**EASY-DATA-Center – Build & Run**

EASYDATACENTER je jednoduchý způsob, jak publikovat existující DB, vytvářet novou DB, spojovat se mezi systémy, vytvářet rozšíření IS, webové stránky, produkční terminály a DALŠÍ běžící na jednoduché technologii AspNetCore6 RESTFULL API + WEBSOCKET

**Univerzální bezpečný RestFull / WebSocket**

**Backend API Server, Podpora LINUX, GooGleCloud, DOCKER, WINDOWS KONSOLE / SERVICE**

**PRO KAŽDOU FIRMU, ŠKOLU, IT ODDĚLENÍ**

**Koncept zůstává na logice -> vytvořit tabulku se závislostmi na jiných tabulkách a dále bude vše nastaveno automaticky stávající logikou na Backend Serveru.**

**Jednoduchá a rychlá každá příchozí/odchozí definice API z připravených úloh.**

**HTTP / HTTPS / WEBSOCKET se zabezpečením**

**Basic / JwtBearer / CORS / COOKIE /HSTS /..**

**Je to univerzální bezpečné řešení Build&Run**

**pro rychlé, levné a jednoduché nastavení I/O dat na databázích MSSQL / MYSQL / ORACLE databáze Azure SQL / PostgreSQL / SQLite / LINQ**

**Řešení obsahuje automatizované logiky i jádro**

EASYDATACenter ASPNET.CORE6 projekt konzolové aplikace s podporou Dockeru

EASYDATACenterWinService pro OS Windows jako Windows Service

DOCKER build konfigurace pro GoogleCloud, GitHub a jiná Linux/Docker řešení

**Univerzální Server běží na OS LIN/WIN/GCloud/..**

Podpora Linux OS s ASPNET.CORE6 (Docker)

Podpora GoogleCloud Service (Docker)

Podpora Windows OS XP+ With ASPNET.CORE 6

Podpora WINDOWS SERVERS With ASPNET.CORE 6

**Funguje jako nenáročná Služba**

Linux ASPNET.CORE6 service

GoogleCloud Service (Docker)

Windows Service, Windows Console Application

**Extrémě stručný vývis Použitých technologií (>100)**

AspNetCore 6

EntityFrameWok 6

HTTP/HTTPS/WebSocket/RestFull / [GET,POST,PATCH,PUT,DELETE,OPTION]

Windows Service / Docker Service / Console application

CookiePolicy , CorsPolicy / Authentification / Authorization / Basic–JwtBearer Tokens

Automatic EndPoints /Controllers, HealthCheck, AutoApi, Auto Data Manage, FTP, WEB, Socket, Auto DB Manage,Inteligent Szstem Control, Tbl use to change,

**Super Akce**

Žádost o koupi licence zašlete na Email nebo objednejte v Eshopu.

Plná verze k vyzkoušení již nyní ke stažení v odkazu Eshopu.

Dokumentace ke stažení v Eshopu v detailu položky.

Zprovoznit vlastní backend server nebylo nikdy jednodušší.

Neváhejte si nasadit toto jednoduché řešení už do 1 hodiny od objednání.

**JSON configuration (folder Data/config.json) (nyní 50pol)**

"DbConnectionString": - MSSQL DB connection string. Je pouze v konfiguračním souboru.

"JwtLocalKey": - JWT Local klíč si můžete nastavit libovolný. [SDB] [R]

"DefaultApiPort": - Výchozí Port pro HTTP/HTTPS/SOCKET. [SDB] [R]

"SocketTimeoutMinutes": - Výchozí WebSocket Timeout. [SDB] [R]

"SocketBufferSizeKb": - Výchozí WebSocket max. velikost zprávy. [SDB] [R]

"TimeTokenValidation": - Zapnuto/Vypnuto expirace tokenu. [SDB]

"LoginTimeoutMinutes": - Počet minut do expirace tokenu. [SDB] [Z]

"HttpsProtocol": - Zapnuto/Vypnuto HTTP/ HTTPS. Jen 1 volba je v provozu. [SDB]

"CertificateDomain": - Certifikát pro HTTPS je generován automaticky pokud je HTTPS aktivní.

Tato doména je vložena do certifikátu. [SDB] [Z]

"CertificatePassword": - Heslo pro zabezpečení certifikátu je povinné. [SDB] [Z]

Legend:

[SDB] - Při Startu servertu je hiodnota přepsána hodnotou z DB. Tabulka: „AdminConfiguration“

[R] - Hodnota je povinná

[O] - Hodnota je volitelná

[Z] - Hodnota závisí na stavu funkcionality

**Instalace Windows Service - Automatická**

Z adresářů From Debug/Release spusťte scripty jako „ADMINISTRÁTOR“ následujícími příkazy:

* „install.bat INSTALL“ - instalace jako Windows Service s názvem „EASYDATACenterWinService“
* „install.bat UNINSTALL“ - uninstall service

**Database Model snapshot to MSSQL Entity Entity Framework 6 Libovolné DB**

První zelené příkazy slouží pro jednoduchou přímou práci s databází MSSQL.

Tento příkaz generuje „úplnou entitu DB modelu“.

Příkaz musí být spuštěn z „konzole správce balíčků“.

1)

Příkaz pro: MSSQL vygenerování celého DB Modelu – Database Model Entity Framework

Scaffold-DbContext "Server=MSSQLSRV\LOCALSRV;Database=DATAPUB;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir DBModel -ContextDir "DBContexts"

Příkaz pro: MSSQL vygenerování Entity DB Modelu pouze pro vybrané tabulky

Scaffold-DbContext "Server=MSSQLSRV\LOCALSRV;Database=DATAPUB;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -ContextDir DbContext -OutputDir DBTables -Tables UserList,UserRoleList,LoginHistory

Příkaz pro: MSSQL vygenerování celého DB Modelu – Database Model Entity Framework

Scaffold-DbContext "Server=localhost;Database=test;Uid=test;Pwd=test;Port=3306" Pomelo.EntityFrameworkCore.MySql -OutputDir Models1

Příkaz pro: MSSQL vygenerování Entity DB Modelu pouze pro vybrané tabulky

Scaffold-DbContext "Server=localhost;Database=test;Uid=test;Pwd=test;Port=3306;" Pomelo.EntityFrameworkCore.MySql -OutputDir MySqlDBModel -Tables UserList,UserRoleList

2)

Nahraďte Connection string v souboru DBContext tímto nastavením.

(Zde si také můžete zapnout Debugování SQL)

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer(Program.ServerSettings.DbConnectionString,

x => x.MigrationsHistoryTable("MigrationsHistory", "dbo"));

//.UseLoggerFactory(LoggerFactory.Create(builder => { builder.AddConsole(); }))

//.LogTo(message => Debug.WriteLine(message));

//.LogTo(Console.WriteLine);

}

3) Pokud chcete vypnout ForeignKey kontrolu, musíte přidat tyto parametry JSONIgnore & ValidateNever přímo k danému poli do DBModelu. Příklad z UserList:

[JsonIgnore]

[ValidateNever]

public virtual UserRoleList Role { get; set; } = null!;

**UKÁZKY**

**Ukázka JSON konfigurace**

{

"DbConnectionString": "Server=MSSQL\\LOCALSRV;Database=DATAPUB;User Id=tester;Password=tester;",

"JwtLocalKey": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9",

"DefaultApiPort": "5000",

"SocketTimeoutMinutes": 2,

"SocketBufferSizeKb": 10,

"TimeTokenValidation": false,

"LoginTimeoutMinutes": 15,

"HttpsProtocol": false,

"CertificateDomain": "127.0.0.1",

"CertificatePassword": "password"

}

**Ukázka HTTPGET List Definice**

[HttpGet("/AdminConfiguration")]

public async Task<string> GetAdminConfiguration()

{

List<AdminConfiguration> data;

using (new TransactionScope(TransactionScopeOption.Required, new TransactionOptions

{ IsolationLevel = IsolationLevel.ReadUncommitted })) //with NO LOCK

{ data = new DATAPUBContext().AdminConfigurations.ToList(); }

return JsonSerializer.Serialize(data);

}

**Ukázka HTTPPOST List Definice**

[HttpPost("/LoginHistory/Name")]

[Consumes("application/json")]

public async Task<string> GetLoginHistoryName([FromBody] NameFilter Name)

{

if (string.IsNullOrWhiteSpace(Name.Name)) return JsonSerializer.Serialize(new DBResultMessage()

{ status = DBResult.error.ToString(), recordCount = 0, ErrorMessage = "Id is null" });

List<LoginHistory> data;

using (new TransactionScope(TransactionScopeOption.Required, new TransactionOptions

{ IsolationLevel = IsolationLevel.ReadUncommitted }))

{ data = new DATAPUBContext().LoginHistories.Where(a => a.UserName == Name.Name.ToString()).ToList(); }

return JsonSerializer.Serialize(data);

}

**Ukázka Authentication volání**

POST /Authentication HTTP/1.1

Host: 127.0.0.1:5000

content-type: application/json

authorization: Basic dGVzdGVyOnRlc3Rlcg==

**Ukázka volání s jednoduchým filtrem**

POST /LoginHistory/Name HTTP/1.1

Host: localhost:5000

content-type: application/json

authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiJ0ZXN0ZXIiLCJyb2xlIjoidXNlciIsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL2RucyI6IjEyNy4wLjAuMSIsIm5iZiI6MTY2MzYxNDQ4NCwiZXhwIjoxNjYzNjE1Njg0LCJpYXQiOjE2NjM2MTQ0ODQsImlzcyI6InRlc3RlciJ9.DcQrAyc1vDIARm4Am\_46TujwVxzXmcp1cvxEiwZC5LY

{"Name":"tester"}

**Ukázka HTTPPOST Update volání**

[HttpPost("/UserList")]

[Consumes("application/json")]

public async Task<string> UpdateUserList([FromBody] UserList record)

{

try

{ var data = new DATAPUBContext().UserLists.Update(record);

int result = await data.Context.SaveChangesAsync();

if (result > 0) return JsonSerializer.Serialize(new DBResultMessage()

{ status = DBResult.success.ToString(), recordCount = result, ErrorMessage = string.Empty });

else return JsonSerializer.Serialize(new DBResultMessage()

{ status = DBResult.error.ToString(), recordCount = result, ErrorMessage = string.Empty });

}

catch (Exception ex)

{ return JsonSerializer.Serialize(new DBResultMessage() { status = DBResult.error.ToString(), recordCount = 0, ErrorMessage = ex.Message }); }

}

**Literatura:**

<https://code-maze.com/net-core-web-development-part1/>

<https://docs.microsoft.com/cs-cz/aspnet/core/security/cors?view=aspnetcore-6.0>

<https://docs.microsoft.com/cs-cz/ef/core/cli/dotnet>

<https://medium.com/free-code-camp/an-awesome-guide-on-how-to-build-restful-apis-with-asp-net-core-87b818123e28>

<https://stackoverflow.com/questions/48282223/scaffold-dbcontext-to-different-output-folder>

<https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration/>

**SCREENSHOTS**

 