Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Perpustakaan Univeristas Multimedia Nusantara

BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Manajemen pengembangan proyek piranti lunak pada Oval menggunakan *framework agile scrum*. *Scrum* adalah sebuah *framework* yang membantu proses perkembangan dan pengelolaan produk yang kompleks guna. Setiap pagi pada jam 10,00 WIB sebelum memulai kerja akan diadakan *Standup Meeting* yang bertujuan untuk mengkoordinasi daftar *jobdesc* antar anggota tim Oval, setiap anggota akan menceritakan pekerjaan yang telah dibuat sebelumnya dan apa yang akan dikerjakan pada hari itu, dan akan mendiskusikan dengan anggota lain jika terdapat sebuah masalah dalam pekerjaan. Kemudian terdapat rapat mingguan yang dijadwalkan rutin setiap hari Jumat yang dinamakan *Grooming*, rapat ini bertujuan untuk membahas proses sebuah *Sprint*. *Sprint* adalah sebuah bagian dari keseluruhan *project*, biasanya dalam sebuah *Sprint* setiap orang akan diberikan beberapa *task* untuk diselesaikan dalam jangka waktu 1 *Sprint* yaitu dua minggu atau sepuluh hari kerja.

Kegiatan kerja magang dilakukan dengan pengawasan dan bimbingan dari M. Afdal Wahyu yang berperan sebagai *Senior Backend Engineer* tim Oval dan mentor penulis. Mentor bertugas untuk memberikan tugas dalam tiap *Sprint* bagi penulis dan berkewajiban untuk membimbing dan mengajarkan tata cara pengerjaan proyek serta penulisan dokumentasi *source code* bagi penulis. Di setiap akhir *Sprint* akan ada penilaian dari mentor, dan juga akan terdapat *one-on-one discussion* antara mentor dan penulis. Komunikasi antar tim Oval Kompas Gramedia juga dibantu dengan *software* bernama *Slack* yang menjadi tempat dimana diumumkan jika terdapat sebuah *meeting* dan tempat dimana anggota tim Oval saling berbagi pendapat tentang kerjaan di kantor.

3.2 Tugas yang dilakukan

Terdapat beberapa tugas yang diberikan selama proses kerja magang, yaitu melakukan *integration testing* terhadap *API endpoint* untuk aplikasi *MyValue* dengan menggunakan *Cypress* kemudian membuat dua buah *micro-service* yaitu, sebuah *GraphQL Server* berdasarkan beberapa *API endpoint MyValue*, dan *gRPC Server* berdasarkan *API Endpoint MyValue membership*.

Proses kerja menggnakan perangkat Github sebagai sarana *version control project* dan Microsoft Azure sebagai *platform* untuk memudahkan proses validasi *push/pull request.* Setiap *pull request* yang dimasukkan ke *branch master* wajib melalui *development branch* dahulu, yang kemudian akan dilakukan testing sebelum di *merge* ke *branch master*. Berikut realisasi dari proses kerja magang.

Tabel 3.1 Realisasi Kerja Magang

Minggu	Pekerjaan yang Dilakukan
1	Instalasi *Software NodeJS, Github, Microsoft Azure, Docker* dan Mempelajari *Integration test* menggunakan Cypress
2	Melakukan *Integration Test* menggunakan *Cypress* terhadap backend *service submodules*
3	Melakukan *Integration Test* menggunakan *Cypress* terhadap backend *service submodules*
4	Mempelajari *gRPC* dan *GoLang* dan mempelajari pembuatan *client-server gRPC* menggunakan *GoLang*
5	Pembuatan *Client - Server gRPC* menggunakan *GoLang*, *Javascript*, dan *Python*
6	Mempelajari *GraphQL*, dan membuat struktur *GraphQL server* sederhana
7	Memulai pembuatan *GraphQL server* berdasarkan *API endpoint MyValue*

Tabel 3.1 Realisasi Keja Magang (Lanjutan)

Minggu	Pekerjaan yang Dilakukan
8	Membuat *gRPC service* yang akan berperan sebagai penghubung *GraphQL* dengan *API endpoint membership MyValue*
9	Membuat gRPC service yang akan berperan sebagai penghubung *GraphQL* dengan *API endpoint membership MyValue*
10	*Refactor Code*, dan mendokumentasikan *Server GraphQL* dan *Server gRPC server*

3.3 Uraian Pelakasanaan Kerja Magang

Proses pelaksanaan kerja magang dibagi menjadi tiga bagian yaitu proses pelaksanaan, kendala yang ditemukan, dan solusi atas kendala yang ditemukan.

3.3.1 Proses Pelaksanaan

Proses pelaksanaan pengembangan *backend service* MyValue pada Oval Kompas Gramedia membutuhkan perangkat lunak dan perangkat keras. Perangkat lunak yang digunakan dalam pengembangan *backend service* ini adalah sebagai berikut.

1. Visual Studio Code versi 1.4
2. NodeJS versi 12.7.0
3. Python versi 3.7.4
4. GoLang versi Go 1.13
5. Docker versi 19.03.8
6. Cypress versi 3.4.1
7. gRPC versi 1.28.1
8. GraphQL 15.0.0
9. Github versi 2.17
10. Slack versi 4.0.0
11. Microsoft Azure DevKit

12. Firefox v68.0.2 (64 bit) dan Google Chrome v75.0.3770.142
13. Operating System Ubuntu 18.04.2 (64 bit)

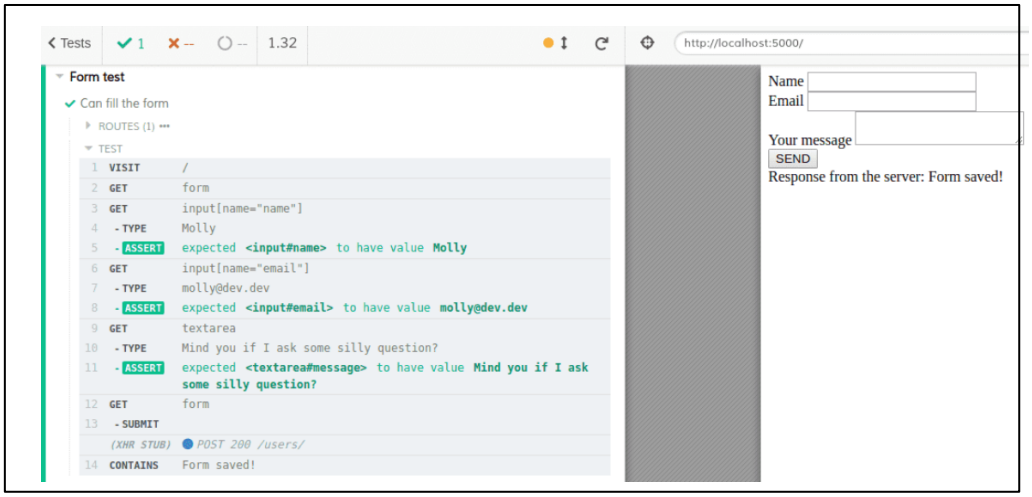
Perangkat keras yang digunakan dalam pengembangan *backend service* MyValue pada Oval Kompas Gramedia adalah Lenovo IdeaPad G40-80 6AID dengan spesifikasi sebagai berikut.

1. Prosesor Intel Core i3 4005U
2. RAM 8 GB
3. HD Graphics 5500 (GT2)

Proses pengerjaan selama pengembangan dibagi menjadi 3 bagian, yaitu melakukan *integration test* terhadap *backend service,* merancang *GraphQL server schema, dan* membangun *client-server GRPC* seperti yang sudah didaftarkan pada table realisasi kerja magang.

A. Integration Test

Integration Test adalah sebuah metode *testing* dimana semua fungsi individu digabung dan diberlakukan *testing* secara sekaligus secara otomatis. Hasil dari *integration test* akan di dokumentasikan lalu diperbaiki. *Integration Test* pada Oval Kompas Gramedia menggunakan *framework* bernama *Cypress*. Cypress adalah sebuah *End-to-End Testing Framework* Javascript yang digunakan penulis untuk melaksanakan *integration Test* terhadap *API endpoints* dari MyValue. Untuk melakukan *integration test* penulis menggunakan *Docker* untuk dijadikan *localhost* bagi *API endpoints* agar dapat dilakukan *testing* secara terpisah dari *API endpoints* yang sedang di *deploy.* Untuk menjalankan *integration test* penulis menggunakan bahasa pemrograman NodeJS pada Sistem Opersi Linux Ubuntu. Berikut adalah contoh sederhana program *integration testing cypress* yang dibuat penulis.



Gambar 3.1 Contoh gambar *Integration Test* Cypress

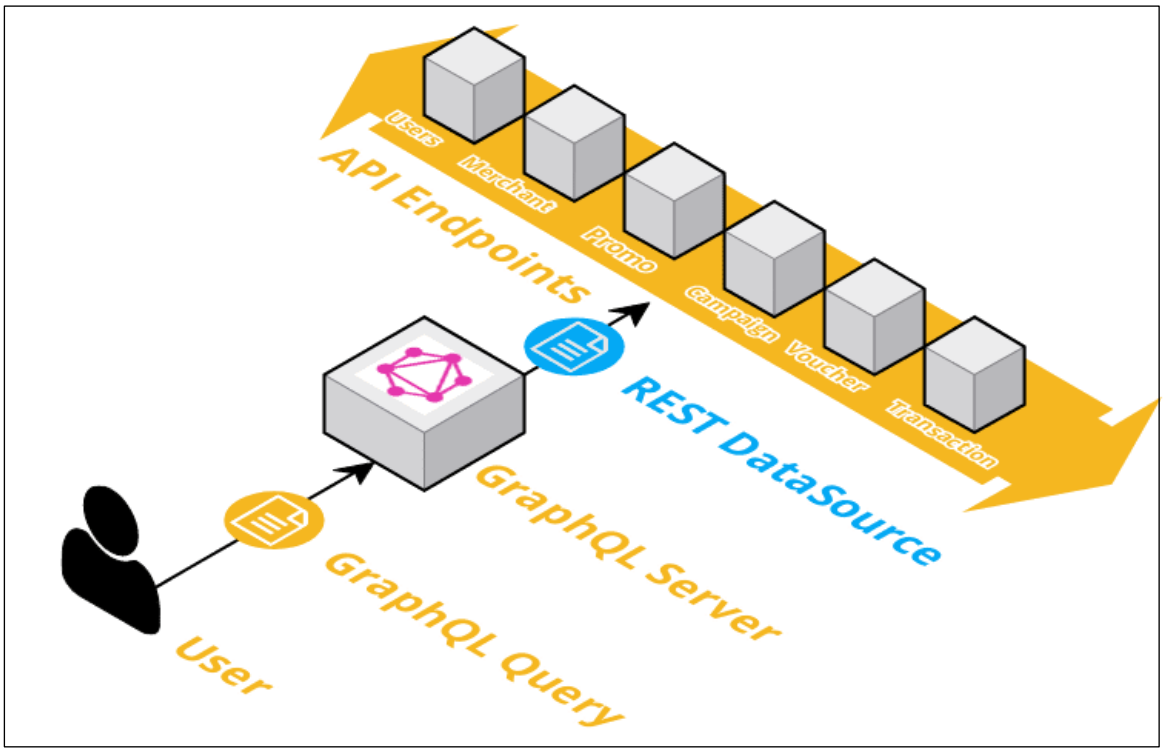
A.1. Dokumentasi

Dokumentasi hasil *Integration Test* dicatat ketika melakukan *commit* setelah proses *testing* yang kemudian akan muncul pada *bot Slack* yang dapat dibaca tim Oval untuk diperbaiki atau dilanjutkan ke tahap berikutnya.

B. GraphQL

GraphQL adalah sebuah *query and manipulation language* untuk *API*. *GraphQL* digunakan untuk mengembalikan detil data yang lengkap dan gampang dimengerti. Keunggulan *GraphQL* dari *web service architecture* yang lain adalah *GraphQL* mengembalikan data yang diminta persis dengan apa saja yang di minta pada *query,* hal ini mengurangi jumlah data yang tidak diperlukan dan meringankan proses pengembalian data. Oval Kompas Gramedia berencana untuk mengganti *monolithic architecture* menjadi *Backend for Frontend (BFF) micro-service* dengan bantuan *GraphQL* sebagai *entrypoint* dalam mengakses *API endpoints.*

Penulis diberikan tugas untuk membuat sebuah *GraphQL Server* yang akan terhubung dengan *MyValue API Endpoints* agar dapat menerima *query* dari *user* dan mengembalikan data dari *API* sesuai dengan *query* permintaan dari *user*. Berikut adalah contoh diagram dari *backend architecture* dengan *GraphQL* sebagai perantara *user* dan *API*, berikut adalah diagram perancangan *GraphQL Server.*



Gambar 3.2 Diagram perancangan *Backend Architecture* MyValue dengan *GraphQL*

B.1. *GraphQL* Schema

GraphQL Schema adalah sebuah fungsi dari *GraphQL* yang berfungsi untuk mendefinisikan struktur data dari kumpulan *API endpoint*. Berikut adalah contoh potongan *code* dari struktur *schema API* MyValue yang dikerjakan penulis.

```
graphqlServer > src > schema > JS schema.js > ...
1   const { gql } = require('apollo-server');
2
3   const usersSchema = require('./users.schema');
4   const campaignSchema = require('./campaign.schema');
5   const categorySchema = require('./category.schema');
6   const merchantSchema = require('./merchant.schema');
7   const promoSchema = require('./promo.schema');
8   const publishedBySchema = require('./publishedBy.schema');
9   const transactionSchema = require('./transaction.schema');
10  const userSchema = require('./user.schema');
11  const voucherSchema = require('./voucher.schema');
12
13
14  const typedefs = gql`
15    scalar Date
16  `;
17
18  module.exports = [
19    usersSchema,
20    typedefs,
21    campaignSchema,
22    categorySchema,
23    merchantSchema,
24    promoSchema,
25    publishedBySchema,
26    transactionSchema,
27    userSchema,
28    voucherSchema,
29  ];
30
```

Gambar 3.3 Contoh *Screenshot Schema API* MyValue

Pada *GraphQL*, *schema* mempunyai beberapa atribut yang akan mendefinisikan operasi yang akan dilakukan pada data dari *API endpoint*. *Query* akan mendefinisikan cara *user* dapat melakukan *query* data pada *API endpoint* yang diminta. *Mutation* dilakukan untuk melakukan manipulasi data *API* seperti *write, update,* dan *delete*. Berikut adalah contoh *code* dari *merchant schema* dari *API* MyValue yang dibuat oleh penulis.

```
graphqlServer > src > schema > merchant.schema.js > typeDefs
1   const { gql } = require('apollo-server');
2
3   const typeDefs = gql`
4
5     extend type Query {
6       merchants(page: Int = 1, pageSize: Int = 10): MerchantPagination
7       merchantById(id: ID!): Merchant
8     }
9
10    type MerchantPagination {
11      count : Int,
12      next: String,
13      previous: String,
14      results: [Merchant]
15    }
16
17    type Merchant {
18      url: String,
19      id: ID,
20      image: String,
21      promos: String,
22      totalPromo: Int,
23      totalLocation: String,
24      name: String,
25      isActive: Boolean,
26      logoImage: String,
27      backgroundImage: String,
28      description: String,
29      priority: Int,
30      instagramUrl: String,
31      facebookUrl: String,
32      twitterUrl: String,
33      websiteUrl: String,
34      createdDate: String,
35      updatedDate: String
36    }
37
38  `;
39
40  module.exports = typeDefs;
41
```

Gambar 3.4 Contoh *Screenshot Merchant Schema API* MyValue

B.2 *GraphQL Resolver*

GraphQL Resolver adalah sebuah fungsi dari *GraphQL* yang bertujuan untuk mengubah sebuah operasi *GraphQL* yang telah didefinisikan pada sebuah *schema* menjadi data yang kemudian akan dikembalikan kepada *user* dalam bentuk JSON. Sebuah *resolver* juga dapat memanggil *resolver* yang lain untuk membantu penyusunan pengembalian data. Berikut adalah contoh struktur *resolver* pada *API* MyValue dan yang dibuat penulis yang berisi berbagai *resolver* untuk mengambil data dari berbagai *API endpoints* MyValue.

```
graphqlServer > src > resolver > JS resolver.js > ...
1   const { GraphQLScalarType } = require('graphql');
2   const userResolver = require('./user.resolver');
3   const campaignResolver = require('./campaign.resolver');
4   const categoryResolver = require('./category.resolver');
5   const merchantResolver = require('./merchant.resolver');
6   const promoResolver = require('./promo.resolver');
7
8   const dateResolver = {
9     Date: new GraphQLScalarType({
10      name: 'Date',
11      description: 'A Date Time, represented as an ISO-8601 string',
12      serialize: value => new Date(value).toISOString(),
13      parseValue: value => new Date(value),
14      parseLiteral: ast => new Date(ast.value),
15    }),
16  };
17
18  const resolvers = [
19    dateResolver,
20    userResolver,
21    campaignResolver,
22    categoryResolver,
23    merchantResolver,
24    promoResolver,
25  ];
26
27  module.exports = resolvers;
28
```

Gambar 3.5 Contoh *Screenshot GraphQL Resolver API* MyValue

Sebuah *resolver* biasanya berisi *Query* dan *Mutation* yang telah didefinisikan pada *schema* yang akan dimasukkan data untuk dikembalikan kepada *user*. *Resolver* akan mengatur parameter dari *query* dan akan menentukan dari mana hasil *query* akan datang. Berikut adalah contoh *merchant resolver* pada *API* MyValue.

```
graphqlServer > src > resolver > JS merchant.resolver.js > ...
1  const merchantResolver = {
2    Query: {
3      merchants: (obj, args, { dataSources }) => dataSources.myValueApi.getMerchants(args),
4      merchantById: (obj, args, { dataSources }) => dataSources.myValueApi.getMerchantById(args),
5    },
6  };
7
8  module.exports = merchantResolver;
```

Gambar 3.6 Contoh *Screenshot GraphQL Merchant Resolver API* Myvalue

B.3 *GraphQL Server*

Untuk menjalankan *GraphQL* penulis menggunakan Appolo Server dari NodeJS sebagai *framework* yang akan menopang *GraphQL* agar dapat berjalan sebagai *web application*. Apollo Server juga mempunyai fungsi *REST DataSource* yang dapat digunakan *resolver GraphQL* untuk mengambil data dari sebuah *endpoint.* Berikut adalah contoh *code* yang dibuat penulis untuk *GraphQL Server* menggunakan Apollo Server.

```
graphqlServer > src > JS serverApp.js > ...
1  const { ApolloServer, makeExecutableSchema } = require('apollo-server');
2  const dataSource = require('./dataSource');
3
4  const typeDefs = require('./schema/schema');
5  const resolvers = require('./resolver/resolver');
6  const contextHandler = require('./utils/contextHandler');
7
8  const schema = makeExecutableSchema({ typeDefs, resolvers });
9
10 const server = new ApolloServer({
11   schema,
12   context: contextHandler,
13   dataSources: () => dataSource,
14 });
15
16 server.listen(4000, () => {
17   console.log('Server started on port 4000');
18 });
```

Gambar 3.7 Contoh *Screenshot GraphQL* Server untuk *API* MyValue

B.3.1 *GraphQL* Server Web Application

GraphQL Server mempunyai *Web Application* yang dapat diakses agar dapat memudahkan visualisasi *Query* dan *Manipulation API*. Berikut adalah tampilan *GraphQL Web Application* dan hasil *query* yagn dibuat oleh penulis.

Gambar 3.8 Contoh tampilan Web Application *GraphQL*.

B.3.2 Apollo Server Rest Datasource

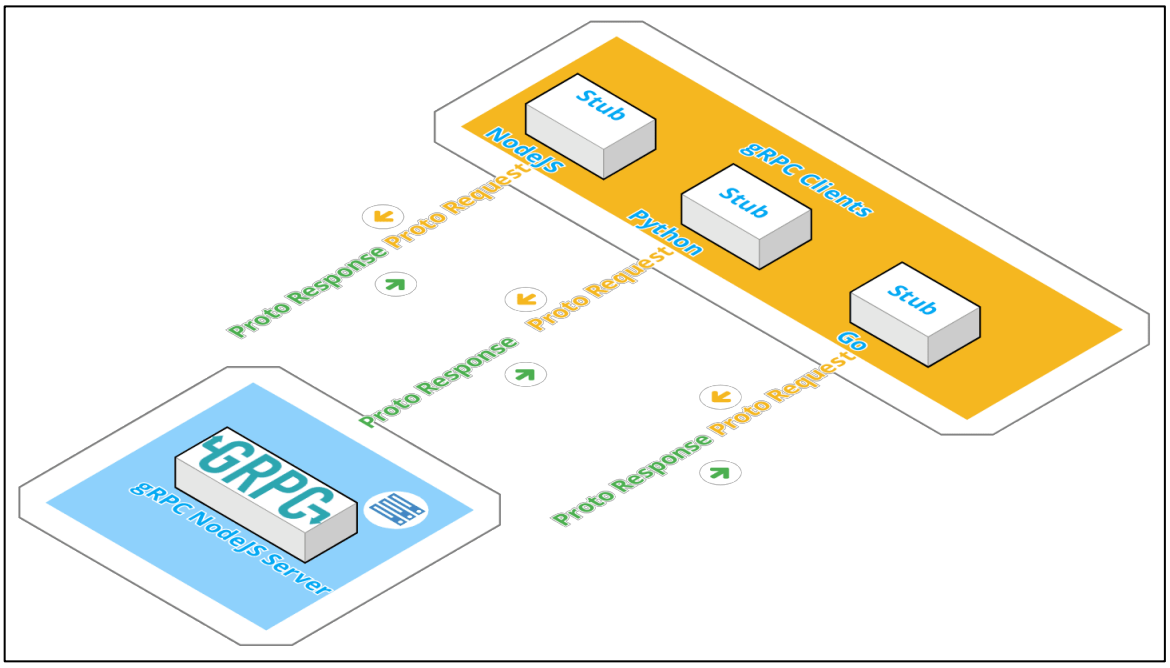
Apollo Server *REST Datasource* digunakan penulis untuk mengambil data dari *API endpoint* MyValue untuk diproses *GraphQL resolver* yang kemudian data tersebut akan dikembalikan kepada *user* yang memasukkan *query* dalam bentuk JSON. Pada sebuah *REST Datasource* terdapat berbagai fungsi yang bertujuan untuk mengirim *request* dengan *method* yang disesuaikan dengan jenis *query* atau *mutation* yang dilakukan. Berikut adalah contoh *code Rest Datasource* untuk pengambilan data *merchant schema* yang dibuat penulis.

```
1   const { RESTDataSource } = require('apollo-datasource-rest');
2
3   class MerchantApi extends RESTDataSource {
4     constructor() {
5       super();
6       this.baseURL = '127.0.0.1:50051';
7     }
8
9     async getMerchants(args) {
10      const merchants = await this.get(args);
11
12      return merchants;
13    }
14
15    async getMerchantById(id) {
16      const queryUrl = String(id);
17      const merchantById = await this.get(`merchants/${queryUrl}`);
18
19      return merchantById;
20    }
21  }
22
23  module.exports = MerchantApi;
```

Gambar 3.9 Contoh *Screenshot* Apollo Server *REST Datasource* untuk *merchant*

C. GRPC

GRPC (*general-purpose Remote Procedure Calls*) adalah sebuah RPC(*Remote Procedure Call*) *system* yang bersifat *open source* yang bertujuan untuk menghubungkan *backend service* dengan beberapa *device* atau *browser clients.* GRPC digunakan penulis untuk menghubungkan *API endpoints* MyValue dengan *GraphQL* dengan komunikasi *Client-Server* yang bersifat *cross-platform*. Hal ini memudahkan komunikasi antar *Client* dan *Server* yang berbeda bahasa pemrograman. Berikut adalah model diagram dari komunikasi antar *Client* dan *Server* menggunakan *gRPC.*



Gambar 3.10 Model diagram komunikasi *client* dan *server gRPC*

C.1. Protocol Buffers

Pembuatan *Server* & *Client gRPC* dibantu oleh *package* yang bernama *Protocol Buffers*. *Protocol Buffers* akan menghasilkan *server & client code* yang akan digunakan server dan client untuk berkomunikasi meskipun bahasa dasar dari *server* maupun *client* berbeda. File *Protocol Buffers* biasanya mempunyai *extension* ".proto" yang terdapat berbagai service untuk dijalankan *Server* dan dipakai oleh *Client.* Berikut adalah contoh *Protocol Buffer* untuk *endpoimt membership* dari *API* MyValue yang diterima oleh penulis.

```
syntax = "proto3";

package MembershipService;

message Empty {
}

service MembershipService {
    rpc RegisterUser (RegisterUserRequest) returns (UserResponse);
    rpc RegisterTemporaryUser (RegisterUserRequest) returns (RegisterUserResponse);
    rpc RegisterUserByAdmin (RegisterUserByAdminRequest) returns (UserResponse);
    rpc UpdateUser (UpdateUserRequest) returns (SimpleUserResponse);
    rpc UpdateTemporaryUser (UpdateTemporaryUserRequest) returns (Empty);
    rpc ApproveUpdatedUser (ApproveUpdatedUserRequest) returns (SimpleUserResponse);

    rpc GetUserByValueID (GetUserByValueIDRequest) returns (UserResponse);
    rpc GetUserByEmail (GetUserByEmailRequest) returns (UserResponse);
    rpc GetUserByMobilePhone (GetUserByMobilePhoneRequest) returns (UserResponse);
    rpc GetUserByUserID (GetUserByUserIDRequest) returns (UserResponse);
    rpc GetUserWithPagination (GetUserWithPaginationRequest) returns (UserResponsePagination);

    rpc GetMemberByMemberNo (GetMemberByMemberNoRequest) returns (MemberResponse);
    rpc GetMemberByValueID (GetMemberByValueIDRequest) returns (MemberResponse);
    rpc GetMemberByCardNumber (GetMemberByCardNumberRequest) returns (MemberResponse);
    rpc GetMemberByEmail (GetMemberByEmailRequest) returns (MemberResponse);
    rpc GetMemberByMobilePhone (GetMemberByMobilePhoneRequest) returns (MemberResponse);
    rpc GetMemberWithPagination (GetMemberWithPaginationRequest) returns (MemberResponsePagination);

    rpc GetUserRoleByUserID (GetUserRoleByUserIDRequest) returns (UserRoleResponse);
    rpc GetUserRoleByValueID (GetUserRoleByValueIDRequest) returns (UserRoleResponse);

    rpc CardMapping (CardMappingRequest) returns (CardMappingResponse);
    rpc VerifyCardMapping (VerifyCardMappingRequest) returns (VerifyCardMappingResponse);
    rpc ApproveCardMappingByAdmin (ApproveCardMappingByAdminRequest) returns (CardMappingResponse);
    rpc ActivateUser (ActivateUserRequest) returns (ActivateUserResponse);
    rpc BanUser (BanUserRequest) returns (UserResponse);
    rpc Authenticate (AuthenticationRequest) returns (UserResponse);
    rpc SetPassword (SetPasswordRequest) returns (SetPasswordResponse);

    rpc UpdateUserEmailByValueID (UpdateUserEmailByValueIDRequest) returns (UpdateUserEmailByValueIDResponse);
    rpc UpdateUserMobilePhoneByValueID (UpdateUserMobilePhoneByValueIDRequest) returns (UpdateUserMobilePhoneByValueIDResponse);
}
```

Gambar 3.11 *Screenshot Protocol Buffer* untuk *API membership* MyValue

C.2. GRPC Server

GRPC Server dibuat penulis menggunakan Javascript untuk menjalankan *service API endpoint membership* MyValue. Berikut adalah beberapa contoh *code* gRPC *server* yang dibuat oleh penulis beserta penjelasan bagian *code* tersebut.

```
const grpc = require('grpc');
const protoLoader = require('@grpc/proto-loader');

const proto = grpc.loadPackageDefinition(
  protoLoader.loadSync('membership.proto', {
    keepCase: true,
    longs: String,
    enums: String,
    defaults: true,
    oneofs: true,
  }),
);
```

Gambar 3.12 *Screenshot ProtoLoader* gRPC *server*

Pada gambar 3.11 teradapat fungsi *protoLoader* yang berfungsi untuk mengenali *server* kepada *Protocol Buffer* yang ingin di pakai. Hal ini bertujuan agar *server* dapat menyesuaikan *service* yang dijalankan sama dengan yang akan digunakan *client* nanti. Terdapat dua parameter pada fungsi loadSync yang berisi nama dari *protocol buffer* dan sebuah kumpulan variabel yang berfungsi untuk menyesuaikan pembacaan *Protocol Buffer*.

```
server.addService(proto.MembershipService.MembershipService.service, {
  GetUserByUserID: (call, callback) => {
    const user = call.request;
    const userIdx = users.findIndex(x => String(x.ID) === String(user.UserID));
    callback(null, users[userIdx]);
  },
  GetUserByValueID: (call, callback) => {
    const user = call.request;
    const userIdx = users.findIndex(x => x.ValueID === user.ValueID);
    callback(null, users[userIdx]);
  },
  GetUserByEmail: (call, callback) => {
    const user = call.request;
    const userIdx = users.findIndex(x => String(x.Email) === String(user.Email));
    callback(null, users[userIdx]);
  },
  GetUserByMobilePhone: (call, callback) => {
    const user = call.request;
    const userIdx = users.findIndex(x => String(x.MobilePhone) === String(user.MobilePhone));
    callback(null, users[userIdx]);
  },
```

Gambar 3.13 *Screenshot* potongan *code gRPC Server Services*

Pada gambar 3.12 terdapat fungsi "*addService*" yang berfungsi untuk menambahkan *service* ke dalam *server. Service* pada gambar bertujuan untuk mengembalikan *object user* pada fungsi "*callback"* dengan menggunakan parameter yang berbeda-beda.

```
server.bind('127.0.0.1:50051',
  grpc.ServerCredentials.createInsecure());
console.log('Server running at localhost:50051');
server.start();
```

Gambar 3.14 *Screenshot* potongan *code* gRPC *Server Bind and Start*

Pada gambar 3.13 terdapat fungsi "bind" yang berfungsi untuk menentukan tempat berjalannya server, dan terdapat fungsi "*createInsecure*", yang berfungsi untuk menjalankan *server* dalam mode *insecure* yang berarti proses HTTP *authentication* tidak dilindungi oleh SSL, hal ini dibiarkan karena sedang dijalankan pada *localhost.*

3.3.2 Kendala yang Ditemukan

Dalam proses pelaksanaan kerja magang, terdapat beberapa kendala yang dihadapi. Kendala-kendala yang ditemukan adalah antara lain seperti berikut:

1. Proses penyesuaian dengan tahapan dan struktur pengerjaan project dengan menggunakan *framework agile scrum.*
2. Beberapa konsep *backend development* yang belum pernah dipelajari, terutama dengan materi *asynchronous programming* seperti, *call, callbacks,* dan *promise.*

3.3.3 Solusi atas Kendala yang Ditemukan

Dari kendala-kendala yang ditemukan selama proses kerja magang, terdapat beberapa solusi yang ditemukan untuk menyelesaikan kendala tersebut agar tidak menghambat pengerjaan tugas kerja magang. Solusi dari kendala-kendala yang ditemukan adalah antara lain sebagai berikut.

1. Mencoba menyesuaikan diri dengan rajin menghadiri *meeting* yang diadakan dan melakukan observasi pada cara kerja anggota tim Oval Kompas Gramedia.
2. Mempelajari *asynchronous programming* dari berbagai website seperti Youtube dan Stackoverflow, serta melakukan bimbingan bersama mentor dalam meeting *one on one.*