SUBSEMBLY
Banking Apps & APIs

BankAccessServer

Installation und Betrieb

Version 1.4.2

Subsembly GmbH

Hofmannstr. 7b
81379 München

http://subsembly.com
bas@subsembly.com

Stand: 31.08.2018

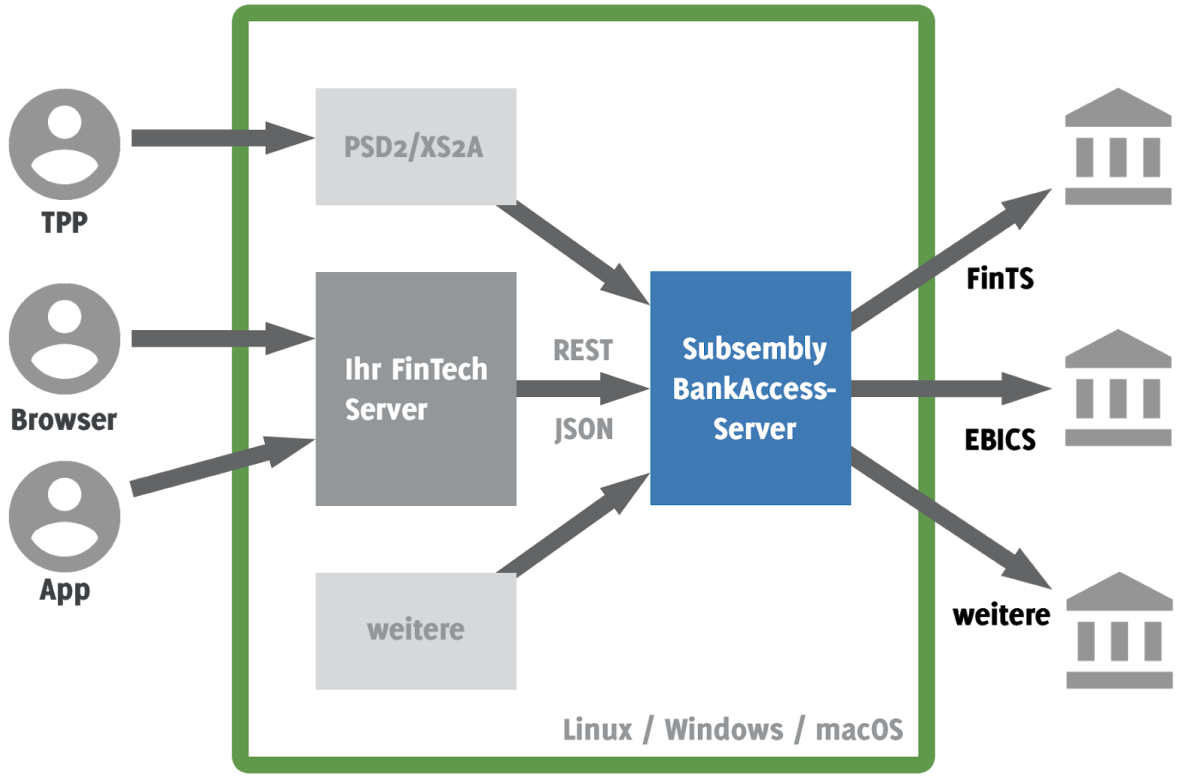
Author: Klaus Igel, Subsembly GmbH

Inhaltsverzeichnis

Einleitung

Der Subsembly BankAccessServer ist ein ASP.NET Core basierendes Servermodul für den multibanken-fähigen Zugriff auf Kontodaten und zur Übermittlung von Zahlungen via FinTS/EBICS/XS2A.

Das Softwareprodukt richtet sich an Banken und FinTech Entwickler, die hiermit eine kostengünstige Lösung für multibankenfähigen Kontenzugriff erhalten, ohne dafür einen externen Dienstleister zwischenschalten zu müssen.



Der Subsembly BankAccessServer benötigt im laufenden Betrieb keine Datenbank oder eine lokale Datenspeicherung. Alle Zugangsdaten, Session-Daten und alle Übertragungsprotokolle werden zur Speicherung direkt an den Aufrufer zurückgegeben.

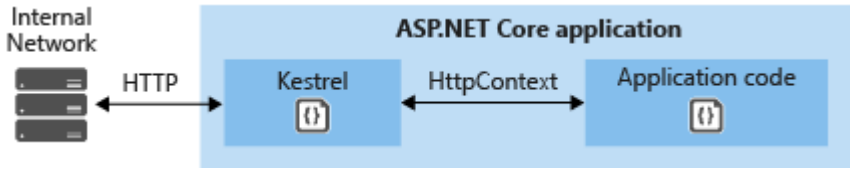
Der Installation erfolgt als lokaler HTTP Dienst auf einem Server und anwendungsseitig über eine REST API angesprochen. Die REST API ermöglicht dabei die einfache Nutzung aus beliebigen Anwendungen - unabhängig davon ob diese beispielsweise mit PHP, Java oder Node.js implementiert wurden.

Auf Grundlage des mitgelieferten Swagger-Files sind Entwickler in der Lage, die Codegenerierung anhand der API Spezifikation vorzunehmen. Ein entsprechender Client kann somit für die verschiedensten Programmiersprachen in kürzester Zeit generiert werden.

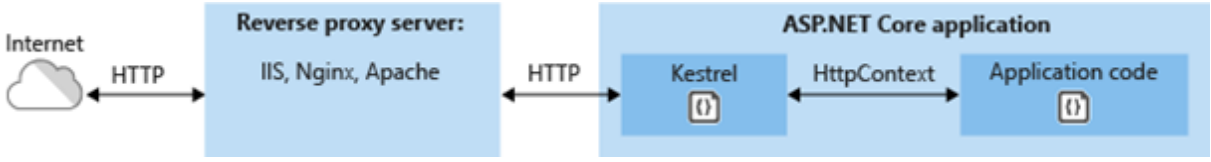
Systemvoraussetzungen

Bei dem Subsembly BankAccessServer handelt es sich um eine ASP.NET Core WebApi Applikation, die wahlweise direkt auf einem Cross-Plattform HTTP Server (Kestrel) oder hinter einem Reverse Proxy Server wie Apache, IIS oder Nginx betrieben werden kann.

Sofern der BankAccessServer nur in einem geschützten internen Environment eingesetzt werden soll, kann ein direkter Zugriff auf den Kestrel Server erfolgen.



Sofern der Zugriff auch über das Internet erfolgt ist der Einsatz eines Revers Proxy Servers empfehlenswert.



Weitere Informationen finden Sie unter folgender URL
https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/

Voraussetzung für den Betrieb des BankAccessServers ist die Installation des .NET Core SDKs oder der Runtime. Das .NET Core Runtime steht für Windows, macOS und zahlreiche Linux Distributionen zur Verfügung. Die jeweiligen Versionen und Anleitungen finden sich unter https://www.microsoft.com/net/download

Wichtiger Hinweis: Die aktuelle Version des BankAccessServers benötigt zur Ausführung das .NET Core 2 runtime.

Folgende Plattformen werden gegenwärtig unterstützt:

- Windows
- macOS
- CentOS 7
- Debian 8
- Ubuntu 14.04 / 16.04
- RHEL 7

Installation

Die Installation des Bank Access erfordert nur wenige Schritte.

- **Download der aktuellen .NET Core 2 SDKs** für die ausgewählte Plattform. https://www.microsoft.com/net/download/core#/runtime
- **Kopieren und Extrahieren der BankAccessServer Distribution** basdist.zip in ein Verzeichnis auf dem Server
- **Start des BankAccessServers** im zuvor verwendeten Verzeichnis auf dem Server dotnet BankAccessServer.dll
- **Test der Installation** anhand einer einfachen BankInfo-Abfrage.

Die oben erwähnten Schritte reichen zur Installation und direkten Ausführung des BankAccessServers aus, gleichwohl sind für den Serverbetrieb in jedem Fall grundlegende Anpassungen vorzunehmen, die je nach verwendetem Web Server und Betriebssystem im Detail abweichen können.

Nachfolgend wird die komplette Installation eines BankAccessServers auf einer Standard CentOS 7 Installation dargestellt.

Installation des aktuellen .NET Core 2.0 SDKs für CentOS 7

Vor der Installation von .NET Core muss einmalig der Microsoft Product Feed registriert werden.

```
sudo rpm -Uvh
https://packages.microsoft.com/config/rhel/7/packages-microsoft-prod.rpm
```

Anschließend werden die für .NET Core 2 erforderlichen Updates/Komponenten sowie .NET Core selbst installiert.

```
sudo yum update
sudo yum install libunwind libicu
sudo yum install dotnet-sdk-2.1.202
echo 'export PATH=$PATH:$HOME/dotnet' > /etc/profile.d/dotnetdev.sh
sudo ln -s /usr/share/dotnet/dotnet /usr/local/bin
```

Anleitungen für die jeweiligen Linux-Distributionen finden sich hier:
https://www.microsoft.com/net/download/linux-package-manager/centos/sdk-2.1.202

Das Kommando `dotnet --info` sollte nun erfolgreich ausgeführt werden können und die installierten Runtimes/SDKs auflisten:

```
.NET Command Line Tools (2.1.2)

Product Information:
 Version:            2.1.202

Runtime Environment:
 OS Name:     centos
 OS Version:  7
 OS Platform: Linux
 RID:         centos.7-x64
 Base Path:   /usr/share/dotnet/sdk/2.1.202/

Microsoft .NET Core Shared Framework Host

  Version  : 2.0.9
```

Als nächstes Kopieren und Extrahieren wir die Distribution des BankAccessServers in das Verzeichnis /opt/bas .

Als letzten Schritt wird nun der BankAccessServer gestartet:

`cd /opt/bas/`

`dotnet BankAccessServer.dll`

Da noch keine Konfigurationsanpassungen vorgenommen wurden, startet der BankAccessServer auf Port 5000.

```
Hosting environment: Production
Content root path: /opt/bas listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Bevor im weiteren Verlauf die Installation verfeinert wird erfolgt jetzt ein initialer Test, in dem in einer neuen Konsole folgender Request für eine IBAN Konvertierung auf Basis einer nationalen Kontonummer aufgerufen wird:

```
curl -X POST "http://localhost:5000/api/info" -H "Content-Type:
application/json" -d "{'Orders': [{
'Type':'GetAccountInfoFromNationalAccountRequest', 'OrderId' :
'1','AcctNo': '18', 'BankCode' : '12030000' }]}"
```

Sofern alles korrekt lief sollte die Ausgabe wie folgt aussehen:

`{"Responses":[{"Type":"GetAccountInfoFromNationalAccountResponse","OrderId":"1","BIC":"BYLADEM1001","AcctNo":"18","BankCode":"12030000","BankName":"Deutsche Kreditbank Berlin","IBAN":"DE98120300000000000018","CountryCode":"DE","ValidationCode":0,"ValidationMessage":"OK"}]}`

Vorerst können wir nun den BankAccessServer wieder stoppen und mit der Verfeinerung der Installation fortfahren.

Empfehlenswert ist in jedem Fall der Einsatz eines Proxy Servers, der die eingehenden Requests an den BankAccessServer weiterleitet. Im folgenden Beispiel wird simplifiziert davon ausgegangen, dass der Apache Web Server bereits installiert ist und kein SSL benutzt. Im ersten Schritt legen wir die Datei `/etc/httpd/conf.d/bas.conf` mit folgendem Inhalt an:

```
# forward dynamic requests to kestrel
<VirtualHost *:80>
  ProxyRequests Off
  ProxyPreserveHost On
  <Proxy *>
    Order deny,allow
    Allow from all
  </Proxy>
  ProxyPass /bankaccess http://localhost:5000
  ProxyPassReverse /bankaccess http://localhost:5000
</VirtualHost>
```

Somit werden alle http-Requests für /bankaccess direkt an den Kestrel Server, der in einer Produktionsumgebung nicht von außen erreichbar sein sollte, weitergeleitet.

Im Anschluss wird die Konfiguration getestet: `sudo service httpd configtest`
Die Ausgabe sollte wie folgt aussehen: `Syntax OK`

Zum Abschluss wird der Apache Web Server neu gestartet.

```
service httpd stop
service httpd restart
```

Im nächsten Schritt wird ein Startskript `/opt/bankaccess.sh` angelegt:

```
cd /opt/bas/

dotnet BankAccessServer.dll
```

Damit der BankAccessServer automatisch gestartet werden kann erstellen wir ein Service-File unter /systemd/system/kestrel-bankaccess.service mit folgendem Inhalt:

```
[Unit]
Description=Subsembly BankAccessServer

[Service]
ExecStart=/usr/bin/sh /opt/bankaccess.sh
Restart=always
RestartSec=10
SyslogIdentifier=dotnet-bankaccess
User=root
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target
```

Hinweis: Das Environment (Produktion | Development) kann über das Service-File gesteuert werden.

Environment=ASPNETCORE_ENVIRONMENT=Production

Environment=ASPNETCORE_ENVIRONMENT=Development

Anschließend wird der Service noch aktiviert:

```
systemctl enable kestrel-bankaccess.service
```

Output:

```
Created symlink from
/etc/systemd/system/multi-user.target.wants/kestrel-bankaccess.servi
ce to /etc/systemd/system/kestrel-bankaccess.service.
```

Im letzten Schritt wird der BAS Service gestartet und verifiziert:

```
service kestrel-bankaccess start

service kestrel-bankaccess status
```

Output:

```
● kestrel-bankaccess.service - Subsembly BankAccessServer
   Loaded: loaded (/etc/systemd/system/kestrel-bankaccess.service;
enabled; vendor preset: disabled)
   Active: active (running) since Thu 2017-10-05 18:54:37 CEST; 5s
ago
 Main PID: 107072 (dotnet)
   CGroup: /system.slice/kestrel-bankaccess.service
           └─107072 /usr/local/bin/dotnet /opt/bas/BankAccessSe...

Oct 05 18:54:37 linux systemd[1]: Started Subsembly
BankAccessServer.
Oct 05 18:54:37 linux systemd[1]: Starting Subsembly
BankAccessServer...
Oct 05 18:54:38 linux bas[107072]: Hosting environment: Production
Oct 05 18:54:38 linux bas[107072]: Content root path: /opt/bas/
Oct 05 18:54:38 linux bas[107072]: Now listening on:
http://localhost:5000
Oct 05 18:54:38 linux bas[107072]: Application started. Press Ctrl+C
to shu...n.
Hint: Some lines were ellipsized, use -l to show in full.
```

Test der Reverse Proxy Kommunikation:

```
curl -X POST "http://localhost/bankaccess/api/info" -H
"Content-Type: application/json" -d "{'Orders': [{
'Type':'GetAccountInfoFromNationalAccountRequest', 'OrderId' :
'1','AcctNo': '18', 'BankCode' : '12030000' }]}"
```

Somit kann der BankAccessServer über den vorgelagerten Apache Webserver erreicht werden.

Linux: Besonderheiten bei der Generierung von QR-Codes

Sofern der BankAccessServer unter Linux betrieben wird und für die Generierung von EPC-069 QR-Codes verwendet werden soll, ist die Installation von **libgdiplus** notwendig. Hinweise zur Installation sind unter folgender URL zu finden:

https://github.com/zkweb-framework/zkweb.system.drawing

Konfiguration

Im Installationsverzeichnis des BankAccessServers finden sich json-basierte Konfigurationsdateien zur Steuerung des Loggings, der Server-URLs sowie zur Lizenzierung und Zugriffssteuerung.

Config-Files

appsettings.json - Anwendungseinstellungen, wenn der BankAccessServer im Production Mode betrieben wird:

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "System": "Warning"
    },
    "LogToConsole": true,
    "LogToFile": true,
    "PathFormat": "Logs/BankAccessServer-{Date}.log"
  }...
}
```

appsettings.Development.json - Anwendungseinstellungen, wenn der BankAccessServer im Development Mode betrieben wird:

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    },
    "PathFormat": "Logs/BankAccessServerDevelopment-{Date}.log"
  }...
}
```

hosting.json - steuert die Server URL, über die der BAS erreichbar ist

```
{
  "server.urls": "http://localhost:5000"
}
```

Lizenzierung

Für den Einsatz des BankAccessServers wird eine gültige Lizenz benötigt, die Sie beim Kauf oder im Rahmen einer Teststellung direkt von uns erhalten.

Der Lizenzschlüssel wird in der Datei appsettings.json im Abschnitt “BasConfig” definiert:

```
"BasConfig": {
      "License": "yourLicenseKey",
      ...
}
```

Zugriffssteuerung

Ungeachtet einer abweichenden Konfiguration des vorgeschalteten Proxy Servers können nach einer Standardinstallation des BankAccessServers Requests für alle verfügbaren Endpoints ausgeführt werden.

Für die Definition individueller Zugriffsrechte kann der BankAccessServer mit individuellen API Keys, die wiederum auf bestimmte Endpoints beschränkt werden können, erstellt werden.

Ob bei den Requests die Gültigkeit von API Keys geprüft werden soll kann in der Datei appsettings.json im Abschnitt “BasConfig” definiert werden:

```
"BasConfig": {
      ...
      "CheckAccessToken": true | false
}
```

Im Lieferumfang ist ein Tool zur Generierung von AccessTokens enthalten, das wie folgt aufgerufen werden kann:

```
dotnet BasAccessTokenGenerator.dll -<argname> <argvalue> ...
```

Folgende Werte für <argname> sind möglich:

```
license     License Token aus appsettings.json
id          Mapping zu gespeicherten EBICS Kontakte/Schlüsseln
name        Name des Access Tokens, z.B. Username
year        Ablaufjahr <yyyy>
month       Ablaufmonat <m>
day         Ablauftag <d>
module      Module-Name: fints | ebics | xs2a | sepa | info.
rights      Berechtigung: ebics-admin | ebics-download | ebics-upload
```

Die Ausgabe kann getrennt in Daten (out.txt) und Protokollierung

(log.txt) wie folgt umgeleitet werden:

```
dotnet BasAccessTokenGenerator.dll ... >out.txt 2>log.txt ...
```

Im nachfolgenden Beispiel wird für die angegebene BankAccessServer Lizenz ein Zugriffstoken erstellt, das bis zum 01.01.2020 gültig ist, den Zugriff auf die Endpoints fints, ebics, sepa und info erlaubt und EBICS Downloads ermöglicht:

```
dotnet BasAccessTokenGenerator.dll -license <yourLicenseToken> -name
"Klaus Igel" -id "Subsembly" -year 2020 -month 1 -day 1 -module
ebics -module fints -module sepa -module info -rights ebics-download
>out.txt 2>log.txt
```

Die Ausgabe wird in die Datei out.txt umgeleitet:

```
license [yourLicencsToken]
name [Klaus Igel]
id [Subsembly]
year [2020]
month [1]
day [1]
module [ebics]
module [fints]
module [sepa]
module [info]
rights [ebics-download]
---AccessToken---
<generated access token>
```

```
---AccessToken---
```

Das generierte AccessToken kann dann bei entsprechenden Requests angegeben werden.

```
{
      "AccessToken":"XXX",
      …
}
```

Verschlüsselung von Sessions und serialisierten Kontakten

Rückgabewerte, wie Session Tokens und serialisierte Kontakte werden mit dem angegebenen Passwort verschlüsselt.

```
"BasConfig": {
      ...
      "EncodePassword":"password"
}
```

Somit können diese Daten nicht unbefugt auf anderen Servern für die Durchführung weiterer Aktionen verwendet werden, solange das Passwort zwischen den Serverinstanzen nicht übereinstimmt. Sofern mehrere BankAccessServer über einen Load Balancer angesprochen werden ist es sinnvoll, bei allen Servern dasselbe Passwort zu verwenden, um alle Folgeaktionen auf jedem Server innerhalb der Serverfarm abwickeln zu können.

Speichern von EBICS Kontakten und temporären Dateien

Der BankAccessServer unterstützt den kompletten Workflow zur Einrichtung von EBICS Zugängen. Die Speicherung der Zugänge/Schlüssel kann wahlweise durch den BankAccessServer vorgenommen oder bei jedem Request mitgeschickt werden.

Sofern die EBICS Kontakte/Schlüssel auf dem BankAccessServer gespeichert werden sollen, kann hierzu das zu verwendende Verzeichnis angegeben werden. Darüber hinaus kann ein temporäres Verzeichnis angegeben werden, in dem temporäre Dateien im Rahmen der EBICS Zugangseinrichtung gespeichert werden.

```
"BasConfig": {
   ...
    "EbicsStoragePath": "EbicsContacts/",
    "EbicsTempPath": "EbicsTemp/"
}
```

XS2A Konfiguration

Verfügbare Endpoints

Grundsätzlich können folgende Schnittstellen der Banken/Zahlungsdienstleister für den Kontozugriff über den XS2A Endpoint verwendet werden:

- FinTS/HBCI - Standard-Online-Banking-Schnittstelle in Deutschland, unterstützt von den meisten Banken.
- PSD2 XS2A - Offizieller PSD2-konformer Zugang zu Account-Schnittstellen, die von Account-Service-Providern für Kontoinformationsdienste zur Verfügung gestellt werden. Der standarisierte PSD2 konforme Zugriff ist derzeit auch seitens der Anbieter noch nicht verfügbar und daher nur konzeptionell vorgesehen.
- API - Jede proprietäre API, die vom Zahlungsdienstleister für den Zugriff auf Kontoinformationen zur Verfügung gestellt wird.
- Screen Scraping - Wenn keine der oben genannten Schnittstellen oder APIs verfügbar ist, wird auf das Screen-Scraping der Webseiten zurückgegriffen, um die Kontoinformationen zu sammeln.

Die Konfiguration des XS2A Endpoints wird in der Datei appsettings.json im Abschnitt “Xs2aConfig” definiert:

```
"Xs2aConfig": {
  "RegisterBuiltInScrapers" : true
}
```

Sofern die Zugriffsart FinTS/HBCI auch für den XS2A Endpoint genutzt werden soll, ist für “RegisterBuiltInScrapers” der Wert “true” anzugeben.

Session Handling

Darüber hinaus kann auch das Session Handling für den XS2A Endpoint angepasst werden:

```
"Xs2aConfig": {
    "RespondWithSessionObjectWhenSuspend": true|false,
    "SessionCleanInterval": <ms>,
    "SessionTimeout": <ms>
  }
```

RespondWithSessionObjectWhenSuspend - Sofern mit Sessions gearbeitet wird kann festgelegt werden ob das gesamte Session Objekt (true) oder nur die Session Id (false) in der Response zurückgeliefert wird.

SessionCleanInterval - Intervall in ms, in dem auf dem Server geprüft wird ob abgelaufene Sessions entfernt werden können.

SessionTimeout - Wert in ms, nach dem inaktive Sessions entfernt werden.

Connection Handling

Die maximale Anzahl von gleichzeitig erlaubten Verbindung kann über folgenden Eintrag gesteuert werden:

```
"BasConfig": {
     "MaxConConnect":  32
}
```

Wichtig: Die Anzahl bezieht auf den gesamten BankAccessServer und nicht auf einzelne Endpoints/Module.

Bitte achten Sie darauf, dass der Wert in jedem Fall geringer ist als die maximale Anzahl Threads im ThreadPool (siehe BankAccessServer.runtimeconfig.json im Programmverzeichnis).

Protokollierung

Die Protokollierung kann wahlweise auf der Konsole und/oder in Dateien erfolgen. Zudem kann die Ausführlichkeit der Protokollierung festgelegt werden.

Für die einzelnen Requests werden eindeutige Request IDs geloggt, so dass auch eine Zuordnung der jeweiligen Aufträge zu den Requests in den Log-Dateien erfolgen kann.

Updates

Bei der Bereitstellung von Updates unterscheiden wir zwischen folgenden Update-Typen:

1. Update der Bankinformationen-/Zugangsdaten - erfordert den Austausch der von bereitgestellen FinBanks.csv Datei, die im Installationsverzeichnis unter /Data gespeichert wird.
2. Update der Programmdateien - Austausch der BankAccessServer.dll iund ggfs. weiterer Dateien im Installationsverzeichnis
3. Einspielen der gesamten Dateien des BankAccessServers inkl. Abhängigkeitsdateien - hierzu gibt es separate Updatehinweise, die ggfs. auch eine Aktualisierung der Laufzeitumgebung voraussetzen können.

Hinweis: Nach Einspielen der Updates muss der BankAccessServer neu gestartet werden.

Weitere Informationen

Subsembly BankAccessServer

Webseite: https://subsembly.com/bank-access-server.html

Produktinformationen: https://subsembly.com/download/BankAccessServer.pdf

API Spezifikation: https://subsembly.com/download/BankAccessServerClientInterface.pdf

Subsembly Banking APIs / SDKs

FinTS API: https://subsembly.com/sepa-api.html

Ebics API: https://subsembly.com/ebics-api.html

SEPA API: https://subsembly.com/sepa-api.html

XS2A API: https://subsembly.com/xs2a-api.html

Spezifikationen

Subsembly Payments Datenformate (SUPA): https://subsembly.com/supa.html

Deutsche Kreditwirtschaft / Ebics:
https://die-dk.de/zahlungsverkehr/electronic-banking/dfu-verfahren-ebics/

Deutsche Kreditwirtschaft / FinTS:

https://die-dk.de/zahlungsverkehr/electronic-banking/fints/

Deutsche Kreditwirtschaft / PSD2 Kontoschnittstelle:

https://die-dk.de/zahlungsverkehr/electronic-banking/psd2-kontoschnittstelle/

NextGenPSD2 Access to Account Interoperability Framework

Berlin Group / Market Consultation

https://www.berlin-group.org/market-consultations

Berlin Group / PSD2 XS2A Introduction

https://docs.wixstatic.com/ugd/c2914b_722a8136e12d4417abeea85dc47c5555.pdf?index=true

Laufzeitumgebung

Microsoft .NET Core 2: https://www.microsoft.com/net/download

Codegenerierung

Swagger CodeGen: http://swagger.io/swagger-codegen/