(void))]
    0 references | 0 changes | 0 authors, 0 changes
    public IHttpActionResult PutDataSiswa(string id, DataSiswa dataSiswa)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        if (id != dataSiswa.Siswa_Id)
        {
            return BadRequest();
        }

        db.Entry(dataSiswa).State = EntityState.Modified;

        try
        {
            db.SaveChanges();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!DataSiswaExists(id))
            {
                return NotFound();
```

Gambar 4. 8 *Coding Put API* Data Siswa

Pada gambar 4.9 memperlihatkan *coding* yang digunakan untuk method dalam kelompok *HTTP* untuk mengupdate item/resource yang telah ada. (balik ke Gambar 4.8).

```
[Route("api/datasiswas/{id}")]
// POST: api/DataSiswas
[ResponseType(typeof(DataSiswa))]
0 references | 0 changes | 0 authors, 0 changes
public IHttpActionResult PostDataSiswa(DataSiswa dataSiswa)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    db.DataSiswas.Add(dataSiswa);

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateException)
    {
        if (DataSiswaExists(dataSiswa.Siswa_Id))
        {
            return Conflict();
        }
        else
        {
            throw;
```

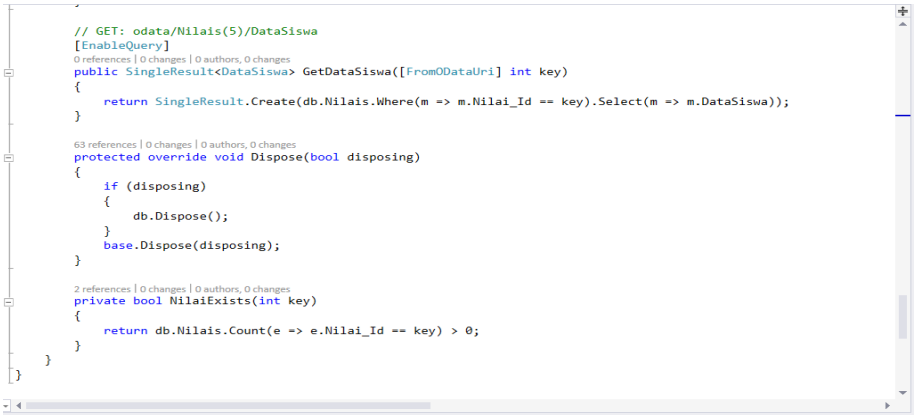
Gambar 4. 9 *Coding Post API* Data Siswa

Pada gambar 4.10 memperlihatkan *coding* yang digunakan untuk method dalam mengembalikan keluaran/output dari*request URI* /*api*/datasiswas/{id}/ yang disebut procedure.

Gambar 4. 10 *Route API* Data Siswa

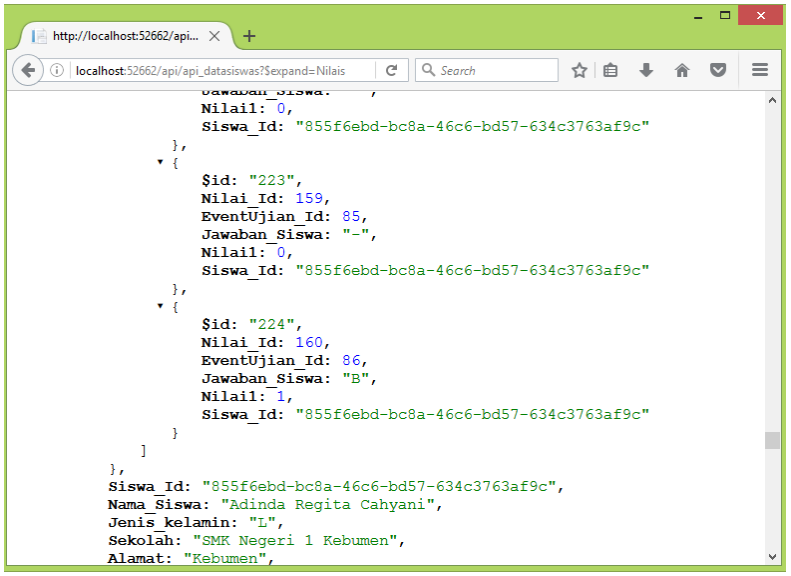
Pada gambar 4.11 memperlihatkan *coding* yang digunakan untuk menampilkan hasil route atau parameter *query* rute yang sesuai dengan URI dengan tujuan untuk menampilkan data *API*. Untuk jalur URI "*api*/datasiswas?$top=2/id" dan filter datasiswa yang dapat melihat nilai ujian hasil rute akan berisi :

- *Controller* : "DataSiswas"
- Id :"?$top=2/ 0c10a52d-e15e-4d16-b66a-bf8dee1cb550"
- Filter : *api*/*api*_datasiswas?$expand=Nilais
- Select by : *api*/*api*_datasiswas?$select=Nama_Siswa, Kelas, Sekolah



Gambar 4. 11 *Coding* Nilai Per Siswa

Pada gambar 4.12 memperlihatkan *coding* yang digunakan untuk menampilkan hasil nilai dengan per siswa yang termasuk dalam data siswa.



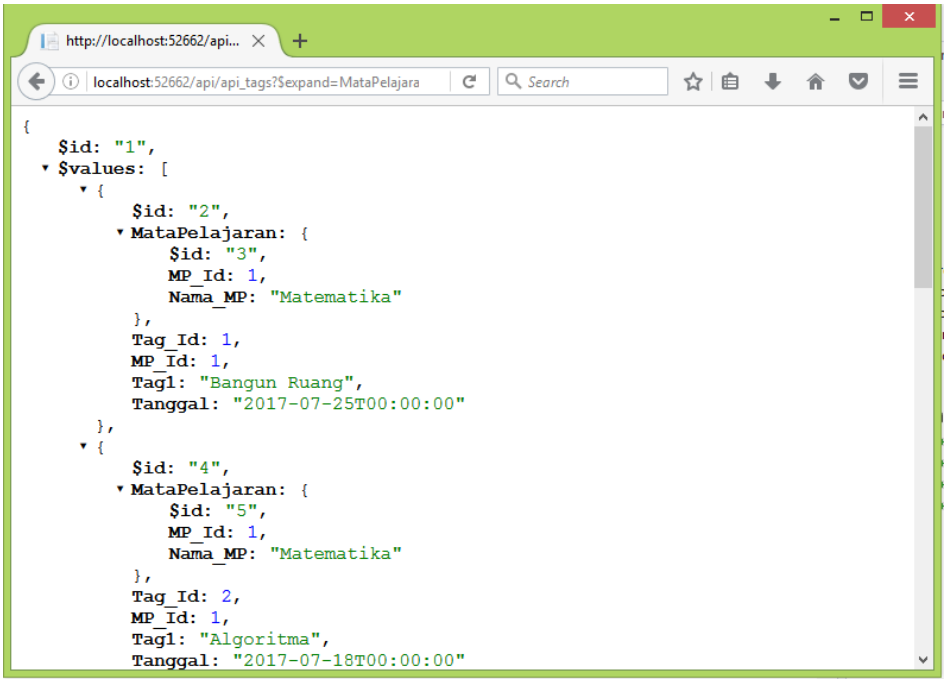
Gambar 4. 12 Hasil *Filter* Nilai Per Siswa

Pada gambar 4.13 memperlihatkan hasil *request* nilai dengan data siswa, jawaban siswa, serta id siswa ke halaman *website* yang tercantum dengan hasil nilai tersebut.

```
        return StatusCode(HttpStatusCode.NoContent);
    }

    // GET: odata/Tags(5)/MataPelajaran
    [EnableQuery]
    0 references | 0 changes | 0 authors, 0 changes
    public SingleResult<MataPelajaran> GetMataPelajaran([FromODataUri] int key)
    {
        return SingleResult.Create(db.Tags.Where(m => m.Tag_Id == key).Select(m => m.MataPelajaran));
    }

    // GET: odata/Tags(5)/Soals
    [EnableQuery]
    0 references | 0 changes | 0 authors, 0 changes
    public IQueryable<Soal> GetSoals([FromODataUri] int key)
    {
        return db.Tags.Where(m => m.Tag_Id == key).SelectMany(m => m.Soals);
    }

    63 references | 0 changes | 0 authors, 0 changes
    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            db.Dispose();
        }
        base.Dispose(disposing);
    }
```

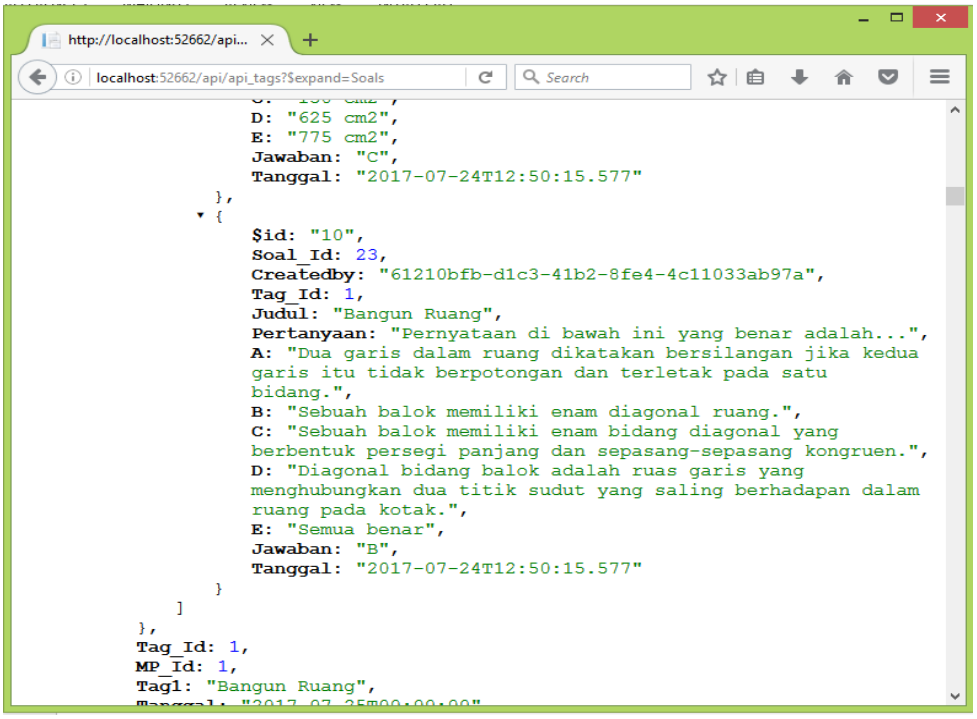
Gambar 4. 13 *Coding Tag* Per Mata Pelajaran dan Soal

Pada gambar 4.14 memperlihatkan *coding* untuk menampilkan *tag* per mata pelajaran dan *tag* per soal.



Gambar 4. 14 Hasil *Tag* Per Mata Pelajaran

Pada gambar 4.15 memperlihatkan hasil *request* dari *tag* per mata pelajaran ke sebuah halaman *website.*



Gambar 4. 15 Hasil *Tag* Per Soal

Pada gambar 4.16 memperlihatkan *request* yang digunakan untuk menampilkan hasil *tag* per soal ke halaman *website*.

```
namespace FinalSkripsi.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class TagBySiswaController : ApiController
    {
        private examdbEntities db = new examdbEntities();

        [Route("api/tagbysiswa")]
        [HttpGet]
        [EnableQuery]
        0 references | 0 changes | 0 authors, 0 changes
        public List<TagDataService.TagSiswa> getTagBySiswa()
        {
            var tag = new List<TagDataService.TagSiswa>();
            tag = (from ds in db.DataSiswas
                        join n in db.Nilais on ds.Siswa_Id equals n.Siswa_Id
                        join eu in db.EventUjians on n.EventUjian_Id equals eu.EventUjian_Id
                        join s in db.Soals on eu.Soal_Id equals s.Soal_Id
                        join t in db.Tags on s.Tag_Id equals t.Tag_Id
                        select new TagDataService.TagSiswa
                        {
                            Tag_Id = t.Tag_Id,
                            Tag1 = t.Tag1,
                            Nama_Siswa = ds.Nama_Siswa,
                            Siswa_Id = ds.Siswa_Id
                        }).ToList();

            return tag;
        }
```

string DataSiswa.Nama_Siswa { get; set; }

Gambar 4. 16 *Coding Tag* By Siswa

Pada gambar 4.17 memperlihatkan *coding* yang digunakan untuk menampilkan hasil *tag* per siswa yang termasuk dalam data siswa.

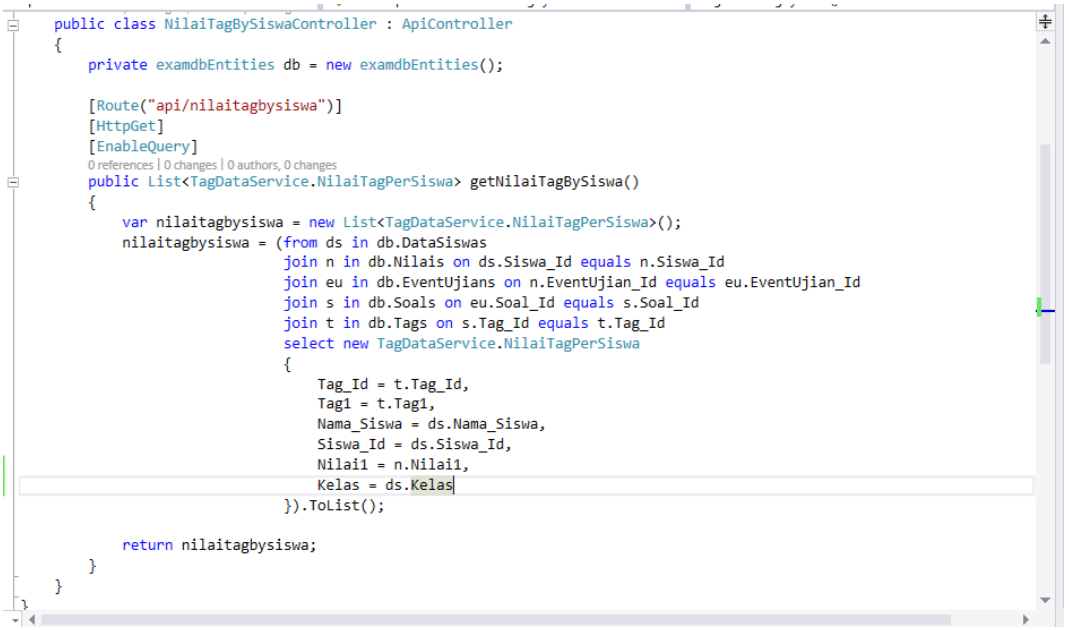
localhost:52662/api/tagbysiswa/getTagBySiswa


```
{
  $id: "1",
  $values: [
    {
      $id: "2",
      Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
      Nama_Siswa: "Ameylia Inriaswari",
      Tag_Id: 1,
      Tag1: "Bangun Ruang"
    },
    {
      $id: "3",
      Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
      Nama_Siswa: "Ameylia Inriaswari",
      Tag_Id: 3,
      Tag1: "Aljabar"
    },
    {
      $id: "4",
      Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
      Nama_Siswa: "Ameylia Inriaswari",
      Tag_Id: 4,
      Tag1: "Trigonometri"
    },
    {
      $id: "5",
      Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
      Nama_Siswa: "Ameylia Inriaswari",
      Tag_Id: 3,
      Tag1: "Aljabar"
    },
    {
      $id: "6",
      Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
      Nama_Siswa: "Ameylia Inriaswari",
```


$values

Gambar 4. 17 Hasil *Tag* By Siswa

Pada gambar 4.18 memperlihatkan *request* yang digunakan untuk menampilkan hasil *tag* per siswa yang termasuk dalam data siswa.



```
public class NilaiTagBySiswaController : ApiController
{
    private examdbEntities db = new examdbEntities();

    [Route("api/nilaitagbysiswa")]
    [HttpGet]
    [EnableQuery]
    0 references | 0 changes | 0 authors, 0 changes
    public List<TagDataService.NilaiTagPerSiswa> getNilaiTagBySiswa()
    {
        var nilaitagbysiswa = new List<TagDataService.NilaiTagPerSiswa>();
        nilaitagbysiswa = (from ds in db.DataSiswas
                           join n in db.Nilais on ds.Siswa_Id equals n.Siswa_Id
                           join eu in db.EventUjians on n.EventUjian_Id equals eu.EventUjian_Id
                           join s in db.Soals on eu.Soal_Id equals s.Soal_Id
                           join t in db.Tags on s.Tag_Id equals t.Tag_Id
                           select new TagDataService.NilaiTagPerSiswa
                           {
                               Tag_Id = t.Tag_Id,
                               Tag1 = t.Tag1,
                               Nama_Siswa = ds.Nama_Siswa,
                               Siswa_Id = ds.Siswa_Id,
                               Nilai1 = n.Nilai1,
                               Kelas = ds.Kelas
                           }).ToList();

        return nilaitagbysiswa;
    }
}
```

Gambar 4. 18 *Coding* Nilai *Tag* By Siswa

Pada gambar 4.19 memperlihatkan *coding* yang digunakan untuk menampilkan hasil nilai *tag* per siswa yang termasuk dalam data siswa.

```
localhost:52662/api/nilaitagbysiswa/getNilaiTagBySiswa

{
   $id: "1",
  - $values: [
     - {
          $id: "2",
          Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
          Nama_Siswa: "Ameylia Inriaswari",
          Tag_Id: 1,
          Tag1: "Bangun Ruang",
          Nilai1: 0,
          Kelas: "XII IPA 1"
       },
     - {
          $id: "3",
          Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
          Nama_Siswa: "Ameylia Inriaswari",
          Tag_Id: 3,
          Tag1: "Aljabar",
          Nilai1: 1,
          Kelas: "XII IPA 1"
       },
     - {
          $id: "4",
          Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
          Nama_Siswa: "Ameylia Inriaswari",
          Tag_Id: 4,
          Tag1: "Trigonometri",
          Nilai1: 0,
          Kelas: "XII IPA 1"
       },
     - {
          $id: "5",
          Siswa_Id: "0c10a52d-e15e-4d16-b66a-bf8dee1cb550",
          Nama_Siswa: "Ameylia Inriaswari",
          Tag_Id: 3,
```

Gambar 4. 19 Hasil Nilai *Tag* By Siswa

Pada gambar 4.20 memperlihatkan *request* yang digunakan untuk menampilkan hasil nilai *tag* per siswa yang termasuk dalam data siswa.

4.5 Pengujian

Pengujian merupakan bagian penting dalam siklus pembuatan atau pengembangan perangkat lunak. Pengujian dilakukan untuk menjamin kualitas dan juga mengetahui kelemahan dari perangkat lunak. Tujuan dari pengujian perangkat lunak ini adalah untuk menjamin bahwa perangkat lunak yang dibangun memiliki kualitas dan dapat diandalkan. Pengujian perangkat lunak ini menggunakan metode pengujian *Postman* dan *Unit Test*. Pengujian *Postman* Digunakan untuk menguji suatu akses *web service* dengan *access_Token* setiap pengguna yang login dari sistem *Web api* perangkat lunak yang dirancang, sedangkan metode pengujian *Unit Test* digunakan untuk menguji fungsi-fungsi khusus dari aplikasi yang di rancang.

4.5.1 Rencana Pengujian

Rencana Pengujian adalah pengujian terhadap fungsi-fungsi yang ada didalam sistem, apakah fungsional dari aplikasi tersebut berfungsi sesuai yang di harapkan atau tidak. Berikut ini tabel rencana pengujian dari sistem yang di bangun.

Tabel 4. 7 Rencana Pengujian

Rencana Pengujian **Aplikasi *Web api Tag* Based Exam**		
Item Uji	**Detail Uji**	**Jenis Uji**
Api Login Data Guru	Data *Api* Sukses	Unit Test

Api Login Data Siswa	Data *Api* Sukses	Unit Test
Halaman Data Guru	Menerima response	Unit Test
Halaman Data Siswa	Menerima response	Unit Test
Api Guru_Id	Menerima value sesuai Id yang di request	Unit Test
Api Siswa_Id	Menerima value sesuai Id yang di request	Unit Test
Data Guru	Menerima request *API* data guru	Unit Test
Data Siswa	Menerima request *API* data siswa	Unit Test
Nilai	Menerima response *API* nilai	Unit Test
Mata Pelajaran	Menerima response *API* mata pelajaran	Unit Test
Tag	Menampilkan hasil *API Tag*	Unit Test
Ujian	Menampilkan hasil *API* Ujian	Unit Test
Tag By Mata Pelajaran	Menampilkan hasil *API Tag* Per Mata Pelajaran	Unit Test
Tag By Soal	Menampilkan hasil *API Tag* Per Soal	Unit Test
Nilai *Tag* By Siswa	Menampilkan hasil *API* Nilai Per Siswa	Unit Test
Tag By Siswa	Menampilkan hasil *API Tag* Siswa	Unit Test

4.5.2 Kasus dan Hasil Pengujian

Hasil Pengujian **Aplikasi *Web api Tag* Based Exam**			
Item Uji	**Detail Uji**	**Hasil yang diharapkan**	**Ket**
Api Login Data Guru	Data *Api* Sukses	Menampilkan sebuah *web api*	Berhasil

Api Login Data Siswa	Data *Api* Sukses	Menampilkan sebuah *web api*	Berhasil
Halaman Data Guru	Menerima repsonse	Menampilkan halaman Da*Tag*uru	Berhasil
Halaman Data Siswa	Menerima repsonse	Menampilkan halaman DataSiswa	Berhasil
Api Guru_Id	Menerima value sesuai Id yang di request	Menampilan *access_token* Guru	Berhasil
Api Siswa_Id	Menerima value sesuai Id yang di request	Menampilan *access_token* Siswa	Berhasil
Data Guru	Menerima request *API* Data Guru	Menampilkan hasil request	Berhasil
Data Siswa	Menerima request *API* Data Siswa	Menampilkan hasil request	Berhasil
Nilai	Menerima response *API* Nilai	Menampilkan hasil response	Berhasil
Mata Pelajaran	Menerima response *API* Mata Pelajaran	Menampilkan hasil response	Berhasil
Tag	Menampilkan hasil *API Tag*	Menampilkan response	Berhasil
Ujian	Menampilkan hasil *API* Ujian	Menampilkan response	Berhasil
Tag By Mata Pelajaran	Menampilkan hasil *Tag* Per Mata Pelajaran	Menampilkan response	Berhasil
Tag By Soal	Menampilkan hasil *Tag* Per Soal	Menampilkan response	Berhasil
Nilai *Tag* By Siswa	Menampilkan hasil nilai *Tag* by siswa	Menampilkan response	Berhasil
Tag By Siswa	Menampilkan hasil *Tag* by siswa	Menampilkan response	Berhasil

4.5.3 Kesimpulan Hasil Pengujian

Berdasarkan hasil pengujian kasus uji sampel diatas dengan menggunakan unit test maka dapat disimpulkan bahwa sistem bebas kesalahan sintaks dan secara

fungsional mengeluarkan hasil yang sesuai dengan yang diharapkan yaitu bermanfaat bagi user dalam memberikan informasi ke *web services* (*API*). Namun tidak menutup kemungkinan dapat terjadi kesalahan suatu saat pada saat aplikasi digunakan, sehingga membutuhkan proses maintenance untuk lebih mengetahui kekurangan dari aplikasi.