مرکز تنظیم مقررات نظام پایانه‌های فروشگاهی و سامانه مودیان

سند

«راهنمای اتصال به سامانه مودیان از طریق SDK دات نت همراه با گواهی امضاء»

شهریورماه ۱۴۰۲

مقدمه

زیرسامانه‌ی جمع‌آوری صورتحساب یکی از زیرسامانه‌های سامانه‌ی مودیان است که وظیفه‌ی دریافت صورتحساب، ارسال به هسته، گرفتن نتیجه اعتبارسنجی صورتحساب و ذخیره‌کردن آن و پاسخ به استعلام‌های صورتحساب‌های ارسالی از سمت مودی را برعهده دارد. این زیرسامانه دارای یک وب‌سرویس می‌باشد که تمامی درخواست‌ها از طریق این وب‌سرویس به سامانه ارسال شده و پاسخ داده می‌شوند. این وب‌سرویس در چهارچوب REST API پیاده سازی شده و فراخوانی آن نیازمند احراز هویت[1] مودی از طریق امضای دیجیتال می‌باشد.

در این سند نحوه‌ی استفاده از کیت توسعه‌ی نرم‌افزاری (SDK) اتصال به سامانه‌ی مودیان توضیح داده خواهد شد. این کیت شامل کتابخانه‌ها و ابزارهای نوشته شده به زبان .NET می‌باشد که فرآیند اتصال به وب‌سرویس جمع‌آوری سامانه‌ی مودیان و ارسال صورتحساب را برای مودی تسهیل می‌نماید. مثال‌های کاربردی از هر یک از عملیات‌های قابل انجام توسط وب‌سرویس جمع‌آوری به همراه نمونه کد و توضیحات ذکر خواهد شد.

لازم به ذکر است استفاده از این SDK برای ارسال صورتحساب اختیاری است و جهت سهولت کار مودیان قرار داده شده است.

[1] Authentication

فهرست مطالب

نیازمندی‌های راه‌اندازی پروژه

برای راه اندازی پروژه و ارسال صورتحساب در .NET نیز موارد زیر را نیاز دارید:

۱. کیت توسعه .NET dotnet framework 4.7 به بالا یا dotnet و dotnet core نسخه ۳ به بالا

۲. نوگت کتابخانه TaxCollectData.Library.2.0.9.nupkg

۳. گواهی امضای دیجیتال صادرشده توسط مراکز میانی معتبر کشور (به فرمت crt یا cer)

۴. کلید خصوصی متناظر با گواهی امضا

- به فرمت PKCS#8 Pem Encoded
- یا توکن سخت‌افزاری (فعلا تنها توکن epass3003 پشتیبانی می‌شود.)[۲]

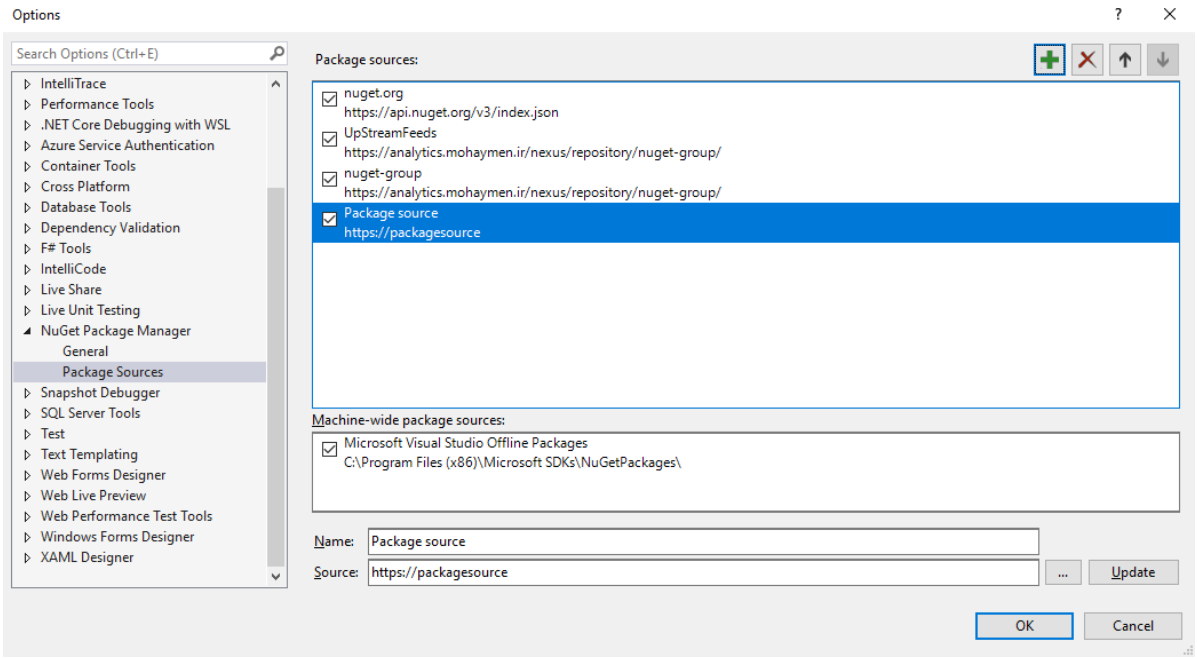
راه‌اندازی پروژه

برای اضافه کردن SDK در .NET. ابتدا یک پروژه‌ی جدید ایجاد می‌کنیم. سپس باید بسته‌ی نوگت را به پروژه اضافه نمائیم.

راهنمای اضافه کردن نوگت به پروژه با استفاده از Visual Studio

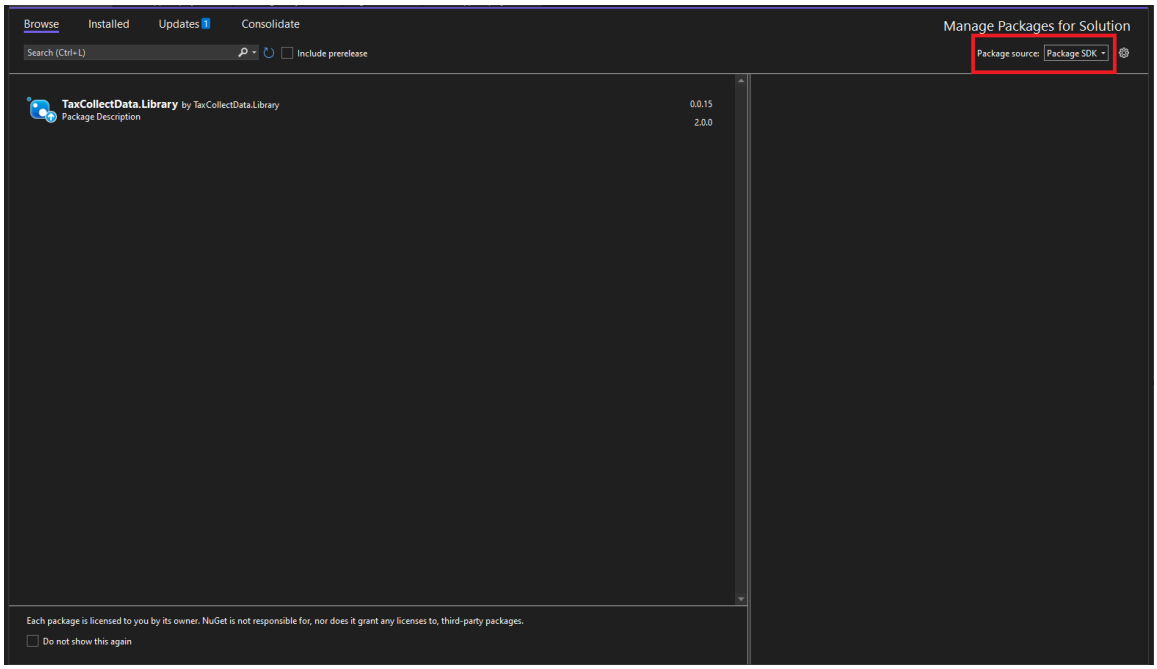
ابتدا فایل نوگت با پسوند .nupkg را دانلود کرده و آن را داخل یک پوشه دلخواه قرار می‌دهیم. سپس از منوی Tools قسمت NuGet Package Manager روی گزینه Package Manager Settings کلیک می‌کنیم. سپس در پنجره بازشده از سمت چپ روی قسمت Package Sources کلیک می‌کنیم. پنجره بازشده مشابه تصویر زیر است:

[۲] راهنمای اتصال به توکن سخت‌افزاری در قسمت پیوست گفته می‌شود.



حال از قسمت بالا روی دکمه + کلیک می‌کنیم. بعد در قسمت Source روی دکمه ... کلیک می‌کنیم و آدرس فولدری را که فایل نوگت مورد نظر داخل آن است را انتخاب می‌کنیم و OK را می‌زنیم.

حال از منوی Tools قسمت NuGet Package Manager روی گزینه Manage NuGet Packages for Solution... کلیک می‌کنیم:



از قسمت بالا روی گزینه Package source کلیک کرده و گزینه نشان‌داده‌شده یعنی Package source (یا در صورت انتخاب نام دلخواه در مرحله قبل نام مربوطه) را انتخاب می‌کنیم تا مطابق شکل پکیج TaxCollectData.Library ظاهر شود. سپس پکیج مذکور را انتخاب کرده و با کلیک بر روی گزینه install آن را نصب می‌کنیم.

پیکربندی SDK

قبل از شروع کار به استفاده از SDK باید آن را پیکربندی نمائیم و یک شیء TaxApi بسازیم. بدین منظور تکه کد زیر را در ابتدای پروژه خود بنویسید:

```
Pkcs8SignatoryFactory pkcs8SignatoryFactory = new Pkcs8SignatoryFactory();
TaxProperties properties = new TaxProperties("MEMORY_ID");
EncryptorFactory encryptorFactory = new EncryptorFactory();

TaxApiFactory taxApiFactory = new TaxApiFactory("API_URL", properties);

ISignatory signatory = pkcs8SignatoryFactory.Create(
    "PRIVATE_KEY_FILE",
    "CERTIFICATE_FILE");

ITaxPublicApi publicApi = taxApiFactory.CreatePublicApi(signatory);
IEncryptor encryptor = encryptorFactory.Create(publicApi);
ITaxApi taxApi = taxApiFactory.CreateApi(signatory, encryptor);
```

ITaxApi اینترفیسی است که عملیات ارتباط با سامانه‌ی مودیان برای ما انجام می‌دهد و تکه‌کد بالا یک پیاده سازی از این اینترفیس را در اختیار ما قرار می‌دهد. توجه داشته باشید که باید مقادیر زیر را در کد با توجه به نیازمندی‌های گفته شده مقداردهی کنید:

- `MEMORY_ID`: شناسه یکتای حافظه مالیاتی. رشته‌ای به طول ۶ از اعداد و حروف، قابل دریافت از قسمت عضویت کارپوشه. نمونه: `A11216`
- `API_URL`: آدرس محل API. نمونه: https://tp.tax.gov.ir/requestsmanager
- `PRIVATE_KEY_FILE`: آدرس فایل کلید خصوصی امضا به فرمت PKCS#8 و Encode شده به فرمت Base64. نمونه: `C:/private_key.pem`
- `CERTIFICATE_FILE`: آدرس فایل گواهی امضای دیجیتال به فرمت Base64. نمونه: `C:/certificate.cer`

همچنین اینترفیس دیگری به نام ILowLevelTaxApi وجود دارد که یک رابط کاربری سطح پایین‌تر (نسبت به ITaxApi) جهت ارتباط با سامانه‌ی مودیان در اختیار ما می‌گذارد. این اینترفیس فراخوانی‌کننده‌ی APIهای سامانه‌ی مودیان می‌باشد و هنگام ساخت یک Signatory دریافت می‌کند و جهت فراخوانی هر API عملیات احراز هویت (گرفتن چالش تصادفی و امضای توکن) را در ضمن آن انجام می‌دهد. نحوه‌ی ساخت ILowLevelTaxApi:

```
ILowLevelTaxApi lowLevelApi = taxApiFactory.CreateLowLevelApi(signatory);
```

تولید شماره مالیاتی

اولین گام برای تولید یک صورتحساب الکترونیک، تولید یک شماره‌ی مالیاتی می‌باشد. شماره‌ی مالیاتی رشته‌ی منحصربه‌فردی است که صورتحساب را به شکل یکتا در اکوسیستم صورتحساب الکترونیک کشور مشخص می‌کند. برای آشنایی دقیق با نحوه تولید شماره مالیاتی می‌توانید به سند «قالب شناسه یکتای حافظه مالیاتی و شماره منحصر به فرد مالیاتی» مراجعه کنید. تکه کد زیر با استفاده از توابع موجود در SDK، یک شماره سریال دریافت می‌کند و به وسیله‌ی آن (۱) و شناسه حافظه مالیاتی صادر کننده‌ی صورتحساب (۲) و تاریخ و زمان صدور صورتحساب (۳)، برای ما یک شماره‌ی مالیاتی را تولید می‌نماید:

```
TaxIdProvider taxIdProvider = new TaxIdProvider(new VerhoeffAlgorithm());
string memoryId = "MEMORY_ID";
Random random = new Random();
long randomSerialDecimal = شماره سریال غیر تکراری;
DateTimeOffset now = new DateTimeOffset(DateTime.Now);
string taxId = taxIdProvider.GenerateTaxId(
    memoryId,
    randomSerialDecimal,
    now.DateTime);
string inno = randomSerialDecimal
    .ToString("X")
    .PadLeft(10, '0');
long indatim = now.ToUnixTimeMilliseconds();
```

تولید و صدور صورتحساب

در مرحله‌ی بعدی باید یک صورتحساب تولید کنیم. در صورتی که با صورتحساب الکترونیک آشنایی ندارید به سند «دستورالعمل صدور صورتحساب الکترونیکی» مراجعه کنید. برای ارسال صورتحساب از طریق SDK باید آن را در قالب یک شیء از نوع InvoiceDto ایجاد کنیم. بدین منظور به وسیله‌ی

Constructor و مشخص کردن مقادیر فیلدهای مورد نیاز می‌توان صورتحساب را ایجاد نمود. دقت کنید که شماره مالیاتی تولید شده در قسمت قبل را که در متغیر taxId قرار داشت، به همراه سریال صورتحساب (inno) و تاریخ و زمان کنونی سیستم به عنوان تاریخ و زمان صدور صورتحساب (indatim) در Header صورتحساب قرار داده‌ایم. در ادامه نمونه کد تولید صورتحساب و مقداردهی فیلدهای دلخواه قرار داده خواهد شد:

```
InvoiceDto invoice = new InvoiceDto()
{
    Header = new HeaderDto()
    {
        taxid = taxId,
        inno = inno,
        indatim = indatim,
        inty = 1,
        inp = 1,
        ins = 1,
        tins = "14003778990",
        ...
    },
    Body = new List<BodyItemDto>()
    {
        new ()
        {
            sstid = "2710000138624",
            sstt = "سازی فولاد صنعت قطعات سرسیلندر",
            mu = "164",
            am = 2,
            fee = 10_000,
            prdis = 20_000,
            vam = 1_755,
            ...
        },
        new ()
        {
            ...
        }
    },
    Payments = new List<PaymentItemDto>()
    {
        new ()
        {
            ...
        },
        new ()
        {
            ...
        }
    }
};
```

ارسال صورتحساب

حال می‌توانیم از طریق taxApi صورتحساب را ارسال کنیم. taxApi یک متد SendInvoices دارد که لیستی از صورتحساب‌ها را گرفته و ارسال می‌کند.

```
List<InvoiceDto> invoiceList = new List<InvoiceDto>() {invoice};
List<InvoiceResponseModel> responseModels =
        taxApi.SendInvoices(invoiceList);
```

خروجی تابع SendInvoices لیستی از InvoiceResponseModelهاست که به ازای هر صورتحساب یک عضو دارد که شامل فیلدهای زیر است:

InvoiceResponseModel	
نام فیلد	**توضیحات**
Data	اطلاعات مربوط به پاسخ درخواست شما. در ابتدا که صورتحساب را می‌فرستید این مورد null است. بعدا در استعلام می‌توانید نسبت به وضعیت صورتحساب اطلاعات بیشتری کسب کنید.
Uid	شناسه یکتای مربوط به درخواست شما. این شناسه توسط خود SDK به ازای هر صورتحساب تولید می‌شود و بعدا می‌توان از آن برای استعلام وضعیت صورتحساب استفاده نمود.
ReferenceNumber	شماره پیگیری. این شماره توسط سرور هنگامی که صورتحساب را دریافت می‌کند تولید می‌شود و می‌توان از آن برای استعلام وضعیت صورتحساب استفاده نمود.
TaxId	شماره مالیاتی صورتحساب فرستاده شده.

ارسال صورتحساب از طریق ILowLevelTaxApi

برای ارسال صورتحساب از طریق ILowLevelTaxApi باید عملیات رمزنگاری و امضای صورتحساب را با استفاده از encryptor و signatory انجام دهید، سپس یک شناسه درخواست (uid) تولید کنید و سپس صورتحساب را ارسال نمایید. این عملیات در ITaxApi به صورت خودکار انجام می‌شود. نمونه کد ارسال صورتحساب با API سطح پایین:

```
ILowLevelTaxApi lowLevelApi = taxApiFactory.CreateLowLevelApi(signatory);

ITaxPublicApi publicApi = taxApiFactory.CreatePublicApi(signatory);
IEncryptor encryptor = new EncryptorFactory().Create(publicApi);

string invoiceJson = Encoding.UTF8.GetString(
    JsonSerializer.SerializeToUtf8Bytes(
        invoice, JsonSerializerConfig.JsonSerializerOptions));
string payload = encryptor.Encrypt(signatory.Sign(invoiceJson));

PacketHeaderDto packetHeaderDto = new PacketHeaderDto
{
    RequestTraceId = "RANDOM_UID",
    FiscalId = "MEMORY_ID"
};
PacketDto packet = new PacketDto
{
    Header = packetHeaderDto,
    Payload = payload
};

BatchResponseModel response = lowLevelApi.SendInvoices(new List<PacketDto>
{
    packet
});
```

lowLevelApi یک شیء API سطح پایین است که توسط taxApiFactory ساخته می‌شود. سپس از طریق همین taxApiFactory یک ITaxPublicApi می‌سازیم که با استفاده از آن یک شیء IEncryptor بسازیم. IEncryptor اینترفیسی است که عملیات رمزنگاری صورتحساب را برای ما انجام می‌دهد. سپس شیء صورتحساب را که invoice نام دارد و از نوع InvoiceDto است تبدیل به رشته‌ای به فرمت Json می‌کنیم. سپس payload را می‌سازیم که همان صورتحساب امضا شده‌ی رمز شده است. بدین صورت که ابتدا invoiceJson را با تابع signatory.Sign امضا می‌کنیم و سپس آن را با تابع encryptor.Encrypt رمزگذاری می‌کنیم. سپس باید یک PacketDto بسازیم که ورودی درخواست ارسال صورتحساب می‌باشد. بدین منظور یک PacketHeaderDto تولید می‌کنیم (شامل uid درخواست و شناسه یکتای حافظه صادرکننده صورتحساب) و آن را در کنار payload که صورتحساب امضا شده‌ی رمز شده است قرار می‌دهیم.

توضیح اینترفیس‌های ISignatory و IEncryptor

۱. اینترفیس ISignatory:

این اینترفیس برای ما بسته‌ی امضاشده‌ی JWS (طبق استاندارد API سامانه‌ی مودیان) تولید می‌کند. نحوه‌ی ساخت این اینترفیس بدین صورت است که باید از کلاس Pkcs8SignatoryFactory یا Pkcs11SignatoryFactory استفاده نماییم و کلید خصوصی و گواهی امضای بسته را در آن قرار دهیم (برای PKCS#11 باید راه ارتباطی با توکن امنیتی را در کد ایجاد نماییم که نحوه‌ی استفاده از آن در پیوست توضیح داده می‌شود. همچنین فعلا فقط از توکن امنیتی ePass3003 پشتیبانی می‌شود.) مثالی از تولید ISignatory در قسمت پیکربندی SDK (صفحه‌ی ۶) قرار داده شده است.

۲. اینترفیس IEncryptor:

این اینترفیس برای ما بسته‌ی رمزشده JWE (طبق استاندارد API سامانه‌ی مودیان) تولید می‌کند. فلسفه کار آن بدین صورت است که هر صورتحساب امضا شده توسط مودی باید برای ارسال به سامانه‌ی مودیان رمز شود و طبق استاندارد JWE و با استفاده از کلید عمومی سازمان این بسته‌ی رمزشده تولید می‌شود. مراحل ساخت IEncryptor در کد:

۱. ساخت ISignatory با گواهی امضای معتبر (به منظور احراز هویت)

۲. ساخت TaxPublicApi به منظور گرفتن کلید عمومی سامانه‌ی مودیان به وسیله‌ی متد CreatePublicApi در TaxApiFactory (این متد یک شیء ISignatory به عنوان ورودی می‌گیرد.)

۳. ساخت IEncryptor به وسیله‌ی متد Create در EncryptorFactory. این متد یک ITaxPublicApi به عنوان ورودی می‌گیرد و در درون خود متد GetServerInformation اینترفیس ITaxPublicApi را صدا می‌زند تا کلید عمومی سازمان را دریافت کند. نحوه‌ی کار متد GetServerInformation نیز به صورت زیر است:

نحوه فراخوانی ITaxPublicApi.GetServerInformation()		
ورودی	ندارد	ندارد
خروجی	Server Information Model	• serverTime: از نوع long و دارای زمان کنونی سرور • publicKeys: از نوع List<KeyModel> که هر یک دارای فیلدهای زیر است: ○ Key: کلید عمومی سرور به فرمت Base64 ○ Id: شناسه کلید ○ Algorithm: برابر با RSA ○ Purpose: برابر با ۱

استعلام وضعیت صورتحساب ارسالی

برای استعلام وضعیت صورتحساب چند راه وجود دارد:

۱. استعلام به وسیله شماره پیگیری
۲. استعلام به وسیله uid
۳. استعلام بر اساس بازه زمانی

نکته : توجه کنید بین ارسال صورتحساب و استعلام آن باید حداقل ۱۰ ثانیه فاصله باشد. برای این کار می‌توانید از تکه کد زیر استفاده کنید (این تکه باید بین کد ارسال صورتحساب و کد استعلام آن باشد):

```
Thread.Sleep(10_000);
```

استعلام به وسیله‌ی شماره پیگیری

اینترفیس ITaxApi یک متد به نام InquiryByReferenceId دارد که وضعیت صورتحساب‌های ارسال شده را بر اساس شماره پیگیری استعلام می‌کند و نحوه کار آن به صورت زیر است:

نحوه فراخوانی ITaxApi.InquiryByReferenceId		
ورودی	InquiryByRefer-enceNumberDto	دارای: • referenceNumbers: لیستی از ReferenceNumberها یا همان شماره پیگیری‌های صورتحساب ارسالی • start: شروع بازه زمانی که صورتحساب در آن قرار دارد. • end: پایان بازه زمانی که صورتحساب در آن قرار دارد.
خروجی	List<Inquiry-ResultModel>	به ازای هر شماره‌ی پیگیری، یک InquireResultModel دریافت می‌کنید که برای هر صورتحساب فیلدهای زیر را مشخص می‌کند: • referenceNumber: همان شماره پیگیری صورتحساب ارسالی • uid: شناسه درخواست ارسالی صورتحساب • status: وضعیت صورتحساب دارای حالت‌های زیر ○ PENDING: صورتحساب هنوز اعتبارسنجی نشده و در صف بررسی می‌باشد. ○ FAILED: صورتحساب ارسالی دارای خطا بوده و رد شده است. ○ SUCCESS: صورتحساب فاقد خطا بود و با موفقیت در کارپوشه ثبت شد. ○ TIMEOUT: پردازش صورتحساب بیش از اندازه طول کشیده و در کارپوشه ثبت نشد. ○ NOT_FOUND: شماره پیگیری داده شده یافت نشد. • data: جزئیات اعتبارسنجی صورتحساب ارسالی شامل موارد زیر: ○ error: لیست خطاهای صورتحساب با کد خطا و توضیحات

		○ warning: لیست اخطارهای صورتحساب با کد خطا و توضیحات ○ success: اینکه صورتحساب دارای خطا بوده یا نه (true/false) • packetType: نوع بسته • fiscalId: شناسه حافظه ارسال‌کننده‌ی صورتحساب

توجه کنید که زمان‌های start و end اختیاری است و در صورتی که در ورودی مقداردهی نشوند، به طور پیشفرض، API در ۲۴ ساعت گذشته به دنبال referenceId شما می‌گردد.

نمونه کد (ارسال صورتحساب و) استعلام به وسیله شماره پیگیری:

```
List<InvoiceResponseModel> responseModels = taxApi.SendInvoices(invoiceList);

Thread.Sleep(10_000);

DateTime startDate = DateTime.Now.AddDays(-1).ToLocalTime();
DateTime endDate = DateTime.Now.ToLocalTime();

List<string> referenceNumbers = responseModels.Select(
        r => r.ReferenceNumber
    ).ToList();

InquiryByReferenceNumberDto inquiryDto = new InquiryByReferenceNumberDto(
    referenceNumbers,
    startDate,
    endDate);

List<InquiryResultModel> inquiryResultModels =
taxApi.InquiryByReferenceId(inquiryDto);
```

invoiceList لیستی از صورتحساب‌ها می‌باشد.

خط اول صورتحساب‌ها را ارسال و پاسخ را از سرور دریافت می‌کند (شامل شماره پیگیری و uid برای هر صورتحساب).

خط دوم یک تاخیر ۱۰ ثانیه‌ای بعد از عملیات ارسال صورتحساب ایجاد می‌کند تا صورتحساب‌ها اعتبارسنجی شوند. ممکن است اعتبارسنجی یک صورتحساب بیشتر از ۱۰ ثانیه طول بکشد که در این صورت وضعیت صورتحساب پس از استعلام، PENDING خواهد بود و باید پس از چند ثانیه مجددا درخواست استعلام وضعیت صورتحساب مربوطه را ارسال کنیم.

سپس یک شیء از نوع InquiryByReferenceNumberDto می‌سازیم (که آبجکت ورودی متد استعلام صورتحساب بر اساس شماره پیگیری است) که شامل لیستی از شماره پیگیری‌ها و شروع و پایان زمان جستجو می‌باشد. برای به دست آوردن شماره پیگیری‌ها یک Mapping بر روی responseModels انجام می‌دهیم که به ازای هر InvoiceResponseModel، شماره پیگیری آن را قرار دهد و لیستی از شماره پیگیری‌ها برای ما ایجاد کند. همچنین بازه زمانی جستجو برای استعلام را از ۱ روز قبل تا امروز قرار داده‌ایم.

خط آخر هم درخواست استعلام را ارسال می‌کند و خروجی را در قالب یک لیست از InquiryResultModel ها به ما برمی‌گرداند که گزارش وضعیت هر یک از صورتحساب‌ها را نشان می‌دهد.

استعلام به وسیله‌ی uid

دقیقا همانند استعلام بر اساس شماره پیگیری است با این تفاوت که لیستی از uidهای صورتحساب‌های ارسالی به همراه شناسه حافظه صادرکننده صورتحساب می‌گیرد و نتیجه اعتبارسنجی صورتحساب‌ها با uidهای داده‌شده را برمی‌گرداند. در این متد نیز باید یک بازه زمانی به تابع داده شود که در صورت مقداردهی نشدن، به طور پیشفرض صورتحساب‌های ۲۴ ساعت گذشته مورد بررسی قرار می‌گیرند.

```
List<InvoiceResponseModel> responseModels = taxApi.SendInvoices(invoiceList);

Thread.Sleep(10_000);

List<string> uidList = responseModels.Select(r => r.Uid).ToList();
InquiryByUidDto inquiryByUidDto = new InquiryByUidDto(uidList, "MEMORY_ID",
startDate, endDate);
List<InquiryResultModel> inquiryByUidResultModels =
        taxApi.InquiryByUid(inquiryByUidDto);
```

استعلام به وسیله‌ی تاریخ

در این شیوه می‌توان وضعیت صورتحساب‌های موجود در یک بازه زمانی را به صورت صفحه‌بندی شده استعلام نمود. بدین صورت که یک شیء از نوع InquiryByTimeRangeDto می‌سازیم و به وسیله متد InquiryByTime در taxApi بر اساس آن استعلام می‌گیریم. نحوه کار این متد به صورت زیر است:

نحوه فراخوانی taxApi.inquiryByTime		
ورودی	InquiryDto	دارای موارد زیر:

		• start: تاریخ شروع بازه‌ی زمانی • end: تاریخ پایان بازه‌ی زمانی • status: وضعیت صورتحساب‌های مورد بررسی. شامل موارد زیر SUCCESS, PENDING, TIMEOUT, FAILED • pageable: تنظیمات مربوط به صفحه‌بندی نتیجه استعلام ○ از آنجایی که ممکن است تعداد صورتحساب‌ها در بازه‌ی زمانی درخواستی زیاد باشد، لازم است نتیجه به صورت صفحه‌بندی شده برگردانده شود. بدین صورت که شماره صفحه (pageNumber) و اندازه‌ی هر صفحه (pageSize) را در درخواست مشخص می‌کنیم و نتیجه استعلام‌ها بر اساس صفحه‌بندی مشخص شده به ما برگردانده می‌شود. ○ مقدار پیشفرض pageNumber = ۱. ○ مقدار پیشفرض pageSize = ۱۰ مقادیر مجاز: ۱ تا ۱۰۰.
خروجی	List<Inquiry-ResultModel>	دقیقا مانند دو مورد قبلی

در صورتی که تاریخ شروع و تاریخ پایان داده نشوند، به طور پیشفرض صورتحساب‌های ۲۴ ساعت گذشته استعلام گرفته می‌شوند. در صورتی که ورودی status داده نشود نیز صورتحساب‌ها با تمامی وضعیت‌ها برگردانده می‌شود. نمونه کد استعلام صورتحساب‌های وضعیت FAILED بر اساس تاریخ:

```
DateTime startDate = DateTime.Now.AddDays(-1).ToLocalTime();
DateTime endDate = DateTime.Now.ToLocalTime();
RequestStatus status = RequestStatus.FAILED;

Pageable pageable = new Pageable(pageNumber:1, pageSize:20);

InquiryByTimeRangeDto inquiryDto = new InquiryByTimeRangeDto(
    startDate,
    endDate,
    pageable,
    status);

List<InquiryResultModel> inquiryResult =
        taxApi.InquiryByTime(inquiryDto);
```

استعلام اطلاعات حافظه و مودی

اینترفیس TaxApi دارای دو متد به نام‌های GetTaxpayer و GetFiscalInformation می‌باشد که به ترتیب با گرفتن «شماره اقتصادی» و «شناسه یکتای حافظه مالیاتی» اطلاعات پرونده‌ی مودی (شامل نام، وضعیت، نوع مودی، شناسه ملی، آدرس و کد پستی) و اطلاعات حافظه مالیاتی (شامل نام، کد ملی، وضعیت، حدمجاز فروش و کد اقتصادی متصل به آن) را برمی‌گرداند. نحوه‌ی فراخوانی آن به شکل زیر است:

```
TaxpayerModel taxpayer = taxApi.GetTaxpayer("ECONOMIC_CODE");
FiscalFullInformationModel fiscalInformation =
        taxApi.GetFiscalInformation("MEMORY_ID");
```

پیوست‌ها

کد کامل تولید و ارسال صورتحساب در .NET. و استعلام نتیجه

```
using TaxCollectData.Library.Abstraction.Clients;
using TaxCollectData.Library.Algorithms;
using TaxCollectData.Library.Dto;
using TaxCollectData.Library.Factories;
using TaxCollectData.Library.Models;
using TaxCollectData.Library.Properties;
using TaxCollectData.Library.Providers;
using static System.Console;

namespace TaxApp;

public static class TaxClient
{
    private const string MemoryId = "A11216";
    private const string ApiUrl = "https://tp.tax.gov.ir/requestsmanager";
    private const string PrivateKeyPath = "C:/privateKey.pem";
    private const string CertificatePath = "C:/certificate.crt";

    public static void Main(string[] args)
    {
        ITaxApi taxApi = CreateTaxApi();

        InvoiceDto validInvoice = CreateValidInvoice();
        InvoiceDto invalidInvoice = CreateInvalidInvoice();

        List<InvoiceDto> invoiceList = new List<InvoiceDto>()
        {
            validInvoice,
            invalidInvoice
        };

        List<InvoiceResponseModel> responseModels =
taxApi.SendInvoices(invoiceList);
```

```
        Thread.Sleep(10_000);

        InquiryByReferenceNumberDto inquiryDto = new
InquiryByReferenceNumberDto(
            responseModels.Select(
                r => r.ReferenceNumber
            ).ToList());
        List<InquiryResultModel> inquiryResults =
            taxApi.InquiryByReferenceId(inquiryDto);

        PrintInquiryResult(inquiryResults);
    }

    private static void PrintInquiryResult(
        List<InquiryResultModel> inquiryResults)
    {
        foreach (var result in inquiryResults)
        {
            WriteLine("=======================================");
            WriteLine("Status = " + result.Status);
            WriteLine("ReferenceId = " + result.ReferenceNumber);
            WriteLine("Errors:");
            var errors = result.Data.Error;
            foreach (var error in errors)
            {
                var code = error.Code;
                var message = error.Message;
                WriteLine("Code: " + code + ", Message: " + message);
            }

            WriteLine("Warnings:");
            var warnings = result.Data.Warning;
            foreach (var warning in warnings)
            {
                var code = warning.Code;
                var message = warning.Message;
                WriteLine("Code: " + code + ", Message: " + message);
            }
        }
    }

    private static ITaxApi CreateTaxApi()
    {
        Pkcs8SignatoryFactory pkcs8SignatoryFactory =
                new Pkcs8SignatoryFactory();
        EncryptorFactory encryptorFactory = new EncryptorFactory();
        TaxProperties properties = new TaxProperties(MemoryId);

        TaxApiFactory taxApiFactory = new TaxApiFactory(
            ApiUrl,
            properties);

        ISignatory signatory = pkcs8SignatoryFactory.Create(
            PrivateKeyPath,
```

```
            CertificatePath);

        ITaxPublicApi publicApi = taxApiFactory.CreatePublicApi(signatory);
        IEncryptor encryptor = encryptorFactory.Create(publicApi);
        return taxApiFactory.CreateApi(signatory, encryptor);
    }

    private static InvoiceDto CreateValidInvoice()
    {
        Random random = new Random();
        long serial = random.NextInt64(1_000_000_000);
        DateTime now = DateTime.Now;
        string taxId = GenerateTaxId(serial, now);
        string inno = serial.ToString("X").PadLeft(10, '0');
        long indatim = new DateTimeOffset(now).ToUnixTimeMilliseconds();

        InvoiceDto invoice = new InvoiceDto()
        {
            Header = new HeaderDto()
            {
                taxid = taxId,
                inno = inno,
                indatim = indatim,
                inty = 1,
                inp = 1,
                ins = 1,
                tins = "14003778990",
                tinb = "10100302746",
                tprdis = 20_000,
                tdis = 500,
                tadis = 19_500,
                tvam = 1_755,
                todam = 0,
                tbill = 21_255,
                setm = 2
            },
            Body = new List<BodyItemDto>()
            {
                new()
                {
                    sstid = "2710000138624",
                    sstt = "سازی فولاد صنعت قطعات سرسیلندر",
                    mu = "164",
                    am = 2,
                    fee = 10_000,
                    prdis = 20_000,
                    dis = 500,
                    adis = 19_500,
                    vra = 9,
                    vam = 1_755,
                    tsstam = 21255
                }
            }
        };
```

```
        return invoice;
    }

    private static InvoiceDto CreateInvalidInvoice()
    {
        Random random = new Random();
        long serial = random.NextInt64(1_000_000_000);
        DateTime now = DateTime.Now;
        string taxId = GenerateTaxId(serial, now);
        string inno = serial.ToString("X").PadLeft(10, '0');
        long indatim = new DateTimeOffset(now).ToUnixTimeMilliseconds();

        InvoiceDto invoice = new InvoiceDto()
        {
            Header = new HeaderDto()
            {
                taxid = taxId,
                inno = inno,
                indatim = indatim,
                inty = 1,
                inp = 7,
                ins = 1,
                tins = "14003778990",
                tinb = "10100302746",
                tprdis = 30_000,
                tdis = 500,
                tadis = 19_500,
                tvam = 1_765,
                todam = 1_000,
                tbill = 21_255,
                setm = 3
            },
            Body = new List<BodyItemDto>()
            {
                new()
                {
                    sstid = "1710000138624",
                    sstt = "اشتباه کالای",
                    mu = "164",
                    am = 2,
                    fee = 10_000,
                    prdis = 20_000,
                    dis = 0,
                    adis = 19 500,
                    vra = 10,
                    vam = 0,
                    tsstam = 20_000
                }
            }
        };
        return invoice;
    }

    private static string GenerateTaxId(long serial, DateTime now)
```

```
    {
        TaxIdProvider taxIdProvider = new TaxIdProvider(new
VerhoeffAlgorithm());
        return taxIdProvider.GenerateTaxId(MemoryId, serial, now);
    }
}
```

پیکربندی SDK برای استفاده از توکن امنیتی

در صورتی که از توکن سخت افزاری epass3003 استفاده می‌کنید برای پیکربندی SDK، می‌توانید تکه کد زیر را در ابتدای پروژه خود بنویسید (توجه کنید که باید فایل dll کتابخانه‌ی ShuttleCsp11_3003 را دانلود کنید و آن را داخل پوشه دلخواه قرار دهید). این تکه کد دقیقا همانند تکه کد بالاست با این تفاوت که به منظور ساخت یک شیء Signatory (امضا کننده) از اینترفیس PKCS#11 که رابط کاربری ارتباط با توکن‌های امنیتی است استفاده می‌کند.

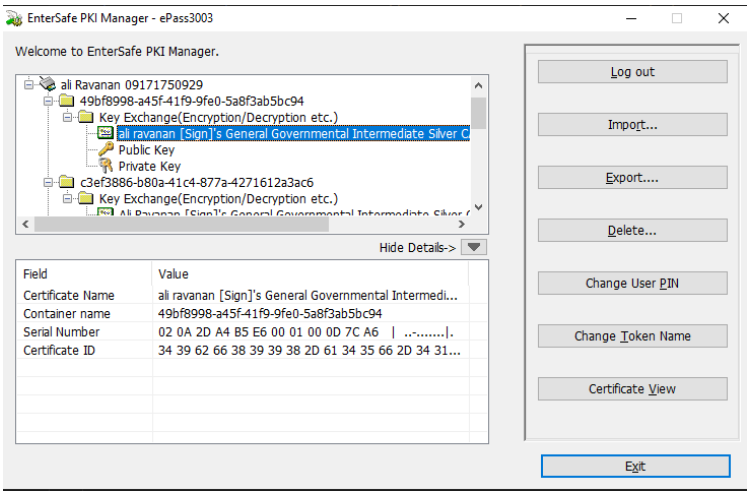
```
Pkcs11SignatoryFactory pkcs11SignatoryFactory = new Pkcs11SignatoryFactory();
EncryptorFactory encryptorFactory = new EncryptorFactory();
TaxProperties properties = new TaxProperties("MEMORY_ID");
TaxApiFactory taxApiFactory = new TaxApiFactory("API_URL", properties);
ISignatory signatory = pkcs11SignatoryFactory
    .Create("CERT_ALIAS", "LIBRARY_PATH", "PASSWORD");
ITaxPublicApi publicApi = taxApiFactory.CreatePublicApi(signatory);
IEncryptor encryptor = encryptorFactory.Create(publicApi);
ITaxApi taxApi = taxApiFactory.CreateApi(signatory, encryptor);
```

توجه داشته باشید که باید مقادیر زیر را در کد با توجه به نیازمندی‌های گفته شده مقداردهی کنید:

- MEMORY_ID: همان شناسه یکتای حافظه مالیاتی. نمونه: A11216
- API_URL: آدرس محل فراخوانی API. نمونه: https://tp.tax.gov.ir/requestsmanager
- LIBRARY_PATH: آدرس کتابخانه ShuttleCsp11_3003. نمونه: C:/ShuttleCsp11_3003.dll
- CERT_SERIAL_NUM: شناسه (Serial Number) گواهی مورد نظر (توضیح بیشتر در ادامه). نمونه: C:/ShuttleCsp11_3003.dll
- PASSWORD: پسورد توکن سخت افزاری. نمونه: 1234

استخراج Alias گواهی امضا

ابتدا نرم‌افزار ePass3003-English-1.0.17.519.exe را نصب کنید. پس از متصل‌نمودن توکن، نرم‌افزار را باز کرده و بر روی دکمه Login کلیک کرده و پسورد خود را وارد کنید. لیستی از کلیدهای توکن مطابق شکل زیر به نمایش در می‌آید:



مقدار موجود در قسمت Certificate Name گواهی مورد نظر شما همان Alias می‌باشد که می‌توانید به عنوان نام گواهی هنگام پیکربندی SDK به تابع create کلاس Pkcs11SignatoryFactory بدهید تا عملیات امضای توکن و صورتحساب را برای شما انجام بدهد.

پایان