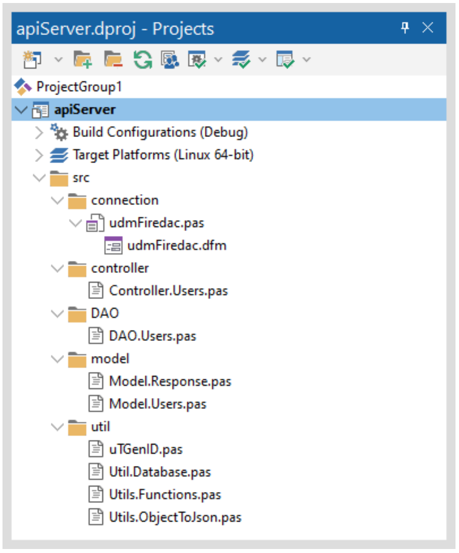
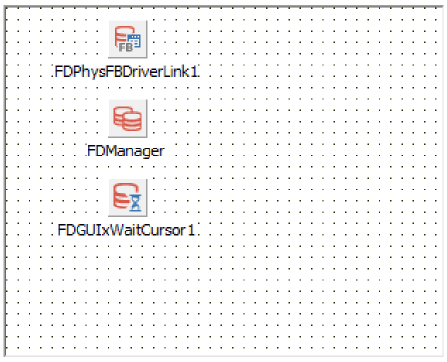
# MsUsers

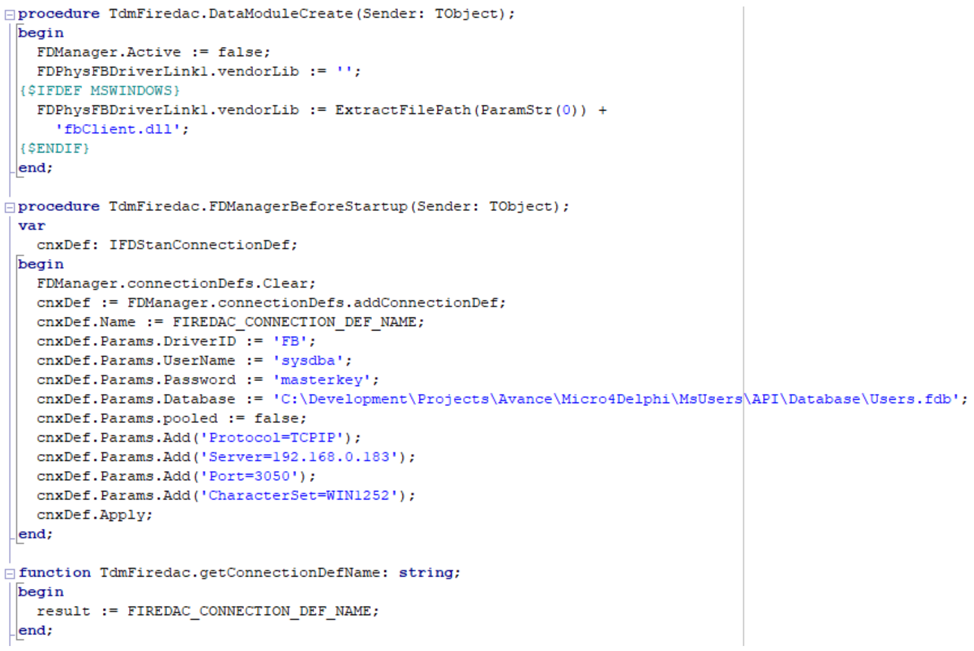
1. The microservice is organized following good software development practices.



```
apiServer.dproj - Projects
ProjectGroup1
  apiServer
    Build Configurations (Debug)
    Target Platforms (Linux 64-bit)
    src
      connection
        udmFiredac.pas
          udmFiredac.dfm
      controller
        Controller.Users.pas
      DAO
        DAO.Users.pas
      model
        Model.Response.pas
        Model.Users.pas
      util
        uTGenID.pas
        Util.Database.pas
        Utils.Functions.pas
        Utils.ObjectToJson.pas
```

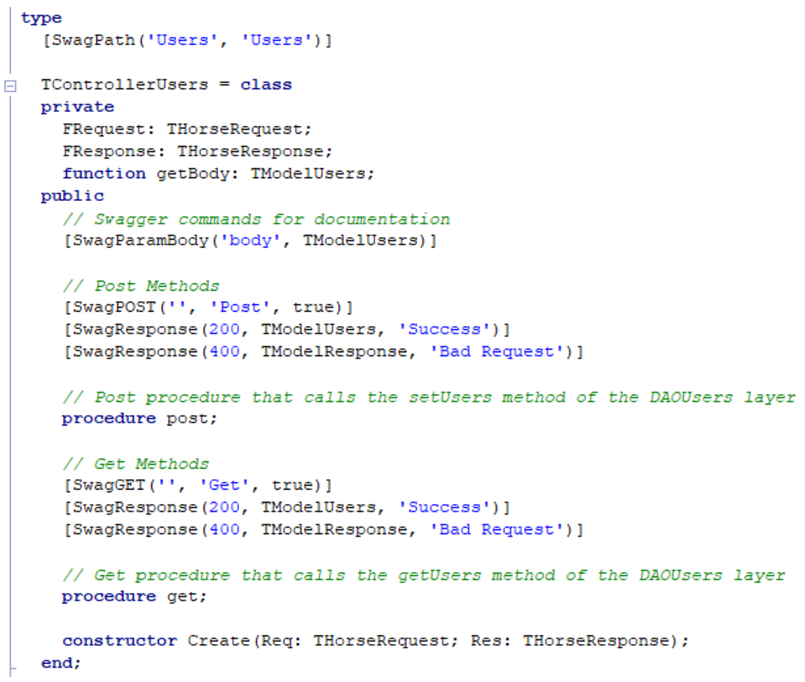
2. In the “connection” folder, we have the “udmFiredac.dfm” data module with the connection components. In the “udmFiredac.pas” unit is implemented the source code for the microservice to connect to a Firebird database.



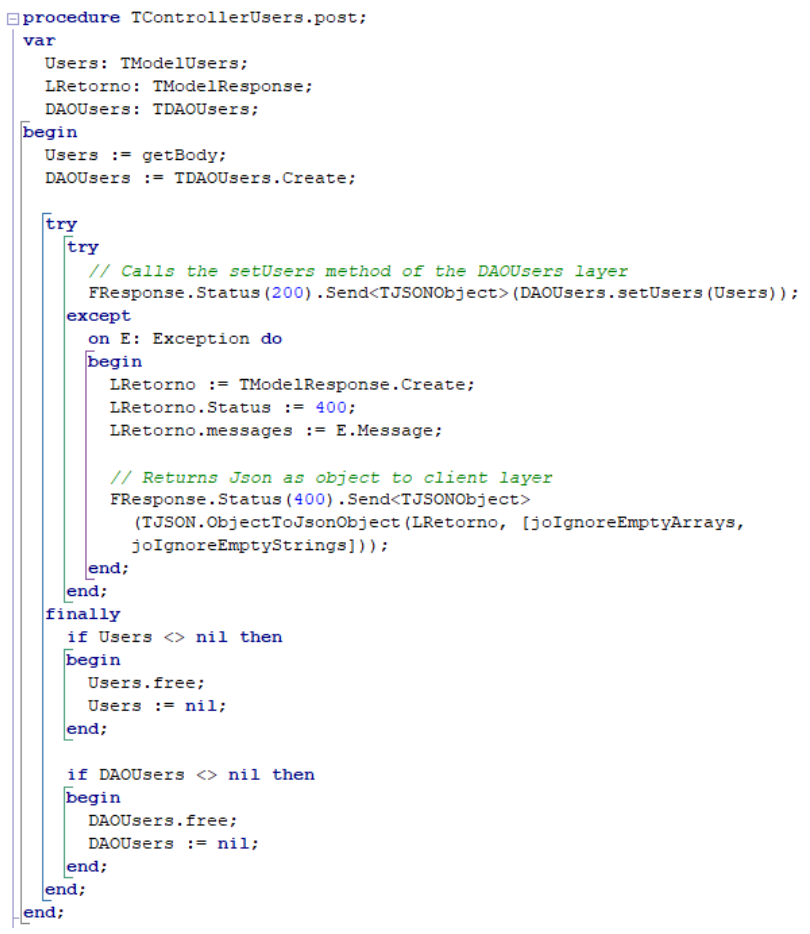


```
procedure TdmFiredac.DataModuleCreate(Sender: TObject);
begin
  FDManager.Active := false;
  FDPhysFBDriverLink1.vendorLib := '';
{$IFDEF MSWINDOWS}
  FDPhysFBDriverLink1.vendorLib := ExtractFilePath(ParamStr(0)) +
    'fbClient.dll';
{$ENDIF}
end;

procedure TdmFiredac.FDManagerBeforeStartup(Sender: TObject);
var
  cnxDef: IFDStanConnectionDef;
begin
  FDManager.connectionDefs.Clear;
  cnxDef := FDManager.connectionDefs.addConnectionDef;
  cnxDef.Name := FIREDAC_CONNECTION_DEF_NAME;
  cnxDef.Params.DriverID := 'FB';
  cnxDef.Params.UserName := 'sysdba';
  cnxDef.Params.Password := 'masterkey';
  cnxDef.Params.Database := 'C:\Development\Projects\Avance\Micro4Delphi\MsUsers\API\Database\Users.fdb';
  cnxDef.Params.pooled := false;
  cnxDef.Params.Add('Protocol=TCPIP');
  cnxDef.Params.Add('Server=192.168.0.183');
  cnxDef.Params.Add('Port=3050');
  cnxDef.Params.Add('CharacterSet=WIN1252');
  cnxDef.Apply;
end;

function TdmFiredac.getConnectionDefName: string;
begin
  result := FIREDAC_CONNECTION_DEF_NAME;
end;
```

3. In the “controller” folder, we have the “Controller.Users” unit responsible for controlling access to the “DAO” layer that makes requests and changes to the database. This unit also contains Swagger commands for the automatic creation of Users API documentation.



```
type
  [SwagPath('Users', 'Users')]

  TControllerUsers = class
  private
    FRequest: THorseRequest;
    FResponse: THorseResponse;
    function getBody: TModelUsers;
  public
    // Swagger commands for documentation
    [SwagParamBody('body', TModelUsers)]

    // Post Methods
    [SwagPOST('', 'Post', true)]
    [SwagResponse(200, TModelUsers, 'Success')]
    [SwagResponse(400, TModelResponse, 'Bad Request')]

    // Post procedure that calls the setUsers method of the DAOUsers layer
    procedure post;

    // Get Methods
    [SwagGET('', 'Get', true)]
    [SwagResponse(200, TModelUsers, 'Success')]
    [SwagResponse(400, TModelResponse, 'Bad Request')]

    // Get procedure that calls the getUsers method of the DAOUsers layer
    procedure get;

    constructor Create(Req: THorseRequest; Res: THorseResponse);
  end;
```

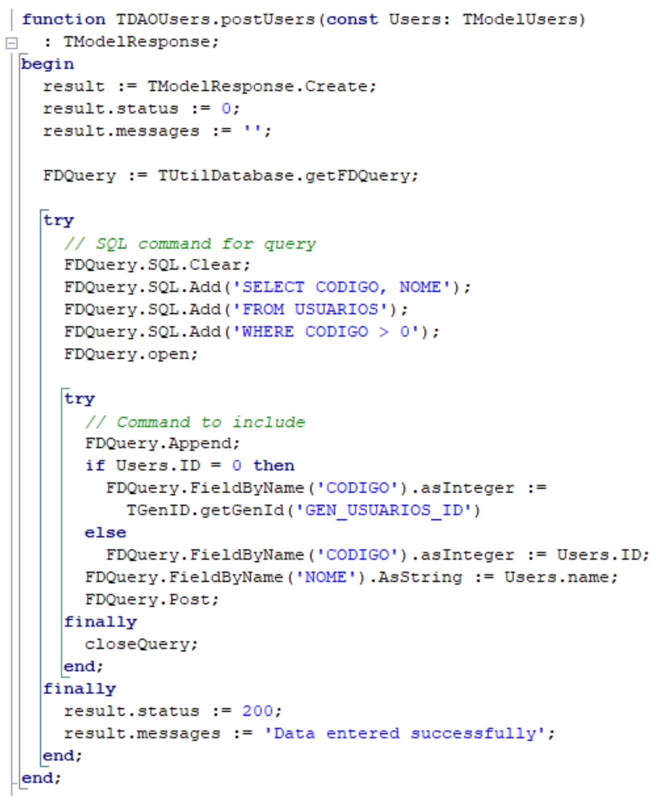


```
procedure TControllerUsers.post;
var
  Users: TModelUsers;
  LRetorno: TModelResponse;
  DAOUsers: TDAOUsers;
begin
  Users := getBody;
  DAOUsers := TDAOUsers.Create;

  try
    try
      // Calls the setUsers method of the DAOUsers layer
      FResponse.Status(200).Send<TJSONObject>(DAOUsers.setUsers(Users));
    except
      on E: Exception do
      begin
        LRetorno := TModelResponse.Create;
        LRetorno.Status := 400;
        LRetorno.messages := E.Message;

        // Returns Json as object to client layer
        FResponse.Status(400).Send<TJSONObject>
          (TJSON.ObjectToJsonObject(LRetorno, [joIgnoreEmptyArrays,
          joIgnoreEmptyStrings]));
      end;
    end;
  finally
    if Users <> nil then
    begin
      Users.free;
      Users := nil;
    end;

    if DAOUsers <> nil then
    begin
      DAOUsers.free;
      DAOUsers := nil;
    end;
  end;
end;
```

```
function TControllerUsers.getBody: TModelUsers;
var
  jsonValue: TJSONObject;
  ReqBody: string;
begin
  result := TModelUsers.Create;
  ReqBody := FRequest.Body;

  if copy(ReqBody, 1, 1) = '[' then
  begin
    ReqBody := StringReplace(ReqBody, '#92', '\', [rfReplaceAll]);
    ReqBody := copy(ReqBody, 2, length(ReqBody) - 2);
  end;

  if ReqBody <> '' then
  begin
    jsonValue := TJSONObject.ParseJSONValue(ReqBody) as TJSONObject;

    if jsonValue <> nil then
    begin
      if jsonValue.GetValue('codigo') <> nil then
        result.id := jsonValue.GetValue<integer>('codigo');
      if jsonValue.GetValue('nome') <> nil then
        result.name := jsonValue.GetValue<string>('nome');
    end;
  end;
end;
```

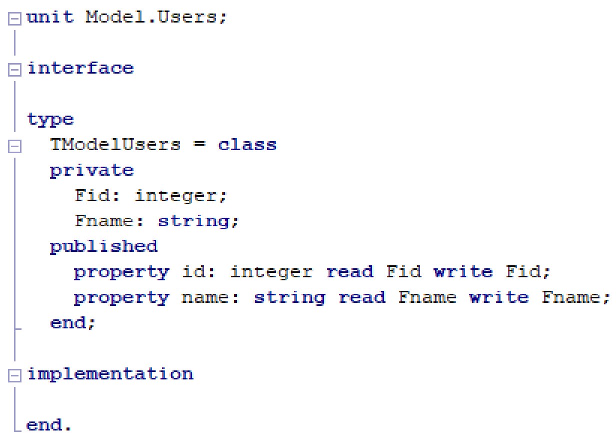
4. In the “DAO” folder, we have the “DAO.Users” unit responsible for performing SQL commands for querying and including users in the database. To query users, the “getUsers” method is used, and, for inclusion, the “postUsers” method is used.

```
function TDAOUsers.getUsers: TJSONArray;
var
  Users: TModelUsers;
  UsersList: TArray<TObject>;
begin
  result := nil;
  UsersList := TArray<TObject>.Create(nil);
  FDQuery := TUtilDatabase.getFDQuery;

  try
    // SQL command for query
    FDQuery.SQL.Clear;
    FDQuery.SQL.Add('SELECT CODIGO, NOME');
    FDQuery.SQL.Add('FROM USUARIOS');
    FDQuery.SQL.Add('WHERE CODIGO > 0');
    FDQuery.open;

    SetLength(UsersList, FDQuery.RecordCount);

    while not FDQuery.Eof do
    begin
      // Writes the return to the properties of Model.Users
      Users := TModelUsers.Create;
      Users.id := FDQuery.FieldByName('CODIGO').asInteger;
      Users.name := FDQuery.FieldByName('NOME').AsString;

      // Adiciona os dados na lista de usuários
      UsersList[FDQuery.recno - 1] := Users;
      FDQuery.next;
    end;

    // Returns the list in Json format to the controller
    if Length(UsersList) > 0 then
      result := getJsonArray(UsersList);
  finally
    closeQuery;

    if UsersList <> nil then
      UsersList := nil;
  end;
end;
```

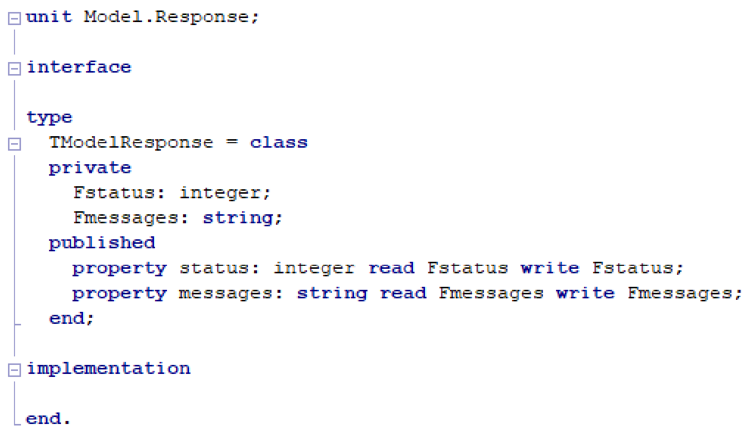


```
function TDAOUsers.postUsers(const Users: TModelUsers)
  : TModelResponse;
begin
  result := TModelResponse.Create;
  result.status := 0;
  result.messages := '';

  FDQuery := TUtilDatabase.getFDQuery;

  try
    // SQL command for query
    FDQuery.SQL.Clear;
    FDQuery.SQL.Add('SELECT CODIGO, NOME');
    FDQuery.SQL.Add('FROM USUARIOS');
    FDQuery.SQL.Add('WHERE CODIGO > 0');
    FDQuery.open;

    try
      // Command to include
      FDQuery.Append;
      if Users.ID = 0 then
        FDQuery.FieldByName('CODIGO').asInteger :=
          TGenID.getGenId('GEN_USUARIOS_ID')
      else
        FDQuery.FieldByName('CODIGO').asInteger := Users.ID;
      FDQuery.FieldByName('NOME').AsString := Users.name;
      FDQuery.Post;
    finally
      closeQuery;
    end;
  finally
    result.status := 200;
    result.messages := 'Data entered successfully';
  end;
end;
```

5. In the “model” folder, we have the “Model.Users” unit responsible for instantiating the properties of the user's table and the “Model.Response” unit responsible for instantiating the return properties of requests from the Client layer.



```
unit Model.Users;

interface

type
  TModelUsers = class
  private
    Fid: integer;
    Fname: string;
  published
    property id: integer read Fid write Fid;
    property name: string read Fname write Fname;
  end;

implementation

end.
```



```
unit Model.Response;

interface

type
  TModelResponse = class
  private
    Fstatus: integer;
    Fmessages: string;
  published
    property status: integer read Fstatus write Fstatus;
    property messages: string read Fmessages write Fmessages;
  end;

implementation

end.
```

6. In the “util” folder, we have the project’s auxiliary units:
    a. “uTGenID”, responsible for searching for the generator of the USUARIOS table in the database;
    b. “Util.Database”, responsible for creating the “udmFiredac” data module and the database connection objects, namely “FFDConnection” and “FDQuery”;
    c. “Utils.Functions”, responsible for allocating all auxiliary functions of the project; and
    d. “Utils.ObjectToJson”, responsible for converting objects into JSON.