INFOSYS - UŽICE

ИНФО АПИ СЕРВЕР В1

УПОТРЕБА
АПИ ФУНКЦИЈА

верзија 1.11
(усклађено са InfoAPI v1.73)

Пеђа Супуровић
Ужице, мај 2014. – новембар 2023.

УВОД

Инфо АПИ сервер је универзалан модул који омогућава приступ Инфо програму и бази података удаљено, преко веб сервиса за све врсте клијената.

АПИ сервисом управља клијент кроз параметре које му шаље. Упит сервису се шаље *HTTP* протоколом у *JSON* формату, а предвиђена је могућност да се користе и други формати, на пример XML али то за сада није подржано.

АПИ протокол је осмишљен тако да је лако применљив на великом броју платформи на којима се клијенти развијају или употребљавају.

За шта се овај протокол може употребити

ИнфоАПИ протокол је осмишљен као универзални протокол који треба да омогући да било која екстерна апликација може да се повеже на информациони систем *Infosys* и да са њим размењује податке у оба смера.

То на пример могу бити:

- рад комерцијалиста на терену (слање поруџбина, ажурање стања артикала и података о комитентима и слично);

- попуњавање докумената употребом спољних програма (документи магацина, документи производње, пописи и друге евиденције)

- самостално попуњавање поруџбина од стране клијената;

- приступ пословним подацима за потребе руковођења предузећем;

- ажурирање података између пословних јединица (продавнице, откупна места и слично);

- повезивање разних специфичних пословних апликација које нису производ *Infosys*-а са Infosys programom (комерцијалом, рачуноводством, књиговодством и другим пословним евиденцијама у *Infosys* програму);

- омогућавање размене података са спољним системима (веб продавнице и слично);

- аквизиција података са специфичних уређаја и њихов упис у евидецију *Infosys* програма (ваге, откупна места, очитавање водомера и слично)

АПИ функције

ИнфоАПИ протокол омогућава комункацију са АПИ функцијама. Сам протокол је универзалан, а АПИ функције су специфичне и Инфосус их развија у договору са својим корисницима.

АПИ функција може да прима одређене податке, да врши одређене обраде и да враћа неке податке клијенту који ју је позвао. Параметри, начин рада и резултати АПИ функције се дефинишу декларацијом функције. Декларација је документ који Инфосис сачињава у договору са својим корисником.

Начин позивања АПИ функција је увек исти и дефинисан је ИнфоАПИ протоколом. То значи да клијентска апликација која има имплементирану подршку са ИнфоАПИ протокол у теорији може да позива било коју АПИ функцију.

Подржане платформе

ИнфоАПИ проток је пројектован тако да је практично применљив на великом броју платворми. Употреба протокола није условљена врстом уређаја (рачунари, таблети, мобилни телефони..) ни оперативним системом (*Windows*, *Linux*, *Android*, *iOS*...), ни развојним платформама (.*NET*, *PHP*, *Java*, *FoxPro*, *Borland*...).

Практично свака платформа са које може да се комуницира *HTTP* протоколом и чита и пише *JSON* формат може се употребити за комуникацију са ИнфоАПИ веб сервисима.

Документација

Ажурна докумнција за ИнфоАПИ се налази на https://podrske.infosys.rs/datoteke/infoapi. Она прати измене и унапређења програма.

САДРЖАЈ

Увод **3**

За шта се овај протокол може употребити 3

АПИ функције 3

Подржане платформе 4

Документација 4

Комуникација са АПИ сервером **9**

Адреса сервера 9

Садржај упита 9

JSON формат 9

Компресија података 10

Мере сигурности 10

Структура АПИ упита **11**

Типови 11

Упити и одговори 11

Структура порука АПИ упита 11

TAPIQueryEnvelope 12

TAPIParams 13

Структура порука АПИ одговора **15**

TAPIResponseEnvelope 15

TAPIResult 16

Обрада грешака **19**

Типови статуса грешака 19

Статус типа ОБАВЕШТЕЊЕ 19

Статус типа КРИТИЧАН 19

Шта све ЧИНИ опис грешке 19

error_status (char) 19

error_message (char) 20
error_data (obj) 20
Листа описа статуса грешака 20
Системски статуси грешака 21
Системски статуси грешака на TAPIResponseEnvelope 21
Системски статуси грешака на TAPIResult 22
Контролни код 23
Лиценцирање 25
Лиценца клијентске апликације 25
Лиценца АПИ сервера 25
Посебне структуре података 27
DatabaseAttachment 27
PLAIN_OBJ формат 27
PLAIN_OBJ_BASE64 формат 28
NATIVE_OBJ формат 28
SQLITE формат 28
ZIPSQLITE формат 28
ZIPDBF формат 28
Attachment 29
Изрази 29
Подржани изрази 30
Страничење резултата 33
Укључивање страничења резултата 33
Употреба страничених података 33
Сесије и инстанце 35
Шта је сесија 35
Инстанца у сесији 35
Када користити сесију а када инстанцу? 36
Када треба користити сесије 36

Сесије треба користити ако: ... 36
Како се ради без сесија ... 36
Пријава корисника ... 37
Одјава корисника ... 37
Одјавом корисника ... 37
Поновном пријавом корисника ... 37
Истеком сесије ... 37
Примери JSON СТРУКТУРА ... 39
Апи функција NONE ... 39
АПИ функција INFO ... 39
АПИ функција GETTABLE ... 39
АПИ функција GETTABLE SQLite формат ... 39
АПИ функција COMPLEXERROR ... 40
Брзи пример ... 41
креирај упит ... 41
Анализирај одговор ... 41
Шта даље ... 43
Честе грешке ... 45
JSON формат ... 45
Обрада грешака ... 45
Консултације са Инфосис подршком ... 46
Пренос табеларних података ... 46
Имплементација подршке за ИнфоАПИ протокол ... 46
Форматирање датума ... 46

КОМУНИКАЦИЈА СА АПИ СЕРВЕРОМ

Адреса сервера

АПИ сервер је веб апликација са којом клијенти комуницирају *HTTP* протоколом. Упит се шаље на адресу *http://<адресасервера>:<port>/api/*.

Адреса сервера је мрежна адреса на којој се налази сервер. То може бити IP адреса или домен. Адреса сервера треба да буде статичка (фиксна). Уколико се серверу приступа само из локалне мреже, онда адреса може бити локална али ако је потребно да се серверу приступа са Интернета онда адреса мора бити јавна. Не препоручујемо употребу динамички додељиваних ИП адреса нити динамички ДНС сервис, осим за потребе тестирања.

Порт сервера је подразумевано 31000. Могуће је на истом рачунару подићи више сервера са различитим подешавањима (обично ради приступа Инфо бази на другом диску). Други сервери се подешавају на другим портовима. Уколико се серверу приступа са Интернета, а сервер је иза рутера или неког другог заштитног зида (firewall), онда је потребно подесити да се порт са јавне ИП адресе исправно усмерава на сервер.

АПИ сервер подржава *GZIP* компресију на нивоу *HTTP* протокола и препоручујемо да на клијенту буде укључено коришћење ове компресије због оптималног преноса података.

Садржај упита

ИнфоАПИ комуникација је знатно поједностављена верзија *RESTful* протокола. Разлика је у томе што се не употребљавају идентификатори ресурса, и употребљава се само ХТТП метод. Сви потребни подаци се налазе у *JSON* поруци.

HTTP упитом се подаци прослеђују *POST* методом, који садржи следеће променљиве:

json (char)	*C*адржи АПИ упит у *JSON* формату према спецификацији ИнфоАПИ сервера.
debuglevel (int)	Ниво дибаг логовања. Ако дибаг лог није потребан, треба да буде нула (или празно)
debugsection (char)	Листа дибаг секција које треба укључити у лог. Ако дибаг није потребан, треба да буде празно.
apiversion (char)	Ознака верзије АПИ-ја коју треба покренути за дати упит. Клијенти који раде по протоколу верзије 1 или новије треба да пошаљу верзију протокола или, боље, верзију АПИ-ја на којој је клијент развијан (на пример 1.18). Ако се не пошаље верзија протокола биће подразумевано покренута верзија 0.

JSON формат

JSON (*JavaScript Object Notation*) је отворени стандард за размену података који је поред рачунара, читак и људима. Ради се о текстуалном формату који може да садржи комплексне структуре података. Подржан је у великом броју савремених програмских језика. Има предност у односу на *XML* јер је читкији и економичнији формат који је лакше програмски анализирати.

Пример једноставне *JSON* структуре:

```
{
   "property_name1": "value1",
   "property_name2": "value2",
   "property_name3": "value3",
}
```

Својства у *JSON* струкури могу садржавати податке разних типова па и сложене типове као што су објекти, низови и листе. То омогућава размену података веома разноврсних струкура.

Више о овом формату можете наћи на http://www.json.org/.

Компресија података

Инсталирани ИнфоАПИ модул у себи садржи и веб сервер. Сервер подржава *gzip* компресију тако да ако клијенти у *HTTP* упиту пријаве да и они подржавају компресију добиће одговор у компресованом формату. Препоручљиво је да се користи компресовани формат јер се тиме може знатно скратити време трајања преноса података.

Када изврши тражену функцију АПИ сервер враћа резултат као *JSON* документ.

Мере сигурности

ИнфоАПИ протокол обезбеђује сигурност података на нивоу аутентификације корисника. На овај начин се контролишу права приступа подацима.

Препоручен начин заштите је да се користи *VPN* (*Virtual Private Network*) ако се АПИ користи за интерне послове (веза комерцијалиста или аквизитера на терену са централом, веза удаљених радних јединица са централом и слично).

Ако се ИнфоАПИ сервер изкаже јавном делу Интернета, препоручено је да се за приступ користи https протокол, на пример, употребом реверзног проксија.

Уколико се не користе друге врсте заштите, јако се препоручује да се приликом аутентификације употребљава AES256 енкрипција лозинке.

СТРУКТУРА АПИ УПИТА

ТИПОВИ

У документу ћемо користити следеће просте типове:

char	низ знакова (*string*)
int	целобројна вредност
num	број, може бити целобројни или са децималном тачком
datetime	датум и време у *ISO* 8601 (YYYY-MM-DDTHH:MM:SS+HH:MM, где је време изражено у локалној временској зони а +HH:MM означава која је локална временска зона). Време не може садржавати делове секунде.
bool	логичка вредност (*true*, *false*)
obj	објекат, односно нека сложена структура

Поред ових простих типова, структура може садржавати и сложене типове: објекте, листе, низове и друго.

Такође протокол дефиниш еи неке сложене типове који су специфични за сам протокол, на пример тип за структуру параметара, структуру резултата упита, структуру за опис статуса извршења (поруке о грешкама), структуру за запис табеларних података, структуру за запис прикачених датотека и друге структуре.

Сложени типови су објашњени касније у овом упутству на местима где се објашњава и њихова употреба.

УПИТИ И ОДГОВОРИ

Поруке које се размењују могу бити упити и одговори. Упит шаље клијентска апликација тражећи од АПИ сервера да изврши жељену АПИ функцију и прослеђујући јој улазне податке, а одговор са резултатима извршења фунције враћа сервер након што АПИ функцију изврши.

Сам садржај упита и одговора је веома сличан и поруке се могу обрађивати на исти начин.

СТРУКТУРА ПОРУКА АПИ УПИТА

JSON упит се састоји из два слоја: оквир (*TAPIQueryEnvelope*) и параметри (*TAPIParams*).

Подаци се смештају у својства наведених класа и могу бити прости типови, низови или објекти. Уколико неко својство нема вредност или има подразумевану вредност, то својство не мора постојати у JSON формату.

TAPIQUERYENVELOPE

Оквир (*TAPIQueryEnvelope*) је универзалан део JSON упита који је непроменљив без обзира на врсту упита. Његова улога је да обезбеди комуникацију између клијента и АПИ сервера. Он у себи садржи угњежден слој који садржи параметре позваној АПИ функцији.

Структура :

req_id (char)	Идентификатор упита. Овај податак генерише клијент а сервер га враћа у одговору. Служи да клијент може да по потреби повеже одговоре са упитима и за рачунање контролног кода за проверу целевитости послатог упита. Клијент треба да употребљава различиту вредност за сваки упит који шаље. Ово је обавезан податак.
session_id (char)	Идентификатор сесије. Овај идентификатор генерише сервер када клијент први пут позове АПИ и враћа га клијенту. Клијент у сваком следећем упиту у оквиру исте сесије шаље идентификатор сесије. Ово омогућава да се одређени подаци или стања могу преносити између више упита са истог клијента ка АПИ-ју. Сесија се прекида одјавом или неактивношћу клијента. Подразумевано, некативност дужа од 10 минута прекида сесију, а то се може подешавати на АПИ северу. Ако клијент не зна идентификатор сесије, односно тек започиње сесију, шаље ово поље празно.
instance_id (char)	Идентификатор инстанце у оквиру сесије. Овај идентификатор сервер генерише ако није добио идентификатор од клијента. Клијент по правилу у сваком наредном упиту шаље идентификатор инстанце који је добио од сервера осим ако не жели да започне нову инстанцу. Клијент може да покрене више инстанци у оквиру исте сесије. Све инстанце се затварају са затварањем сесије.
encoding (char)	Начин на који су кодирани подаци у пољу *data*. Сервер на основу овог податка зна како да декодира податке. Подразумевано се подаци не кодирају и ово поље остаје празно. Енкодирање података није још увек имплементирано.
app_option (char)	Ознака АПИ функције коју клијент позива. Ово је податак на осниву кога АПИ сервер зна коју функцију треба да изврши и какве параметре треба да јој проследи. Ово је обавезан податак.
data (char)	Параметри који се шаљу АПИ функцији. Ово поље садржи угњеждену JSON структуру *TAPIParams*. Та структура је променљива зависно од функције која се позива.
username (char)	Корисничко име за пријаву корисника АПИ серверу. Употребљава се заједно са пољем *password*. Навођењем корисничког имена и лозинк еу упиту креира се нова сесија (ако је клијент претходно имао отворену сесију са АПИ сервером та сесија ће бити затворена) и врши проверу привилегија и права приступа наведеног корисника. Када се клијент пријави са корисничким именом и лозинком, биће отворена нова сесија и надаље клијент у упитима не наводи корисничко име и лозинку већ идентификатор сесије (и идентификатор инстанце), осим ако жели да раскине постојећу сесију и започне нову.

password (char)	Лозинка која се употребљава за пријаву корисника, заједно са *username*.
auth_type (char)	Тип аутентификације. - PLAIN (или празно) - лозинка се шаље отворено. - AES256 - лозинка се енкриптује по AES256 протоколу. Подразумевано је PLAIN али јако препоручујемо да се лозинка енкриптује, нарочито ако се не користе друг мере заштите (VPN, https)
client_id (char)	Ознака клијентског програма (везано за лиценцу). Ово је обавезан податак.
client_version (char)	Ознака верзије клијентског програма
api_version (char)	Ознака верзије АПИ-ја који клијент разуме. Навести конкретну верзију на којој је клијент развијан, на пример 1.18)
poslovna_godina (int)	Ознака пословне године. Употребљава се ако клијент треба да наведе АПИ-ју коју пословну годину жели да употреби. Подразумевано је поље празно, што значи да ће АПИ употребљавати податке из подразумеване пословне године. Подразумевана пословна година је текућа календарска година, осим ако у конфигурацији сервиса није подешено да то буде нека друга. Овај параметар може бити и стринг типа – АПИ ће га конвертовати у нумеричку вредност. Клијентска апликација треба да пошаље пословну годину ако је потребно да АПИ функцију изврши над подацима у пословној години која није подразумевана, на пример да упише неки документ у претходну пословну годину у периоду преласка из старе у нову годину.
job (char)	Ознака *JOB*-а. Употребљава се ако клијент треба да наведе у ком *JOB*-у жели да покрене АПИ функцију. Користи се само изузетно и подразумевано поље остаје празно.
pozicija (char)	Ознака позиције. Употребљава се ако клијент треба да наведе на којој позицији жели да покрене АПИ функцију. Користи се само изузетно и подразумевано поље остаје празно.
checksum (char)	Контролни код поруке. Намена му је да се преко њега може проверити интегритет послатог упита. Ово је обавезан податак.

TAPIPARAMS

У оквиру упита поље *data* садржи параметре послате позваној АПИ функцији. Како ови параметри могу бити разнородни и зависе од сваке функције понаособ, садржај овог поља је у ствари угњеждени *JSON* чија структура је променљива зависно од позване функције.

Структуру параметара чине системска поља и поља која представљају параметре функције.

Системска поља имају унапред дефинисане намене а могу се али и не морају појавити у структури, зависно од потребе.

paging_sort_exp (char)	Клијент може навести израз за сортирање резултата. Пример: "GRUPA,A|MESTO,A|NAME,A|". Формат израза је POLJE,SMER|POLJE,SMER|POLJE,SMER... где су POLJE - назив поља, а SMER, је смер сортирања и то A – растући редослед, D – опадајући редослед, AC

	– растући редослед са занемаривањем величине слова и DC – опадајући редослед са занемаривањем величине слова
paging_selected_page (num)	Клијент може да користи страничење упита. Уколико се очекује да резултат упита садржи табелу са великим бројем слогова које клијент не може одједном да прикаже или је то непрактично, он може да од АПИ сервера тражи да изврши функцију, али да врати само одређену страну табеларног резултата. Употребљава се заједно са пољем *paging_page_len*, које назначава колико слогова табеле чини једну страну.
paging_page_len (num)	Употребљава се заједно са пољем *paging_selected_page* за страничење резултата.
databaseattachment (char)	Ако се као параметри позваној функцији шаљу табеларни подаци они се налазе у овом пољу. Формат података је наведен у пољу *database_format*. Уколико је формат *PLAIN_OBJ* у ово поље се уписује *JSON* облик објекта који садржи табеле, а ако је *SQLITE, ZIPSQLITE* или *ZIPDBF* онда се у ово поље уписује Base64 енкодована датотека *SQLite* базе.
database_format (char)	Назнака формата у коме се налазе табеле у пољу *databaseattachment*. За сада су имплементирана три формата: *PLAIN_OBJ* – табеле су објекти који се преносе као *JSON* стринг *PLAIN_OBJ_BASE64* – табеле су објекти, а шаљу се кодирани (последица је краћи JSON) NATIVE_*OBJ* – табеле су објекти *SQLITE* – табеле су смештене у *SQLite* базу ZIPSQLITE – табеле су смештене у *SQLite* базу a onda kompresovane u ZIP arhivu ZIPDBF – табеле су смештене у *ZIP* архиву као фокспро табеле Ако није наведен формат (поље је празно) подразумева се *PLAIN_OBJ*. Клијентске апликације су обавезне да подрже формате *PLAIN_OBJ* и *SQLITE*.
result_database_format (char)	Опис формата који треба применити за враћање табеларних података као резултата извршења функције. Погледати опис поља *database_format*.

Поља која представљају параметре функције су у потпуности прилагођена АПИ функцији и њихове дефиниције су дате за сваку АПИ функцију понаособ.

СТРУКТУРА ПОРУКА АПИ ОДГОВОРА

На упит АПИ сервер одговара тако што врати „страну“ која садржи *JSON* структуру.

JSON одговор се такође састоји из два слоја: оквир (*TAPIResponseEnvelope*) и резултат функције (*TAPIResult*).

Подаци се смештају у свосјтва наведених класа и могу бити прости типови, низови или објекти. Уколико неко својство нема вредност или има подразумевану вредност, то својство не мора постојати у JSON формату.

TAPIRESPONSEENVELOPE

Оквир (*TAPIResponseEnvelope*) је универзалан део *JSON* одговора који је непроменљив без обзира на врсту одговора. Његова улога је да обезбеди комуникацију између АПИ сервера и клијента. Он у себи садржи угњежден слој који садржи резултате извршавања позване АПИ функцији.

Структура *TAPIResponseEnvelope*:

req_id (char)	Идентификатор упита. Сервер враћа податак који је клијент послао у упиту. Служи да клијент може да по потреби повеже одговоре са упитима и за рачунање контролног кода за проверу целевитости послатог упита.
session_id (char)	Идентификатор сесије.
instance_id (char)	Идентификатор инстанце у оквиру сесије.
encoding (char)	Начин на који су кодирани подаци у пољу *data*. Клијент на основу овог податка зна како да декодира податке. Подразумевано се подаци не кодирају и ово поље остаје празно. Енкодирање података није још увек имплементирано.
app_option (char)	Ознака АПИ функције која је извршена.
data (char)	Резултат извршења АПИ функције. Ово је стринг (карактер) поље садржи угњеждену *JSON* структуру *TAPIResult*. Та структура је променљива зависно од функције која је извршена.
api_version (char)	Ознака верзије АПИ-ја на серверу.
poslovna_godina (char)	Ознака пословне године.
job (char)	Ознака *JOB*-а. Употребљава се ако клијент треба да наведе у ком *JOB*-у жели да покрене АПИ функцију. Користи се само изузетно и подразумевано поље остаје празно.
pozicija (char)	Ознака позиције.
sif_firme (char)	Ознака комитента коме припада корисник.
mesto (char)	Ознака места комитента (место испоруке, магацин...).
referent (char)	Ознака референта (референт продаје, магационер, представник комитента и слично).

error_state (char)	Ознака грешке у комуникацији или системске грешке. - OK значи да нема грешке. - API_FUNCTION_ERROR значи да је извршена функција вратила грешку. Детаљи о тој грешци се налазе у струкури резултата функције. - CARDINAL – системска грешка - UNDEFINED – системска грешка - Било која друга вредност значи да је дошло до неке грешке приликом извршавања АПИ функције.
error_message (char)	Описна порука о грешци која се може приказати кориснику клијентске апликације.
result_error_state (char)	Ознака грешке у извршењу АПИ функције. Вредност OK значи да нема грешке. Било која друга вредност значи грешку. Уколико су ознаке грешака функције документоване, клијент на основу њих може да прилагоди своје понашање.
result_error_message (char)	Описна порука о грешци АПИ функције која се може приказати кориснику клијентске апликације.
result_error_data (obj)	Структура која садржи додатне информације о грешци. Ако се користи документована је АПИ функцијом, иначе је null.
debug_log (char)	Ако је приликом позива АПИ функције укључен дибаг режим, ово поље ће садржавати дибаг лог поруке.
checksum (char)	Контролни код поруке. Намена му је да се преко њега може проверити интегритет послатог упита.

TAPIRESULT

У оквиру одговора поље *data* садржи резултат извршења позване АПИ функција. Функција може да врати податке разних типова и сложене структуре па је тако садржај овог поља је у ствари угнеждени JSON чија је структура променљива зависно од извршене функције.

Структуру резултата чине системска поља и поља која представљају резултате функције.

Системска поља имају унапред дефинисане намене а могу се али и не морају појавити у структури, зависно од потребе.

paging_selected_page (num)	Ако је клијент задао страничење упита, враћа се редни број стране.
paging_page_len (num)	Дужина стране у слоговима.
paging_rec_count (num)	Укупан број слогова у табели.
paging_sort_exp (char)	Израз који је употребљен за сортирање табеле.
databaseattachment (char)	Ако као резултат функције шаљу табеларни подаци они се налазе у овом пољу. Формат података је наведен у пољу *database_format*. Уколико је формат *PLAIN_OBJ* у ово поље се уписује JSON облик објекта који садржи табеле, а ако је *SQLITE, ZIPSQLITE* или *ZIPDBF* онда се у ово поље уписује Base64 енкодована датотека *SQLite* базе
database_format (char)	Назнака формата у коме се налазе табеле у пољу *databaseattachment*. За сада су имплементирана три формата: *PLAIN_OBJ* – табеле су објекти који се преносе као JSON стринг *PLAIN_OBJ_BASE64* – табеле су објекти, а шаљу се кодирани

	(последица је краћи JSON) NATIVE_OBJ – табеле су објекти SQLITE – табеле су смештене у SQLite базу ZIPSQLITE – табеле су смештене у SQLite базу a onda kompresovane u ZIP arhivu ZIPDBF – табеле су смештене у ZIP архиву као фокспро табеле Ако није наведен формат (поље је празно) подразумева се PLAIN_OBJ. Клијентске апликације су обавезне да подрже формате PLAIN_OBJ и *SQLITE*.
attachments (char)	У резултат се могу уврстити прилози у тектуалном или бинарном формату (датотеке на пример). Ово поље садржи струкуру са тим прилозима.
error_state (char)	Ознака грешке у извршењу функције. - OK значи да нема грешке. - COMPLEX_ERROR_STATE – апи функција је вратила листу грешака. Листа се налази у *error_data*. Било која други вредност значи неку грешку. Уколико су ознаке грешака функције документоване, клијент на основу њих може да прилагоди своје понашање.
error_data (obj)	Структура која садржи додатне информације о грешци. Ако је функција вратиал листу грешака ово својство ће садржавати ту листу. Ако ако АПИ функција користи ово својство да врати неке додатне податке о грешци то ће бити документовано у опису функције, иначе је null.
error_message (char)	Описна порука о грешци која се може приказати кориснику клијентске апликације.
exec_date_time (datetime)	Датум и време када је АПИ функција извршена на серверу.

ОБРАДА ГРЕШАКА

Након извршења, свака АПИ функција враћа статус о грешци. Подразумевани статус је да је функција извршена без грешака и да је вратила очекивани резултат.

Уколико дође до било које грешке АПИ функција ће вратити статус грешке чиме ће назначити да је дошло до грешке приликом извршења. Ако се ради о опису грешке коју враћа сама АПИ функција, те грешке ће бити документоване у опису функције.

Клијентска апликација треба да реагује на ове статусе. Подразумевани принцип је, да ако је дошло до неке грешке, клијентска апликација треба да сматра да позвана АПИ функција није урадила обраду која је очекивана нити да је вратила резултате који су очекивани и не сме да настави рад као да грешке није било. Она треба да обавести корисника о насталој грешци (или да је забележи у лог) и не треба да понавља исти позив функцији.

Уколико клијентска апликација познаје статусе грешака, она може на њих да реагује и другачије, односно на начин који следи из статуса грешке.

ТИПОВИ СТАТУСА ГРЕШАКА

Статус типа ОБАВЕШТЕЊЕ

Овај статус клијенту назначава да је настао неки проблем приликом извршења фунције али да он није изазвао прекид рада функције. Обично се тако може пријавити да је неки улазни податак недостајао или је био нетачан али да је АПИ функција сама то на неки начин регулисала, тако да није дошло до грешке.

Клијентска апликација на статус овог типа не мора да реагује, или га може приказати свом кориснику као обавештење.

Статус типа КРИТИЧАН

Овај статус значи да је приликом извршења АПИ функције дошло до критичне грешке и њеног прекида. Функција није извршена до краја, није извршила очекивану обраду и није вратила очекивани резултат.

Клијентска апликација на овакав статус реагује тако што се понаша као да је дошло до критичне грешке. Она не сме наставити рад као да је функција исправно извршена. Уколико уме, треба да на основу конкретног статуса грешке реагује на одговарајући начин. Клијентска апликација не сме да понови позив АПИ серверу са истим подацима ,осим уколико статус грешке не назначава да то треба учинити.

ШТА СВЕ ЧИНИ ОПИС ГРЕШКЕ

Када настане грешка клијентској апликацији ће бити прослеђен статус грешке. Опис грешке се уписује у *TAPIResultClass* а састоји од три податка уписана у својства *error_status*, *error_messageu error_data*:

error_status (char)

Ово је идентификатор статуса грешке. Вредност OK значи да није било грешке приликом извршавања функције. Било која друга вредност значи да је дошло до грешке.

За предвиђене грешке уводе се идентификатори статуса грешака и они су документовани у опису АПИ функције.

На основу идентификатора статуса грешке клијентска апликација може да реагује на одговарајући начин, уколико грешку познаје. Ако клијентска апликација не препознаје грешку треба да на њу реагује као да се ради о критичној грешци.

Идентификатор грешке треба проверавати као *Case Insensitive* низ знакова, односно не треба правити разлику између малих и великих слова.

error_message (char)

Текстуални опис статуса грешке. Садржи слободан текст који детаљније описује статус грешке, а који се може приказати кориснику клијентске апликације. Овај опис по правилу садржи довољно детаља да се одреди где је и због чега настала грешка.

error_data (obj)

Уколико је поред идентификатора, потребно клијенту послати и додатне информације о статусу грешке да би он могао да исправно реагује на грешку, користи се ово својство.

Ово својство може садржати скаларну вредност (на пример број документа приликом чије обраде је дошло до грешке) али и неку сложену структуру.

Ако декларација АПИ функције не садржи опис овог својства онда се оно не користи и биће празно.

ЛИСТА ОПИСА СТАТУСА ГРЕШАКА

Приликом извршавања неких функција може затребати да се клијенту врати не један, него више описа статуса грешака.

Чести случајеви:

- Пре извршења, функција проверава улазне параметре и код више параметара наилази на неисправности. Уместо да прекине извршавање и врати статус грешке на првом провереном параметру, она може да врати статусе грешака за све параметре појединачно, али одједном, као листу.

- Ако функција врши серијску обраду докумената могућа је ситуација да код обраде неких од докумената може доћи до грешке, али да функција треба да заврши обраду оних докумената где се грешка не јавља. У том случају функција може да врати листу статуса грешака за сваки обрађени документ појединачно.

ИнфоАПИ омогућава и овакав начин враћања статуса грешака. Уместо да се у *TAPIResultClass* статус грешке описује преко својстава error_status, error_message и error_data, опис статуса се може вратити клијенту као листа.

Листа статуса је листа објеката који садрже својства error_status, error_message и error_data. Са вредностима аналогним својствима у класи *TAPIResultClass*.

У листу статуса грешака се могу уписивати сви статуси грешака, укључујући и статус *OK*. Уколико сви статуси грешака у листи имају идентификатор статуса OK, онда ће и својство *TAPIResultClass.error_status* такође имати вредност *OK*.

Уколико макар један идентификатор статуса грешке у листи није OK, *TAPIResultClass.error_status* ће имати вредност *COMPLEX_ERROR_STATE*, који упућује да се статуси грешака налазе у листи. Листа статуса са грешкама се налази у *TAPIResultClass.error_list*.

Ако листа статуса грешака садржи само један статус, онда ће и основни статус грешке садржавати исте податке као тај један статус у листи.

СИСТЕМСКИ СТАТУСИ ГРЕШАКА

Системски статуси грешака су статуси који настају на системском нивоу и нису специфични за појединачне АПИ функције.

Статуси грешака специфични за АПИ функције су посебно документовани у опису сваке функције појединачно.

Системски статуси грешака на TAPIResponseEnvelope

ИДЕНТИФИКАТОР	ТИП ГРЕШКЕ	ОПИС СТАТУСА
OK	ОБАВЕШТЕЊЕ	Нема грешака
API_FUNCTION_ERROR	КРИТИЧНО	Извршена функција је вратила грешку. Детаљи о тој грешци се налазе у структури резултата функције и треба тамо проверити о којој грешци се ради.
API_PAGING_NOT_EXECUTED	КРИТИЧНО	Клијентска апликација је тражила податке са страничењем, а позвана АПИ функција не омогућава страничење
INVALID_YEAR	КРИТИЧНО	Не постоји пословна година
CARDINAL	КРИТИЧНО	Системска грешка настала у току извршавања АПИ функције
UNDEFINED	КРИТИЧНО	Дошло је до системске грешке а да АПИ функциа није ни покренута
APPLICATION_ERROR	КРИТИЧНО	Грешка настала у иницијализацији АПИ сервиса
NO_USER_RIGHTS	КРИТИЧНО	Пријављени корисник нема довољна корисничка права да покрене тражену АПИ функцију
UNKNOWN_FUNCTION	КРИТИЧНО	Не постоји позвана АПИ функција
INVALID_SERVER_LICENCE	КРИТИЧНО	Сервер није лиценциран да користи тражену АПИ функцију
INVALID_CLIENT_LICENCE	КРИТИЧНО	Клијент није лиценциран на серверу да користи тражену АПИ функцију
INVALID_FUNCTION_LICENCE	КРИТИЧНО	Серверу није дозвоњено коришћење тражене АПИ функције

ИДЕНТИФИКАТОР	ТИП ГРЕШКЕ	ОПИС СТАТУСА
INVALID_MODULE_LICENCE	КРИТИЧНО	Серверу није дозвољено коришћење модула
CHECKSUM_REQUIRED	КРИТИЧНО	Порука не садржи контролни код
MODULE_LICENCE_EXPIRED	КРИТИЧНО	Истекла је лиценца за модул
CLIENT_LICENCE_EXPIRED	КРИТИЧНО	Истекла је лиценца клијента
CLIENT_NOT_LICENCED	КРИТИЧНО	Сервер нема лиценцу за овог клијента
NO_FUNCTION_CALL	КРИТИЧНО	Упит не садржи ознаку АПИ функције. Обично то значи да упит није послат POST методом.
ERROR_NO_PROCESS_DESC	КРИТИЧНО	АПИ фунција је покушала да региструје процес без описа.
ERROR_DUPLICATE_PROCESS	КРИТИЧНО	Клијент је покушао да истовремно позове исту АПИ функцију са истим улазним параметрима а она то не дозвољава.

Системски статуси грешака на TAPIResult

ИДЕНТИФИКАТОР	ТИП ГРЕШКЕ	ОПИС СТАТУСА
OK	ОБАВЕШТЕЊЕ	Нема грешака
COMPLEX_ERROR	КРИТИЧНО	АПИ функција је вратила листу грешака. Треба погледати у листу да би се утврдило како реаговати на сваку грешку појединачно.
ERR	КРИТИЧНО	Дошло је до грешке у извршавању АПИ функције. ЗАСТАРЕЛО. Не користи се.
MISSING_TABLE	КРИТИЧНО	Недостаје табела кој би требало да буде послата као параметар АПИ функције. У опису грешке може бити наведено која је то табела.
MISSING_PARAMETER	КРИТИЧНО	Недостаје параметар АПИ функције који ј еобавезан. У опису грешке може бити наведено који је то параметар.

КОНТРОЛНИ КОД

Намена контролног кода је да се провери целовитост послате поруке, обезбеди интегритет података и изврши аутентификација клијентске апликације.

Контролни код се рачуна тако што се саберу као стрингови вредности својстава *TAPIQueryEnvelope.data* и *TAPIQueryEnvelope.req_id* и и приватни кључ (добијате га уз лиценцу) и над тако добијеним стрингом израчуна се МД5 хеш.

Израчунати хеш се као хексадекадни број уписује у *TAPIQueryEnvelope.checksum.*

```
TAPIQueryEnvelope.checksum = MD5 (TAPIQueryEnvelope.data + TAPIQueryEnvelope.req_id + priv_key)
```

Ово треба радити као последњу операцију пре претварања објекта у *JSON* формат.

Провера контролног кода се ради тако то се уради исти поступак рачунања контролног кода, само се добијена вредност пореди са оним који је већ уписан у својство *TAPIResponseEnvelope.checksum* пристигле поруке.

На исти начин се рачуна контролни код и упита и резултата упита, односно као што контролни број упита проверава АПИ сервер тако клијентска апликација треба да, из безбедносних разлога, проверава контролни код одговора који добије до сервера.

Приватни кључ је низ знакова који се добија уз лиценцу а везан је за идетификатор апликације. Приватни кључ се не размењује. Њега имају и ИнфоАПИ сервер и клијентска апликација и само га употребљавају за рачунање контролног кода. ИнфоАПИ сервер према идентификатору апликације, који обавезно треба слати у свакој поруци, зна који приватни кључ треба да употреби.

ИнфоАПИ сервер неће прихватити упит који не садржи исправан контролни код који потврђује да је клијентска апликација лиценцирана.

Док развијате апликацију, можете да употребите ДЕМО лиценцу коју ћемо вам доставити на захтев. Она садржи CLIENT_ID и приватни кључ са којима можете да обављате тестирање.

Пример PHP функције која рачуна контролни код:

```
function GetChecksum($pData, $pRequestId, $pPrivateKey)  {

      $lCheckSumData = json_encode ($pDdata, JSON_UNESCAPED_UNICODE)
                        . $pRequestId
                        . $pPrivateKey;

      $lResult = strtoupper (MD5 ($lCheckSumData));

     return ($lResult);

}
```

Пример C# функције која рачуна контролни код:

```
public string GetChecksum (string pData, string pRequestId, string pPrivateKey) {

  MD5 md5 = MD5.Create();

  byte[] inputBytes = Encoding.UTF8.GetBytes(pData + pRequestId + pPrivatekey);
  byte[] hash = md5.ComputeHash(inputBytes);

  string lResult = BitConverter.ToString(hash).Replace("-", "").ToUpper();

  return lResult;

}
```

Пример Python функције која рачуна контролни код:

```
import hashlib

def create_checksum(data: str, req_id: str, license_key: str) -> str:
    """
    data - from main json object string of data property
    req_id - from main json object top property req_id
    license_key - uniq license key
    """

    tmp_string = data + req_id + license_key
    checksum = hashlib.md5(tmp_string.encode())

    return checksum.hexdigest()
```

ЛИЦЕНЦИРАЊЕ

ЛИЦЕНЦА КЛИЈЕНТСКЕ АПЛИКАЦИЈЕ

Да би клијентски програм могао да позива АПИ функције потребно је да има одговарајућа овлашћења. Ова овлашћења се дефинишу лиценцом апликације која одређује коју функционалност АПИ-ја наведена апликација може да користи.

Лиценцу издаје Инфосус произвођачу клијентског програма. Ако произвођач има више програма који треба да комуницирају са АПИ-јем лиценца се издаје за сваки програм посебно.

Лиценца одређује идентификатор и приватни кључ програма. Приликом припреме упита, програм треба да у својство *TAPIQuery.client_id* упише идентификатор из лиценце и да у *TAPIQuery.checksum* упише контролни код упита који се рачуна у функцији од приватног кључа који је добијен лиценцом (погледати одељак који објашњава рачунање контролног кода).

Да би АПИ сервер прихватио упит лиценциране клијентске апликације за извршење функције морају бити испуњени следећи услови:

1. Клијентска апликација мора имати лиценцу Инфосиса да може да комуницира са АПИ-јем и да може да покрене тражену АПИ функцију.

2. АПИ сервер на који се клијентска апликација повезује мора имати лиценцу која му дозвољава да прихвата упите од дате лиценциране клијентске апликације за позиве наведеној АПИ функцији.

4. Упит мора садржавати ИД клијента и исправан контролни број израчунат уз помоћ приватног кључа из лиценце клијентске апликације.

ЛИЦЕНЦА АПИ СЕРВЕРА

АПИ сервер мора имати одговарајућу лиценцу да би могао да прихвата упите за извршавање АПИ апликација. Ову лиценцу набавља корисник Инфосис програма у складу са својим потребама.

Лиценца сервера одређује:

- које АПИ функције се могу извршавати на серверу;

- које клијентске апликације могу да се повезују са сервером и које функције да извршавају;

- укупан број корисника и број истовремених корисника на серверу;

- учесталост и број упита ка појединачним АПИ функцијама, и

- друге параметре сервера и комуникације са клијентима.

ПОСЕБНЕ СТРУКТУРЕ ПОДАТАКА

У структури упита и одговора појављују се неке специфичне подструктуре. Овде ћемо их описати.

DATABASEATTACHMENT

Параметри упита као и резултати извршења АПИ функције могу да садрже табеларне податке. Они се смештају у поље *databaseattachment*.

Подаци у овом пољу могу бити у неколико формата:

PLAIN_OBJ формат

Подаци су организовани као низ објеката који представљају табеле. Свака табела садржи низ описа поља табеле и низ објеката слогова који садрже вредности поља.

DatabaseAttachment објекат садржи својство *items* које је низ објеката табела.

Објекат табеле садржи својства:

fields - опис структуре поља табеле

rows - садржај табеле

table_name – назив табеле

Објекат *fields* садржи својство *items* које представља низ објеката описа поља табеле. Сваки објекат описа поља табеле садржи својство *name* за назив поља и својство *type* за тип поља.

Подржани типови поља су:

C – string, максимална дужина 250 знакова

I - integer

N - numeric

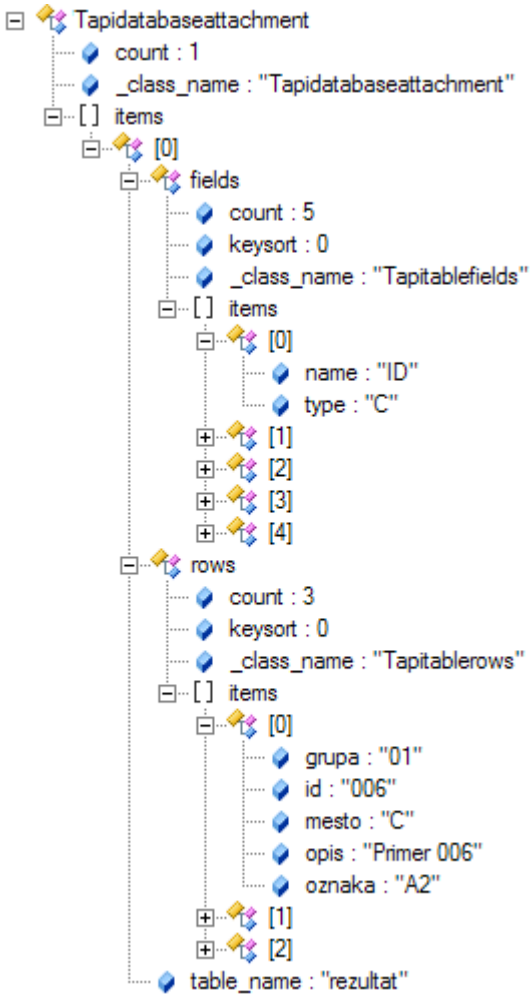
D - date

L - logical

M – memo, стринг неограничене дужине

F – float

Објекат *rows* садржи својство *items* које представља низ објеката слогова табеле. Објекат слога табеле садржи својства која одговарају називима поља табеле са њиховим вредностима.



Пример структуре databaseattachment објекта

Све то се претвара у *JSON* стринг који се уписује у поље *databaseattachment*.

Приликом читања података, треба узети садржај својства *databaseattachment* као *JSON* стринг, декодовати га у објекат и затим читати податке из његове структуре.

Недостатак овог формата је што има доста редундансе у структури записа, па запис заузима много више места него што је потребно за саме податке. Такође, за форматирање се троши доста ресурса. Погодан је за пренос мањег обима података.

PLAIN_OBJ_BASE64 формат

Ово је варијанта *PLAIN_OBJ* формата. Разлика је што се подаци кодирају у *Base64*. Тиме се знатно скраћује дужина података која се преноси између сервера и клијента.

Читање овог формата се врши тако што се садржај декодује *Base64* из па се добије JSON стринг и тада се тај стринг може декодовати у објекте из којих се могу читати подаци.

NATIVE_OBJ формат

Овај формат је веома сличан формату *PLAIN_OBJ*. Подаци су организовани као низ објеката који представљају табеле. Свака табела садржи низ описа поља табеле и низ објеката слогова који садрже вредности поља u strukturi istoj kao za *PLAIN_OBJ*. Разлика је само у томе што је *NATIVE_OBJ* у структури поруке уписује у поље *databaseattachment* као објекат, а *PLAIN_OBJ* се преведу у *JSON* стринг и као такав уписује у у поље *databaseattachment.*

Овај формат троши и више ресурса него *PLAIN_OBJ* тако да га треба употребљавати само изузетно.

SQLITE формат

Креира се SQLite база података и у њој табеле које одговарају по структури и садржају табелама које треба уврстити у АПИ поруку. Тако направљена датотека се енкодује као Base64 стринг и тако уписује у поље *databaseattachment*.

Читање података из овог формата се врши тако што се примљени стринг декодује као као Base64 стринг а добијени садржај сними као бинарна датотека. Ову датотеку можете отворити као и сваку другу SQLite базу података и прочитати табеле које су у њу уписане.

ZIPSQLITE формат

Креира се ZIP arhiva и њу архивира *SQLITE* база под именом data.db у којој су табеле које одговарају по структури и садржају табелама које треба уврстити у АПИ поруку. Тако направљена датотека се енкодује као *Base64* стринг и тако уписује у поље *databaseattachment*.

Читање података из овог формата се врши тако што се примљени стринг декодује као као Base64 стринг а добијени садржај сними као бинарна датотека која је ZIP архива. Архива садржи датотеку data.db коју треба распаковати из ње. Датотеку data.db можете отворити као и сваку другу *SQLite* базу података и прочитати табеле које су у њу уписане.

С обзиром да се садржај обично може веома добро компресовати, коришћењем овог формата се значајно може смањити величина датотеке, односно количина података која се преноси и тиме скратити и време преноса.

ZIPDBF формат

Креира се *ZIP* arhiva и њу архивирају фокспро (*DBF*) табеле које одговарају по структури и садржају табелама које треба уврстити у АПИ поруку. Тако направљена датотека се енкодује као *Base64* стринг и тако уписује у поље *databaseattachment*.

Читање података из овог формата се врши тако што се примљени стринг декодује као као *Base64* стринг а добијени садржај сними као бинарна датотека. Ову датотеку можете распаковати као и сваку другу *ZIP* датотеку и прочитати подаке из табела које се у њој налазе.

ATTACHMENT

Параметри упита као и резултати извршења АПИ функције могу да садрже прилоге у слободном формату. Они се смештају у поље *attachment*.

Подаци су организовани као низ објеката који представљају прилоге. Сваки објекат има поља:

id (char)	идентификатор додатка
type (char)	тип додатка: string – слободан стринг садржај jpg – JPG слика png – PNG слика gif – GIF слика txt – TXT датотека ...
content (char)	Садржај додатка енкодиран као *Base64*
file_name (char)	Име датотеке ако је додатак датотека.
last_update (datetime)	Датум и време последње измене додатка.

ИЗРАЗИ

Неке функције могу као параметре примати изразе. Израз се састоји од идентификатора израза и параметара израза. Израза је представљен као структура која садржи својства:

exp	Ознака израза
params.items	низ вредности параметара у изразу

Пример: израз 2 + 3 се представља као

```
m_filter1.exp = 'ADD';
m_filter1.params.items[0] = 2;
m_filter1.params.items[1] = 3;
```

Параметар у узразу може бити други израз. Тако се могу градити сложени изрази.

Пример: израз 2 + 3 / 2 се представља као

```
m_filter1.exp = 'ADD';
m_filter1.params.items[0] = 2;
m_filter1.params.items[1] = 3;
```

```
m_filter2.exp = 'DIV';
m_filter2.params.items[0] = m_filter1;
m_filter2.params.items[1] = 2;
```

Израз не мора да буде само математички, већ може бити и логички:

Пример: израз 'cena' > 0 се представља као

```
m_filter1.exp = 'GT';
m_filter1.params.items[0] = 'cena';
m_filter1.params.items[1] = 0;
```

Ако је као параметар израза наведен стринг, онда се он третира као име промењиве или назив поља табеле и уместо њега ће бити употребљена одговарајућа вредност. Уколико је потребно проследити стринг вредност као параметар, тада је треба ставити у наводнике.

Пример:

```
m_filter1.params.items[0] = 'cena'       додељује вредност поља или промењиве cena


m_filter1.params.items[0] = '"cena“'     додељује стринг вредност „цена“
```

У ИнфоАПИ-ју изрази се најчешће примењују као филтери – логички изрази, али се могу употребљавати и у свакој другој ситуацији где су изрази потребни.

Подржани изрази

Ознака	Опис	Број параметара
AND	Логичко И	2
OR	Логичко ИЛИ	2
GT	Веће од	2
LT	Мање од	2
GE	Веће или једнако	2
LE	Мање или једнако	2
EQ	Једнако	2
EQX	Апсолутно једнако	2
NE	Није једнако	2
IN	Садржи се у	2
INLIST	Садржи се у листи	2 до 25
LIKE	Личи на	2
UPPER	Претвори у велика слова	1

LOWER	Претвори у мала слова	1
ALLTRIM	Уклони знак размака сапочетка и краја	1
DATETIME	Претвори параметре у TDateTime	6
ADD	Сабери	2
SUB	Одузми	2
MUL	Помножи	2
DIV	Подели	2

СТРАНИЧЕЊЕ РЕЗУЛТАТА

Резултат функције може бити једна или више табела података. Ако је обим тих података велики је у корисничком интерфејсу непрактично да се одједном преузимају и приказују сви подаци, може се применити страничење.

У основи, страничење је поступак којим се на први позив АПИ функције креира кеш табела са подацима, али се не враћа цела већ се враћа само један део табеле, док се у наредним позивима истој функцији преузимају остали слогови из кеш табеле без поновног генерисања података.

Клијент страничење може да употреби само са АПИ функцијама које то омогућавају. То у АПИ функцијама треба да буде документовано. Уколико нека АПИ функција не могућава страничење, а потребно вам је, контактирајте Инфосис.

Укључивање страничења резултата

Клијент укључује страничење тако што приликом позива функције у параметрима (*TAPIParams*) пошаље:

paging_page_len (num)	Дужина стране у слоговима. Употребљава се заједно са пољем *paging_selected_page* за страничење резултата.
paging_selected_page (num)	Страница коју треба вратити. Употребљава се заједно са пољем *paging_page_len*, које означава дужину странице.
paging_sort_exp (char)	Клијент може навести израз за сортирање резултата. Пример: "GRUPA,A|MESTO,A|NAME,A|". Формат израза је POLJE,SMER|POLJE,SMER|POLJE,SMER... где су POLJE - назив поља, а SMER, је смер сортирања и то A – растући редослед, D – опадајући редослед, AC – растући редослед са занемаривањем величине слова и DC – опадајући редослед са занемаривањем величине слова.

Уколико клијент не наведе *paging_page_len*, АПИ ће аутоматски узети да је дужина странице једнака укуном броју слогова у страниченој табели.

Уколико клијент не наведе *paging_selected_page*, АПИ ће аутоматски узети да треба вратити прву страницу станичене табеле.

Уколико клијент наведе *paging_sort_exp*, аутоматски се ресетује кеш за страничење и креира новакеш табела, а *paging_selected_page* се ресетује на прву страницу без обзира коју је страницу клијент тражио.

Употреба страничених података

Станичење се укључује првим позивом АПИ функције која подржава страничење и навођењем параметара страничења: *paging_page_len*, *paging_selected_page* и опционо *paging_sort_exp.*

Тада АПИ функција креира резултујућу табелу и над њом примењује страничење, тако што не враћа све податке из табеле, већ само део који се израчунава на основу задате дужине странице и броја странице коју треба вратити. Страничена табела остаје сачувана у кешу и у наредним позивима исте АПИ фунције у оквиру исте сесије (или инстанце) функција неће поново генерисати резултујућу табелу већ ће из кеширане табеле издвајати и враћати слогове према задатим параметрима страничења.

Уколико клијент наведе *paging_sort_exp*, аутоматски се ресетује кеш за страничење и креира нова кеш табела а *paging_selected_page* се ресетује на прву страницу без обзира коју је страницу клијент тражио.

Уз податке, АПИ функције враћа и опис страничења које је примењено. То су исти подаци које је клијент послао - *paging_page_len*, *paging_selected_page* и *paging_sort_exp* – али вредности могу бити другачије од оних које је кијент послао у упиту (рецимо, клијент је тражио 15-ту страницу а она не постоји, већ је враћена последња, на пример 12-та страница).

Како клијент не добија цео резултат већ само једну страницу, он не може знати колико укупно има података у резултату, што му је потребно да би прилагодио кориснички интерфејс (приказао број страница) али и да би знао колико података још треба да преузме. Ту информацију му даје АПИ функција својством:

paging_rec_count (num)	Укупан број слогова у табели.

Клијент увек треба да прочита ове вредности и да даљи рад са добијеним подацима усклади према њима, а не према параметрима које је послао у упиту.

СЕСИЈЕ И ИНСТАНЦЕ

ШТА ЈЕ СЕСИЈА

Када клијент први пут позове неку АПИ функцију, за њега се отвара сесија. Она обухвата тај и све следеће позиве АПИ функција све док се корисник не одјави или док сесија не истекне због неактивности корисника.

Сесија омогућава да се неки подаци на серверу чувају између два позива АПИ-ја. Сваки позив АПИ функцији може да мења вредности променљивих сесија као и да чита претходно постављене вредности сачуваних било када у току сесије. Када се сесија затвори, бришу се и сви подаци који су у њој били сачувани.

Подаци који се могу чувати у сесији су: променљиве сесије, табеле и датотеке.

У сваком одговору на упит АПИ у *TAPIQuery.session_id* враћа идентификатор текуће сесије. Када добије овај идентификатор, клијент, све док жели да ради у истој сесији, треба да у сваком наредном позиву прослеђује добијени идентификатор. Ако клијент не проследи идентификатор сесије у упиту, биће аутоматски отворена нова сесија.

ИНСТАНЦА У СЕСИЈИ

У пракси постоји потреба да се користе нека врста сесија у сесијама. То је најлакше објаснити на примеру: рецимо да позивате АПИ функцију али она даје страничене податке тако да омогућава да клијент може да позове исту функцију више пута али да сваки пут да узима различиту страну података. То значи да ће табела која представља резултат извештаја бити смештена у сесију и сваки следећи позив исте АПИ функције неће наново креирати ту табелу већ ће користити постојећу.

Међутим ако клијенту треба да истовремено покрене исту АПИ функцију али са различитим параметрима - то постаје проблем. Ако се креира нови резултат извештаја, пошто се ради о истој сесији, стари ће бити преписан и више неће бити доступан.

Због тога АПИ подржава инстанце у оквиру сесије. Када се покрене АПИ функција она добија своју инстанцу те се подаци који се чувају у сесији везују за ту инстанцу. Позив истој АПИ функцији у оквиру исте инстанце ће увек видети исте податке који су смештени у сесију.

Ако се иста АПИ функција покрене са другачијим параметрима, то ће представљати другу инстанцу па ће се креирати нови скуп података који иако су у истој сесији, неће мењати податке других инстанци истог извештаја.

Инстанца постоји све док постоји и сесија. Затварањем сесије затварају се и све инстанце које јој припадају. Инстанцу клијентска аплиакција може и да затвори сама и обично то и треба да чини када покреће нову инстанцу извештаја.

У сваком одговору на упит АПИ у *TAPIQuery.instance_id* враћа идентификатор текуће инстанце. Када добије овај идентификатор, клијент, све док жели да ради у истој инстанци, треба да у сваком наредном позиву прослеђује добијени идентификатор.

Ако клијент жели да опкрене нову инстанцу не треба да у позиву АПИ-ју пошаље ИД инстанце. Тиме се аутоматски креира нова инстанца, чији ИД ће бити враћен у одговору на упит.

Када користити сесију а када инстанцу?

Разлика између смештања података у сесију и смештања у инстанцу је у видљивости.

Оно што је смештено у сесију видљиво је у оквиру сесије односно, доступно је свим инстанцама. Оно што је смештено у инстанцу видљиво је само у оквиру те инстанце.

У начелу, клијент са АПИ-јем треба да комуницира у оквиру инстанце, односно, при сваком позиву АПИ функције треба да проследи добијени ИД сесије и ИД инстанце.

КАДА ТРЕБА КОРИСТИТИ СЕСИЈЕ

Инфо АПИ систем омогућава клијенту да се ослони на сесије али то не условљава. О томе да ли ће клијент да употреби сесије или не, зависи од намене клијента.

Ако клијенту није потребан рад у сесији он просто не треба да води рачуна о Ид сесије (и ид инстанце) већ треба да их игнорише. Сваки нови позив АПИ-ју ће тада креирати нову сесију.

Овакав режим рада има смисла када се ради о клијенту који повремено позива АПИ, нарочито ако је време између два позива дуже од времен истека сесије, па свакако сваки нови позив креира нову сесију.

Такође, ако клијент позива АПИ да преузме или пошаље неке податке и није му потребно страничење, кеширање или било која друга функционалност која је заснована на сесији, он н емора да води рачуна о сесијама.

Сесије треба користити ако:

- Клијент треба често да позива АПИ, да би се једном пријавио и затим сукцесивно позивао АПИ у оквиру те сесије тако да не мора да сваки пут шаље корисничко име и лозинку. На тај начин се смањује могућност неовлашћеног читања корисничког имена и лозинке јер се они шаљу само изузетно, када је то неопходно.

- Клијент позива АПИ функције и потребне су му функционалности које се ослањају на сесије, на пример страничење и кеширање података.

КАКО СЕ РАДИ БЕЗ СЕСИЈА

Ако клијенту сесије нису важне онда их не мора користити. Ако сваки пут, када шаље упит АПИ серверу, даје корисничко име и лозинку, тиме ће практично искључити механизам сесија.

ПРИЈАВА КОРИСНИКА

Да би клијентски програм могао да извршава АПИ функције потребно је да корисник има одговарајућа овлашћења. Корисник се иденификује корисничким именом и лозинком.

Поред очигледне примене за доделу овлашћења, корисничко име може и да утиче на начин рада АПИ функције тако што се окружење прилагођава конкретном пријављеном кориснику.

Корисник се АПИ-ју идентификује само једном у току сесије. Након пријаве, идентитет и статус корисника се чува у сесији и све док је корисник у истој сесији, АПИ зна ко је он и каква су му овлашћења.

Корисничко име и лозинка се шаљу својствима TAPIQuery.username и TAPIQuery.password. Препоручујемо да се, уколико се не користи VPN или https заштита кнкција ка серверу, лозинка енкриптује по AES256 протоколу.

ОДЈАВА КОРИСНИКА

Корисник се одјављује прекидом сесије. Сесија се може прекинути на три начина:

Одјавом корисника

Позивом АПИ функције LOGOUT налаже се АПИ-ју да прекида текућу сесију и тиме се одјављује и корисник.

Поновном пријавом корисника

Ако се корисник у текућој сесији поново пријави (пошаље корисничко име и лозинку) то аутоматски прекида текућу сесију тог корисника и отвара нову, у којој је корисник пријављен послатим корисничким именом и лозинком.

Истеком сесије

Сесија има ограничено трајање. Примењују се два ограничења која се подешавају на серверу:

- ограничење периода између два позива АПИ-ју одређује у ком временском периоду корисник мора да упути нови позив АПИ-ју да се сесија не би прекинула. То значи да ако протекне више времена након позива АПИ-ју, сесија ће бити прекинута и при наредном позиву АПИ ће тражити од корисника да се поново пријави. Подразумевано ограничење је 10 минута.

- ограничење апсолутног трајања сесије одређује колико најдуже сесија може да траје без обзира на активност корисника. То значи да се може ограничити укупно трајање сесије чак и ако је корисник веома активан и никада не долази до истека сесије према претходном ограничењу. Подразумевано ово ограничење не постоји, сесија може трајати бесконачно.

ПРИМЕРИ JSON СТРУКТУРА

Уз овај документ приложени су и примери упита и одговора АПИ-ја на упите у JSON формату.

Примере можете испробати ако што ћете JSON упите слати на демо АПИ сервер на адреси http://apitest.infosys.rs:31000/RestTest.html

АПИ ФУНКЦИЈА NONE

Функција не ради ништа и не враћа никакав резултат осим статуса грешке OK. Она демонстрира најједноставније (празне) облике JSON структура упита и одговора.

Декларација (опис): NONE.pdf

Упит: json-primer-apf-NONE-request.json

Одговор: json-primer-apf-NONE-response.json

АПИ ФУНКЦИЈА INFO

Функција INFO не прима никакве параметре а враћа низ података о окружењу АПИ сервера. Она демонстрира структуру одговора који садржи резултат у облику низа или објекта.

Декларација (опис): INFO.pdf

Упит: json-primer-apf-INFO-request.json

Одговор: json-primer-apf-INFO-response.json

АПИ ФУНКЦИЈА GETTABLE

Функција GETTABLE демонстрира JSON структуру одговора која садржи табелу у PLAIN_OBJ формату.

Декларација (опис): GETTABLE.pdf

Упит: json-primer-apf-GETTABLE-request.json

Одговор: json-primer-apf-GETTABLE-response.json

АПИ ФУНКЦИЈА GETTABLE SQLITE ФОРМАТ

Клијент може да приликом позива АПИ функције изабере у ком податку жели табеле. Овај пример демонстрира како се преузимају подаци у SQLITE формату. Иако је PLAIN_OBJ подразумевани формат, препоручује се употреба SQLITE формата, нарочито ако табеле могу садржавати већи број слогова.

У упиту је подешено својство data.result_database_format да садржи вредност „SQLITE“ чиме је назначен жељени формат табеларних података.

Декларација (опис): GETTABLE.pdf

Упит: json-primer-apf-GETTABLE-sqlite-request.json

Одговор: json-primer-apf-GETTABLE-sqlite-response.json

АПИ ФУНКЦИЈА COMPLEXERROR

Функција COMPLEXERROR демонстрира како изгледа структура одговора функција листа статуса грешака. Сама функција не прима параметре а увек враћа симулирану листу грешака.

Декларација (опис): COMPLEXERROR.pdf

Упит: json-primer-apf-COMPLEXERROR-request.json

Одговор: json-primer-apf-COMPLEXERROR-response.json

БРЗИ ПРИМЕР

Направићемо брзи пример поступка за креирање упита за позив АПИ функције NONE. Ова функција је најједноставнија јер не садржи никакве параметре па је добра за прву пробу.

КРЕИРАЈ УПИТ

Упит се састоји од два нивоа: први ниво представља оквир упита чија је структура универзална, а други ниво је сам упит који може имати различиту структуру зависно од тога која АПИ функција се покреће.

Креирајте заглавље тако што ћете попунити следећа својства:

req_id: "4JW101F08"	јединствени идентификатор упита. Ово ви одређујете, може бити насумичан низ знакова али треба да буде јединствен (да се не понавља) у неком разумном периоду. Ако на АПИ серверу треба да се пропрати шта се догодило са вашим упитом, овај идентификатор је једина веза.
app_option: "NONE"	назнака да желимо да покренемо функцију NONE
client_id: "DEMO"	сваки АПИ клијент мора да се представи идентификатором. За пробу користите идентификатор DEMO
username: "info"	корисничко име
password: "demo"	лозинка
data: ""	ово својство треба да садржи JSON стринг са специфичном структуром упита (параметри и други подаци који се шаљу функцији). Функција NONE не тражи никакве параметре тако да можемо ово да оставимо празно.
checksum: "xxxxx"	на крају се рачуна и уписује контролни код поруке. Он служи да се провери исправност података али и да се потврди идентитет пошиљаоца поруке. На демо серверу је подешено да се чексум не проверава ако га нема, а ако га има биће проверен. То је урађено да би се олакшао развој нових клијената. Дакле, док радите са сервером за тестирање поруке можете слати и без чексума али коначно чексум морате урадити јер је он обавезан на радном серверу.

Овако припремљену JSON поруку треба послати демо АПИ серверу да би он извршио функцију.

Демо сервер се налази на адреси http://apitest.infosys.rs:31000/resttest. Урит шаљете HTTP протоколом. JSON порука се у облику стринга шаље као POST параметар под именом json.

АНАЛИЗИРАЈ ОДГОВОР

Након што пошаљете поруку сервер ће вам одговорити на њу такође JSON струткуром коју треба да анализирате. Та струкура може да изгледа овако:

```
{
  "api_version": "0.10",
  "app_option": "NONE",
  "checksum": "7D001EBDDA805817FF287F4596AD2731",
  "data": "{\"database_format\": \"PLAIN_OBJ\",\"error_data\":
null,\"error_message\": \"\",\"error_state\": \"ok\",\"exec_date_time\": \"2016-01-
28T16:48:55+01:00\",\"name\": \"tapinoneresult2\",\"_class_name\": \"Tapiqueryclass\"}",
  "debug_log": "",
  "disk": "P:",
  "encoding": "PLAIN",
  "error_message": "",
  "error_state": "ok",
  "instance_id": "b7562d32a6caf0b8",
  "job": "fin",
  "mesto": "",
  "name": "Tapiresponseenvelope",
  "poslovna_godina": 2016,
  "pozicija": "01",
  "referent": "0002",
  "req_id": "4JW101F08",
  "result_error_message": "",
  "result_error_state": "ok",
  "session_id": "7975f61e448e5d0",
  "sif_firme": "",
  "_class_name": "Tapiqueryclass"
}
```


Већина ових својстава није у овом тренутку важна и можете их игнорисати. Сетите се правила нашег протокола: „Ако не знаш чему служи одређено својство, игнориши га - користи она својства која су ти потребна“.

Прво ћемо погледати битна својства.

error_state	Ово се прво проверава. Ако ово својство не садржи OK онда је дошло до неке грешке и ово својство садржи код грешке. Код грешке је дефинисан или овим протоколом ако је неки општи или у оису саме функције ако је функција пријавила грешку. Уз код статуса грешке додатне информације могу биди дате у својству error_message.
error_message	Опис грешке. Слободан текст који описно објашњава грешку.

Као што смо рекли, клијент обавезно мора да реагуеј на грешке. Ако не добије статус грешке OK, он мора да се понаша као да није успело извршавање АПИ функције. На основу кода статуса грешке клијент може да препозна о каквој се грешци ради и да прилагоди своју реакцију на њу. Ако грешку не препознаје, просто сматра да функција није извршена. Код статуса грешке и опис статуса грешке се могу уписати у неки лог или приказати кориснику.

data	Ако је error_state OK онда и својству data добијамо JSON са резултатима извршавања АПИ функције. У овом случају не очекујемо никакве резултате тако да то не користимо. Ако се резултати добијају, JSON ће бити у облику стринга који треба десеријализовати у неку структуру из које ћемо прочитати резултате.

Мање битна својста која треба познавати али нису обавезна:

checksum	исто као што клијент рачуна чексум поруке коју шаље и уписује гау поуку то ради и сервер. Пошто је клијент иницирао комуникацију он не мора да проверава чексум али податак је ту, ако је потребна додатна провера исправности и порекла података.

session_id	ИД сесије. Свака комункација са АПИ сервером се врши кроз сесије. Приликом пријема упита, ако већ не постоји сесија, сервер ће креирати нову. Овде је увек дат ИД сесије у којој се врши комуникација. Клијент ово може игнорисати ако не ради са сесијама али ако ради, сваки следећи упит који шаље треба да има исти овај ИД сесије да би се комуникација вратила у истој сесији.
instance_id	Инстанца је подсесија и на њу се односи све исто као и на сесију.

ШТА ДАЉЕ

Када сте успешно припремили и посали први упит функцији NONE и добили и анализирали добијени одговор велики део посла је завршен.

Следећи корак је да направите позиве АПИ функцији INFO. Она има додатни степен сложеност што враћа податке. Дакле позив АПИ функције је и даље једноставан јер функција нема параметара али враћа податке па се можете позабавити њиховом анализом.

Затим пробајте позив АПИ функцији MATH која има и улазне параметре и враћа резултат.

Када то урадите пробајте и остале демо функције које су овде наведене јер свака уводи неки нови механизам.

ЧЕСТЕ ГРЕШКЕ

JSON ФОРМАТ

- Не треба користити <TAB> за форматирање.

- Између назива својства и двотачке не сме стајати знак размака.

- Између двотачке и вредности својства не сме бити више од једног знака размака, а може да нема знак размака.

- Датум и време треба форматирати стриктно према ISO 8601 (YYYY-MM-DDTHH:MM:SS+HH:MM, где је време изражено у локалној временској зони а +HH:MM означава која је локална временска зона).

- Својство које садржи само датум, треба форматирати исто као и својство које садржи датум и време.

- Децимална тачка је увек тачка. Не може се користити зарез.

- JSON стринг увек треба третирати као УТФ-8.

- стринг вредности морају бити уоквирене двоструким наводницима. Једноструки наводници нису дозвољени.

- не треба очекивати да у JSON поруци постоје сва могућа својства која су предвиђена протоколом или декларацијом АПИ функције. АПИ сервер неће у JSON стављати она својства која су празна. Ако клијент очекује неко својство у структури, а оно не постоји, за то својство треба да узме празну или подразумевану вредност.

- JSON структуре могу садржавати и својства која нису предвиђена или документована протоколом односно декларацијом АПИ функције. Такве вредности треба игнорисати.

ОБРАДА ГРЕШАКА

Када се чита одговор добијен од веб сервиса обавезно треба проверити вредност својства TAPIResponse.error_state. Ако је вредност овог својства различита од 'ok' то значи да је дошло до неке грешке приликом извршавања упита.

Уколико је дошло до грешке, апликација на њу мора реаговати на неки начин. Она не сме наставити рад као да грешке није било. Сама појава грешке значи да позвана функција није извршена до краја, односно да није извршила обраду података као и да није вратила тражене резултате.

Вредност error_state може да назначи клијентској апликацији о којој се грешци ради (статуси грешака су документовани) а на основу чега апликација може да закључи како ће на грешку да реагује. Уз статус грешке додатне информације о грешци се могу прочитати из својстава error_message, result_error_state, result_error_message и result_error_data.

Ако дође до грешке приликом позива АПИ функције, не треба понављати исти упит. Уколико се на основу описа грешке закључи да треба поново позвати исту функцију, треба направити нови упит а не користити стари, који није успео. Нови упит мора имати различит request_id.

Консултације са Инфосис подршком

Уколико дође до непредвиђене грешке и потребна је консултација са Инфоси подршком, пре контакта са подршком треба припремити:

- јасан опис грешке која је настала и контекста у коме је настала

- пример JSON поруке са упитом који доводи до грешке

- пример JSON поруке која представља одговор од АПИ сервера а која је последица грешке

ПРЕНОС ТАБЕЛАРНИХ ПОДАТАКА

Подразумевани формат табеларних податакаје PLAIN_OBJ. Тај формат се добија тако што се табеларни подаци учитају у објектну структуру а затим серијализју у JSON. Са овим форматом се брзо и релативно лако ради, али је проблем што он захтева доста ресурса и није погодан ако табеле садрже много података.

Уместо тога, препоручљиво је користити друге формате који омогућавају пренос података у бинарном облику (видети *TAPIParams.result_database_format*)

Приликом серијализације табеларних података (осим ако се не користи бинарни формат), АПИ сервер не серијализује поља табела која су празна. Клијент не треба да очекује да поља постоје у JSON поруци већ да обрађује само она која постоје. Непостојећа поља треба да добију вредност празно или подразумевану вредност, ако је тако дефинисано.

ИМПЛЕМЕНТАЦИЈА ПОДРШКЕ ЗА ИНФОАПИ ПРОТОКОЛ

Најчешћа грешка коју праве програмери који желе да имплементирају ИнфоАПИ протокол је да почну одмах да праве имплементацију за АПИ функцију коју треба да позивају на продукционом серверу.

Уместо, тога, треба направити имплементацију за примере који су дати у овом упутству и то најпре за АПИ функцију NONE која нити прима нити враћа неке податке већ јој је намена да клијенту омогући да испроба основни протокол за комуникацију.

Тек када програмер успешно позове ову АПИ функцију и од ње добије и исправно прочита одговор, треба да се упушта у имплементацију других механизама протокола: слање параметара функцији, примање резултата функције, примање и слање табеларних података и слично.

ИнфоАПИ протокол је дефинисан по слојевима. Слојеви су механизми и то универзални механизми који раде на исти начин без обзира која АПИ функција је у питању. Конкретни подаци који се шаљу и примају су само један слој, а и они се увек преносе на универзалан начин, тако да ако се уради добра имплементација подршке за протокол, имплементација конкретних функција није компликован посао.

ФОРМАТИРАЊЕ ДАТУМА

Ако се датумска вредност која је типа datetime пролседи такода не садржи и ознаку временске зоне, АПИ ће тај пдоатак третирати као д аје у временској зони 0, а то је универзално време, односно време по Гриничу.

Датум и време увек терба слати у формату *ISO* 8601 (YYYY-MM-DDTHH:MM:SS+HH:MM, где је време изражено у локалној временској зони а +HH:MM означава која је локална временска зона). Време не може садржавати делове секунде.