SAFT-BOOK

Ръководство за внедряване на SAF-T в България

От екипа на baraba.org

Генерирано: March 2026

Манифест на Практика: Защо SAF-T не е за адвокати

През последната година социалните мрежи се напълниха с „експерти“, които тълкуват запетайките в бъдещите наредби за SAF-T. Адвокати правят уебинари, юристи пишат статии за глобите, а консултанти продават „правен анализ“ на технически стандарт.

Истината е една: SAF-T няма да бъде спечелен в съда. Той ще бъде спечелен в базата данни.

SAF-T (Standard Audit File for Tax) не е нов закон за счетоводството. Той не променя принципите на двустранното записване, въведени от Лука Пачоли преди 500 години. SAF-T е **инфраструктурен проект**. Това е протокол за пренос на информация.

Ако счетоводната ви система позволява „свободни съчинения“: 1. Сметки без аналитичност, когато такава се изисква. 2. Контрагенти без ЕИК или с грешен формат на VAT номера. 3. Мерни единици като „бр.“, „бройки“, „пакета“ и „кутии“, вместо стандартизирания код „C62“. 4. Небалансирани счетоводни статии, които „ще се оправят в края на годината“.

... тогава никакъв правен съвет няма да ви помогне. Когато сървърът на НАП получи вашия XML файл, той няма да извика адвокат. Той ще пусне един **XSD валидатор**. Ако данните не съответстват на схемата – файлът се отхвърля. Край. Глоба.

Тази книга е за хората, които реално ще вършат работата: - **За счетоводителите**, които трябва да разберат, че вече са и администратори на данни. - **За програмистите**, които трябва да разберат защо дебитът и кредитът трябва да се изравняват до четвъртия знак след запетаята.

Ние не продаваме правни консултации. Ние изграждаме софтуер (baraba.org), който превръща счетоводния хаос в структуриран ред. И в следващите страници ще ви покажем как точно става това.

От авторите на baraba.org

Глава 0: Генетични дефекти на масовия БГ софтуер — Защо SAF-T ще бъде апокалипсис

Преди повече от 20 години един израелски архитект на ERP системи каза нещо, което днес звучи като пророчество: "*Многото аналитични признаци ще направят системата ви чуплива, неуправляема и непродаваема. По света такова нещо не се прави.*"

Днес, през 2026 г., когато НАП бавно, но сигурно затяга примката със **SAF-T (Standard Audit File for Tax)**, българският софтуерен пазар е пред колапс. Не защото нямаме програмисти, а защото сме заложници на **„Навиците на Баба Яга"**.

0.1. Коя е „Баба Яга" в счетоводството?

Това не е конкретен човек. Това е **манталитетът** на „лелката консултант", която вярва, че софтуерът е цифрова тетрадка с безкрайни полета. Тя иска „Аналитичност 1" за името на шофьора, „Аналитичност 2" за номера на ремаркето и „Аналитичност 20" за това дали шефът е бил в добро настроение, когато е подписал фактурата.

Генетичният дефект: Масовият български счетоводен софтуер е проектиран като `flat table` (плоска таблица) с 20-30 текстови полета (`varchar`), наречени „Аналитичности". Те нямат релация, нямат валидация, нямат логика. Всеки от тези „свободни признаци" е просто текстово поле, в което може да се напише абсолютно всичко — от ЕИК номер до рецепта за баница.

Този архитектурен подход беше роден в началото на 2000-те, когато софтуерните компании слушаха не инженери, а „лелки консултанти". Резултатът е софтуер, който е **генетично неспособен** да произведе структуриран изход.

0.2. „Гъвкавостта на входа" — Мантрата на Баба Яга

Българските счетоводители са свикнали софтуерът да е „пластичен". Те казват на разработчика: *„Искам да мога да пиша каквото си поискам в това поле!"*. Програмистът, за да му е мирна главата, слага един `varchar(max)` или 20 свободни аналитични признака.

Резултат: В базата данни влиза „информационна помия". Един път там е име на шофьор, втори път е „спешно за утре", трети път е празно пространство или невалиден символ.

Тази „гъвкавост" се продаваше като предимство. Рекламите казваха: *„Нашият софтуер е напълно конфигурируем!"*. В действителност това означаваше: *„Нашият софтуер няма никаква вътрешна логика."*

0.3. „Смърт на изхода" — Когато SAF-T почука на вратата

Когато държавата (НАП) каже: *„Сега ни дай тази помия, но я подреди в XML по стандарт на ОИСР*", настъпва **клинична смърт** за системата.

- **Липса на мапване:** Не можеш да автоматизираш изхода, защото нямаш логика на входа. Как ще преведеш „Аналитичност 7" към структурирания XML таг `<AnalysisID>`?
- **Валидационен ад:** XML схемата (XSD) очаква структура, а ти й подаваш „свободно съчинение". Всяко гърмене на валидатора за „латинска буква" или „невалиден формат" е директно следствие от това, че си позволил на входа да влиза всичко.

Аксиомата е проста:

> *Гъвкавостта на входа е смъртната присъда на изхода.*

SAF-T не е просто отчет. Това е **релационен извлечен модел**. Той изисква от изхода да излезе математически точна картина на бизнеса. Ако на входа си позволил „гъвкавост" (демек хаос), на изхода получаваш **невалиден файл**. А невалидният файл в очите на НАП е равен на **глоба**.

0.4. Юридически хватки срещу технически изисквания

В момента масовката от софтуерни компании пращат хора на платени семинари. Там юристи и номенклатурчици от НАП обясняват глобите.

Проблемът: Юристът не може да ти обясни как да мапнеш хаоса от „Аналитичност 7" към структурирания XML таг `<AnalysisID>`. Когато софтуерът ти е строен върху пясък, никаква „юридическа консултация" не може да го направи стабилен.

SAF-T е технически стандарт, а не философско есе.

SAF-T изисква строга йерархия: **MasterFiles** (основни данни) и **GeneralLedgerEntries** (записи в главната книга).

- **В нормалния свят:** Използва се минималистичен подход — Контрагенти, Артикули, ДМА. Всичко е свързано с уникални ключове (ID).
- **В света на Баба Яга:** Подават се нулеви файлове, защото при реални данни системата „гърми". Валидаторът на НАП отхвърля файлове заради една латинска буква, защото софтуерът не сравнява ЕИК, а прави примитивно сравнение на текст.

0.5. Отсрочката — „Потьомкинско село"

Това, че в момента само няколкостотин фирми на големите международни ERP системи подават данни (и то често нулеви), е само димна завеса. Отсрочката е дадена, за да може „масовката" да разбере как да превърне каруцата си в космически кораб.

Спойлер: Няма да успеят с „кръпки". Не можеш да закърпиш архитектурен дефект с ъпдейт. Не можеш да добавиш релационна цялост в система, която е проектирана без нея. Не можеш да строиш XML йерархия върху плоска таблица.

0.6. Предупреждение към собствениците на фирми

> *„Когато купувате или разработвате софтуер, не търсете такъв, който ви позволява да правите всичко. Търсете такъв, който ви принуждава да правите нещата правилно. Защото гъвкавостта на входа, с която са свикнали вашите счетоводители днес, ще бъде смъртната присъда на вашата фирма утре, когато софтуерът на НАП откаже да приеме дигиталния ви хаос."*

0.7. Заключение: Време е за ампутация

Единственият начин да оцелеете в ерата на SAF-T е **минимализмът**:

1. Изхвърлете 20-те аналитичности. Заменете ги с релационни връзки.
2. Спрете да слушате юристи за технически проблеми.
3. Изисквайте от софтуера си валидация на входа, а не „свобода".
4. Подредете мастър данните си **сега**, преди да дойде крайният срок.

SAF-T не е заплаха, а **огледало**. И това, което виждаме в него, е **20-годишно закъснение** в архитектурното мислене на масовия български счетоводен софтуер.

Следващите глави на тази книга ще ви покажат как изглежда **правилният подход** — с код, с примери и без илюзии.

Глава 0A: Вторият фронт — „Дигиталните еничари“ и илюзията за модерност

Ако Глава 0 беше посветена на „старата школа“ и нейните 20 аналитични признака, то тук ще разгледаме втория воденичен камък, който ще смаже българския бизнес при сблъсъка със SAF-T: **„Дигиталните еничари“**.

Това са модерните герои на „облачното“ време. Те са виждали интерфейса на Xero или QuickBooks като оператори, научили са малко SaaS (Software as a Service) терминология и са решили, че могат да построят ERP система с трима PHP програмисти на минимална заплата и красив дизайн в браузъра.

0A.1. „Вкарвачи“, преоблечени като „Архитекти“

Проблемът на тези „модерни“ системи е фундаментален: те са проектирани като **складови програми с ДДС модул**, а не като счетоводни системи.

1. **Документална обоснованост vs. Просто преписване:** В България счетоводството е процес на доказателствена стойност. Не става въпрос просто да „препишеш“ една фактура в красив интерфейс. Големите системи имат концепция за счетоводна статия, хронология и данъчен аспект. „Модерните“ БГ платформи често са просто „агенти по въвеждане“ — те нямат счетоводно ядро (Accounting Kernel).
2. **SaaS мечти върху VPS реалност:** Гиганти като SAP или Oracle не продават просто „полета за писане“, а **контролни механизми**. Те имат строги правила (*Hard Postings*), които не позволяват на потребителя да трие или променя записи със задна дата без одитна следа. Българските „модерни“ копия са „гъвкави“ (разбирай — позволяват всичко), което под капака е същият хаос, но в браузъра.

0A.2. Липсата на „Счетоводно сърце“

Всеки, който е внедрявал сериозно ERP, знае, че логиката се настройва месеци наред (Implementation Guide). „Дигиталните еничари“ са виждали само крайния резултат — полето за въвеждане. Те не знаят как да мапнат един **Chart of Accounts** (Сметкоплан) към **Tax Taxonomy** (Данъчна таксономия).

Затова, когато дойде SAF-T, те се оказват точно толкова безпомощни, колкото и „лелките“ от старата школа.

0A.3. Сблъсъкът със SAF-T: Крахът на интеграцията

Тези системи ще се провалят на SAF-T по една много проста причина: **Те не разбират релацията между Склад и Счетоводство.**

- В техните системи често имаш „Доставка“ и „Фактура“, които живеят собствен живот без твърда връзка.

- SAF-T обаче изисква **Inventory (Stock)** модулът да се връзва математически с **General Ledger** (Главната книга).
- Когато НАП поиска `MovementOfGoods` (движение на стоки), ще лъсне голата истина: техният минималистичен подход е всъщност **липса на данни**. Те са преписвали фактури, но не са водили счетоводство.

0A.4. Генетичният дефект на „Евтиния PHP подход“

През златните години на ИТ сектора (2015–2021), десетки фирми решиха, че могат да преборят световните лидери с „лек“ код. Те създадоха „Google Sheets с интерфейс“. Програмистите им, макар и добри в уеб дизайна, не разбират от:

- **Double-entry bookkeeping** (Двойно записване) на ниво база данни.
- **Fiscal Periods** (Фискални периоди) и забрана за промяна след приключване.
- **Tax Point Date vs. Posting Date.**

В истинското ERP има т.нар. **Subledger-to-Ledger posting**. Когато въвеждаш фактура, системата генерира счетоводна статия в реално време. В БГ „модерните“ системи това е „дупка“, защото връзката често не съществува или е направена „на ръка“ чрез експорт в края на месеца.

0A.5. Заключение: Между два воденични камъка

Българският бизнес днес е притиснат:

1. **Старите системи**, които имат твърде много боклук и шум в данните (заради безкрайните аналитичности).
2. **Новите системи**, които нямат никаква дълбочина и данни (заради липсата на счетоводна логика).

И двата модела ще се счупат в SAF-T. Едните ще се удавят в шума си, другите — в празнотата си.

SAF-T ще бъде „ситото“. Той изисква железен контрол и интеграция. Ако софтуерът ви няма „счетоводен таб“, който се попълва автоматично при всяко движение, вие не притежавате бизнес софтуер. Притежавате дигитален архив, който ще „изгърми“ при първата проверка на НАП.

Глава 1: Новата реалност — Счетоводителят като Data Engineer

1.1. Краят на „Хартиеното мислене“

Години наред българското счетоводство беше фокусирано върху това „документът да е изряден“. Ако имаш хартия с подпис и печат, всичко е наред. Софтуерът беше просто пишеща машина с калкулатор.

С въвеждането на SAF-T BG v1.0.1, НАП премества фокуса от **документа** към **структурата на данните**. Вече не е достатъчно да имате фактура. Трябва да имате: - Път на данните (Data lineage) от първичния запис до Главната книга. - Мапване на всеки ваш вътрешен код към Националната номенклатура.

1.2. Защо „Свободният сметкоплан“ е и благословия, и проклятие?

В България всяко предприятие има право на собствен сметкоплан. Можете да използвате сметка 702 за „Приходи от продажби на стоки“, но можете да я наречете и 7021, 7022 или както решите.

Проблемът: Сървърите на НАП не знаят какво означава вашата сметка 7021. **Решението в SAF-T:** Двойната идентификация. В XML структурата виждаме два ключови тага: - `<nsSAFT:AccountID>`: Тук стои кодът от номенклатурата на НАП (напр. 301, 411, 702). - `<nsSAFT:TaxpayerAccountID>`: Тук стои вашият вътрешен код.

Това е „мостът“. Счетоводителят вече не е просто човекът, който „набива фактури“. Той е архитектът, който трябва да построи този мост. Ако мапването е грешно, цялата ви Главна книга (General Ledger) ще бъде отхвърлена като нелогична.

1.3. Анатомия на една грешка: Пример от практиката

Представете си „експерт“, който ви съветва как да избегнете глобата, но не ви казва, че в SAF-T номенклатурата за мерни единици (UOMTable) не съществува единица „кашон“. Съществува „BX“ (Box) или „CT“ (Carton).

Ако в склада ви пише „кашон“, а в SAF-T експорта изпратите „кашон“, XML файлът ще се „счупи“ при първата проверка. Адвокатът ще ви защитава в съда след 2 години. Програмистът и счетоводителят трябва да оправят базата данни **днес**.

1.4. Процесът на „Почистване“ (Data Cleansing)

Преди изобщо да мислите за експорт на SAF-T, трябва да минете през процеса на изчистване на мастър данните (Master Data): 1. **Контрагенти:** Проверка за валидни ЕИК номера. В SAF-T те се превръщат в уникални идентификатори с префикс „10“ за България. 2. **Номенклатури:** Всички мерни единици, видове плащания и данъчни режими трябва да бъдат заменени със съответните международни кодове (ISO 4217 за валути, UN/ECE за мерни единици). 3.

Сметкоплан: Всяка сметка трябва да има своя „близнак“ в номенклатурата на НАП.

Извод: SAF-T е дигиталната трансформация на счетоводството. Тя изисква технически умения за работа с йерархични структури и релационни бази данни. Всичко останало е шум.

Глава 2: Техническият скелет — Как да разбираме XML, без да сме програмисти

2.1. Защо XML, а не Excel?

Много счетоводители се питат: „Защо НАП не иска просто един голям Excel файл?“. Отговорът е в йерархията. Счетоводните данни не са плоска таблица. Една транзакция има заглавна част (дата, описание) и множество редове (дебит, кредит, аналитичност, ДДС). XML (eXtensible Markup Language) позволява тази информация да бъде „увита“ логически.

2.2. Пространства от имена (Namespaces) — Магическото „nsSAFT:“

Ако отворите един SAF-T файл, ще видите, че всеки таг започва с `nsSAFT:`. Пример: `<nsSAFT:AuditFile>`

Какво означава това на прост език? Това е техническият подпис на стандарта. То казва на приемащия сървър: „Внимание, всичко след мен следва стриктните правила на българската схема v1.0.1“. Ако използвате просто `<AuditFile>` без префикса, сървърът на НАП дори няма да започне да чете файла. Той ще го „изхвърли“ на входа.

2.3. Структурата на „Дървото“

Българският SAF-T файл е разделен на няколко основни клона. Разбирането на тези клони е разликата между успешния счетоводител и този, който ще плаща глоби.

1. **Header (Заглавие):** Тук няма счетоводство. Има метаданни. Кой е софтуерът (SoftwareID)? Коя е версията? Кой е човекът за контакт? Ако софтуерът ви е „самоделка“ без SoftwareID, файлът е невалиден.
2. **MasterFiles (Мастър данни):** Това е „речникът“. Преди да кажете, че сте продали „Домати“, трябва в MasterFiles да сте дефинирали продукта „Домати“ с неговия код и мерна единица.
3. **GeneralLedgerEntries (Главна книга):** Тук е истинското движение на пари. Всяка статия трябва да е абсолютно балансирана. Ако имате разлика от 0.01 лв. поради закръгляне в софтуера, XML-ът ще бъде отхвърлен.
4. **SourceDocuments (Първични документи):** Тук описваме фактурите (SalesInvoices / PurchaseInvoices). Връзката между `TransactionID` в Главната книга и `InvoiceNo` в Документите е това, което НАП ще използва за автоматични проверки.

2.4. XSD — Съдията на мача

XSD (XML Schema Definition) е файлът, който съдържа „законите“ на XML структурата. В папката `SAFT_BG` на нашия проект имаме файла `BG_SAFT_Schema_V_1.0.1.xsd`.

Какво прави XSD? Той проверява: - **Тип на данните:** Очаква ли се число или текст? (Ако в

полето за сума напишете „няма“, XSD ще гръмне). - **Задължителност:** Може ли това поле да е празно? - **Формат:** ЕИК номерът точно 9 или 13 цифри ли е?

Урокът: Преди да пратите файл на НАП, вашият софтуер (като baraba.org) трябва да го прекара през локален XSD валидатор. Ако софтуерът ви не поддържа локална валидация, вие играете руска рулетка.

2.5. Пътят на файла

Експорт от счетоводната система -> Локална XSD валидация -> Логическа проверка (баланс на сумите) -> Цифров подпис -> Изпращане към НАП.

Всеки „експерт“, който ви говори за SAF-T, но не споменава XSD валидация, просто не знае за какво говори.

Глава 3: Счетоводни Номенклатури и Двойният Сметкоплан

3.1. Илюзията за свобода

В България счетоводството е исторически „либерално“ по отношение на номерацията на сметките. Всеки софтуер има собствена логика – някои използват 3-цифрени кодове, други 4-цифрени, трети добавят точки, тирета и поднива.

За SAF-T тази свобода приключва.

НАП въведе официална номенклатура от точно **360 стандартни сметки** (от 101 до 999). Ако вашият софтуер ви казва „спокойно, ние ще изпратим вашите сметки“, той ви лъже. Вие трябва да изпратите **техните** сметки, към които са закачени **вашите** данни.

3.2. Мапване (Mapping) — Изкуството на съответствието

В сърцето на SAF-T BG лежи концепцията за двойната идентификация. В XML схемата всяка сметка се описва така:

```
<nsSAFT:Account>
    <nsSAFT:AccountID>411</nsSAFT:AccountID>
    <nsSAFT:AccountDescription>Клиенти</nsSAFT:AccountDescription>
    <nsSAFT:TaxpayerAccountID>411.01.002</nsSAFT:TaxpayerAccountID>
    <nsSAFT:AccountType>Bifunctional</nsSAFT:AccountType>
</nsSAFT:Account>
```

Разшифровка за практици: 1. `<nsSAFT:AccountID>`: Това е „униформата“. Тук задължително трябва да стои код от списъка на НАП. Ако вашата сметка е „411.01“, тук трябва да пише „411“. 2. `<nsSAFT:TaxpayerAccountID>`: Това е вашето „име“. Тук стои реалният код от вашата система (напр. 411.01.002).

Защо адвокатите не разбират това? Защото това не е правен въпрос, а въпрос на **релационна цялост**. Ако мапнете грешно сметка за активи (раздел 2) към сметка за разходи (раздел 6), автоматичните контроли на НАП ще светнат в червено още при качването на файла.

3.3. Капанът на аналитичностите

SAF-T изисква аналитичност за определени сметки (доставчици, клиенти, персонал, банки). Много счетоводители водят „синтетично“ счетоводство в Excel и „аналитично“ някъде другаде. С SAF-T това е невъзможно. Всяка транзакция в Главната книга (`GeneralLedgerEntries`) трябва да носи в себе си идентификатора на контрагента (`CustomerID` или `SupplierID`).

Ако системата ви няма връзка 1:1 между сметкоплана и картотеката на контрагентите, вие няма да можете да генерирате валиден файл. **Това е софтуерен проблем, а не законов.**

3.4. Данъчни кодове (TaxTable)

Забравете за свободните текстове като „ДДС 20%“ или „Освободена доставка по чл. 86“. В SAF-T всяка операция има точно определен **TaxCode**. - **100010** – Общ код за ДДС. - **100211** – Облагаеми доставки със ставка 20%. - **100213** – Облагаеми доставки със ставка 9%.

Счетоводителят вече трябва да знае не само закона за ДДС, но и техническите кодове на НАП. Ако софтуерът ви не поддържа автоматичното им мапване, ще прекарате седмици в ръчна редакция на XML файлове.

Извод: Успехът при SAF-T зависи от това колко подреден е вашият „дигитален архив“. Сметкопланът е скелетът на това подреждане.

Глава 4: Материални запаси — Кошмарът „При поискване“

4.1. Разликата между Месечен и On-Demand отчет

Докато месечният SAF-T файл е до голяма степен заместник на дневниците по ДДС и Оборотната ведомост, отчетът **„При поискване“ (On Demand)** е дигитална ревизия на склада.

Ако НАП ви поиска този файл, вие имате кратък срок да предоставите пълната история на всеки един болт, гайка или литър гориво в базата ви данни. Тук юридическите съвети са безсилни – или имате записи в склада, или нямате.

4.2. Секция PhysicalStock — Моментната снимка

В този раздел описвате наличностите към определена дата. Ключовото тук не е само количеството, а **мерната единица**.

Технически казус: Вие купувате гориво в тонове, но го изписвате в литри. В SAF-T трябва да подадете: - `<nsSAFT:UOMBase>`: Базова мерна единица. - `<nsSAFT:UOMStandard>`: Стандартна мерна единица (по ISO). - `<nsSAFT:UOMToUOMBaseConversionFactor>`: Коефициент на преобразуване.

Ако счетоводителят не разбира тези технически параметри, файлът ще показва абсурдни наличности, което автоматично задейства ревизия.

4.3. MovementOfGoods — Следата на парите и стоките

Всеки запис в склада трябва да е свързан със счетоводна статия в Главната книга. В XML структурата това се постига чрез тага `<nsSAFT:TransactionID>`.

Ако сте заприходили стока, но нямате съответната статия Дебит 304 / Кредит 401 със същия ID, системата на НАП ще открие несъответствието за милисекунди. Това е т.нар. **Cross-module validation**. Вече не е възможно складът да „бяга“ от счетоводството.

4.4. Проблемът с „Разнородните данни“

Най-големият идиотизъм, който се тиражира в социалните мрежи, е че „някак си ще нагласим данните“. При материалните запаси това е невъзможно, защото SAF-T изисква: 1. **WarehouseID:** Точен идентификатор на склада. 2. **ProductCode:** Вашият вътрешен код на продукта. 3. **GoodsServicesID:** Стандартен код на стоката (напр. 01 за стоки).

Ако в софтуера ви един и същ продукт има три различни имена (напр. „Захар“, „Захар на кристали“, „Захар 1кг“), вие ще подадете три различни продукта в SAF-T. Това ще доведе до хаос в наличностите и потенциални глоби за липси или излишъци.

4.5. Как да се подготвим?

Подготовката за материалните запаси започва от **Инвентаризацията на Мастър данните**: - Уеднаквяване на имената на продуктите. - Стандартизиране на мерните единици по номенклатурата на НАП. - Осигуряване на твърда връзка между складов документ и счетоводна операция.

Урокът: SAF-T On Demand превръща счетоводството в реално време (Real-time accounting) в задължително условие за оцеляване. Всяко „отлагане за по-късно“ е директен път към санкция.

Глава 5: Номенклатури и Счетоводни Кореспонденции — Логическият център

5.1. Какво са „Кореспонденции“ в света на SAF-T?

В класическото счетоводство кореспонденцията е връзката между две сметки (Дебит 601 / Кредит 301). В SAF-T обаче НАП иска да знае **защо** се случва това движение. За целта се използват т.нар. **Movement Types** (Типове движения).

Пример за номенклатура на НАП за склад (On Demand): - **10** — Покупка - **30** — Продажба - **50** — Вътрешно преместване

5.2. Автоматизация на мапинга (The Mapping Engine)

За да не се налага счетоводителят ръчно да избира код за всяка хилядна фактура, в baraba.org използваме „Мапинг двигател“. Ето как изглежда логиката му, описана чрез **SurrealQL (NoSQL)**:

```
-- Дефинираме правило: Всяко движение от тип 'Продажба'
-- автоматично кореспондира с определени сметки
DEFINE TABLE saft_correspondence SCHEMAFULL;
DEFINE FIELD movement_type ON saft_correspondence TYPE string; -- Код 30
DEFINE FIELD debit_account ON saft_correspondence TYPE record<account>; -- 411
DEFINE FIELD credit_account ON saft_correspondence TYPE record<account>; -- 702

-- Търсене на правилото чрез Graph Relation
SELECT ->rules_for_saft->(saft_account WHERE code = '702')
FROM movement_type:sale;
```

Счетоводният аспект: Софтуерът трябва да знае, че ако изпишем стока с код „Продажба“, той трябва да потърси в SAF-T номенклатурата съответствието за приходи. Ако счетоводителят промени своята сметка 702 на 7021, мапингът в настройките трябва автоматично да се актуализира.

5.3. Настройки на фирмата (System Settings)

SAF-T изисква и много не-счетоводни настройки, които обаче променят XML структурата. Пример в **Rust** за управление на тези настройки:

```
#[derive(Serialize, Deserialize)]
pub struct SaftSettings {
    pub tax_accounting_basis: String, // 'A' за Търговски предприятия
    pub uom_map: HashMap<String, String>, // Локално 'бр.' -> SAF-T 'C62'
}

// В baraba.org проверяваме настройките преди експорт
pub fn validate_uom(local_unit: &str, settings: &SaftSettings) -> String {
    settings.uom_map.get(local_unit)
        .cloned()
        .unwrap_or_else(|| "C62".to_string()) // Fallback към 'Units'
}
```

5.4. Казусът с „Мерните единици“

Това е техническият кошмар на всеки счетоводител. Вие имате „кг“, „килограма“, „КГ.“. НАП признава само „KGM“. Тук идва ролята на **Python** за предварителна обработка на данни (Data Wrangling):

```
# Python скрипт за уеднаквяване на мерни единици преди импорт в baraba.org
import pandas as pd

def clean_uom(value):
    mapping = {
        'кг': 'KGM',
        'бр': 'C62',
        'л': 'LTR'
    }
    return mapping.get(value.lower().strip(), 'C62')

df = pd.read_excel("sklad.xlsx")
df['SAFT_UOM'] = df['Merna_Edinica'].apply(clean_uom)
```

Извод: Номенклатурите не са просто списъци. Те са релационни масиви, които трябва да бъдат синхронизирани между вашата база данни и изискванията на НАП. Адвокатите виждат списъка, счетоводителят вижда работата, а програмистът вижда алгоритъма.

Глава 6: Техническа Реализация — Кодът зад отчетите

6.1. Защо Rust + SurrealDB? (Архитектурен избор)

Когато генерирате XML файл за НАП, грешка в типа на данните означава отхвърляне. Rust не позволява такива грешки по време на компилация. SurrealDB дава гъвкавост на NoSQL с мощта на релационни заявки.

Стекът на baraba.org: - **Dioxus 0.7** — fullstack framework (SSR + WASM), едно Rust приложение за сървър и клиент - **SurrealDB** — документна база данни с Graph заявки и вградени типове - **quick_xml** — streaming XML writer, който гарантира encoding и escaping - **Server Functions** (`#[server]`) — кодът за генерация живее на сървъра, клиентът вижда само UI

6.2. Моделът на данните (Реален код)

Ето как изглежда `Company` структурата в baraba.org — тя носи всички полета, необходими за SAF-T Header:

```
// src/models.rs — Реален код от baraba.org
#[derive(Debug, Clone, Serialize, Deserialize, PartialEq)]
pub struct Company {
    pub id: Option<String>,
    pub name: String,
    pub eik: String,
    pub vat_number: Option<String>,
    pub address: Option<String>,
    pub city: Option<String>,
    pub country: Option<String>,
    pub post_code: Option<String>,
    pub phone: Option<String>,
    pub email: Option<String>,
    pub manager_name: Option<String>,
    pub street_name: Option<String>,        // SAF-T: отделно улица
    pub building_number: Option<String>,     // SAF-T: отделно номер
    pub region: Option<String>,             // SAF-T: BG-22, BG-23...
    pub tax_accounting_basis: String,       // "A" за начисления
    pub software_company_name: Option<String>, // "baraba.org"
    pub software_id: Option<String>,        // "BARABA"
    pub software_version: Option<String>,   // "1.0"
    pub currency: String,                   // "BGN" или "EUR"
    pub is_vat_registered: bool,
    pub is_part_of_group: Option<String>,
    // ... и още 20 полета
}
```

Забележете: Полета като `street_name`, `building_number`, `region` съществуват **специално за SAF-T** — в класическия адрес те са едно поле, но XML схемата ги иска разделени.

6.3. Генериране на SAF-T ID (Префикс логика)

Най-критичната функция — как ЕИК/ДДС номер се трансформира в SAF-T идентификатор:

```
// src/pages/saft_export.rs — Реален код от baraba.org
fn generate_saft_id(cp: &Counterpart) -> String {
    let country = cp.country.as_deref().unwrap_or("BG");

    // Префикс 10: Български ЕИК
    if let Some(ref eik) = cp.eik {
        if !eik.is_empty() && (country == "BG" || country.is_empty()) {
            return format!("10{}", eik);
        }
    }

    // Префикс 11: ЕС ДДС номер (не-BG)
    if let Some(ref vat) = cp.vat_number {
        if !vat.is_empty() {
            let eu_countries = ["AT","BE","BG","HR","CY","CZ","DK","EE",
                "FI","FR","DE","GR","HU","IE","IT","LV","LT","LU",
                "MT","NL","PL","PT","RO","SK","SI","ES","SE"];
            if eu_countries.contains(&country) && country != "BG" {
                return format!("11{}{}", country, vat);
            }
            // Префикс 12: Извън ЕС
            return format!("12{}{}", country, vat);
        }
    }

    // Префикс 13: Само ЕИК, чуждестранен
    if let Some(ref eik) = cp.eik {
        if !eik.is_empty() {
            return format!("13{}", eik);
        }
    }

    // Префикс 15: Fallback
    format!("15{}", cp.id.as_deref().unwrap_or("0"))
}
```

Счетоводният аспект: Тази функция решава проблема с различните типове контрагенти автоматично. Счетоводителят въвежда ЕИК и държава — софтуерът избира правилния SAF-T префикс.

6.4. XML генераторът (Streaming подход)

baraba.org не строи XML дърво в паметта. Използваме **streaming writer** — пишем елемент по елемент, което работи за файлове от всякакъв размер:

```
// src/pages/saft_export.rs — Реален код от baraba.org

// Помощни функции — елегантни и безопасни
fn w<W: std::io::Write>(writer: &mut quick_xml::Writer<W>,
    name: &str, text: &str) -> Result<(), quick_xml::Error>
{
    use quick_xml::events::{BytesEnd, BytesStart, BytesText, Event};
    writer.write_event(Event::Start(BytesStart::new(name)))?;
    writer.write_event(Event::Text(BytesText::new(text)))?; // auto-escapes &, <, >
    writer.write_event(Event::End(BytesEnd::new(name)))?;
    Ok(())
}

fn open<W: std::io::Write>(writer: &mut quick_xml::Writer<W>,
    name: &str) -> Result<(), quick_xml::Error>
{
    writer.write_event(Event::Start(BytesStart::new(name)))
}
```

```
fn close<W: std::io::Write>(writer: &mut quick_xml::Writer<W>,
    name: &str) -> Result<(), quick_xml::Error>
{
    writer.write_event(Event::End(BytesEnd::new(name)))
}
```

С тези 3 функции се изгражда целият 1300-редов SAF-T генератор:

```
fn build_saft_xml(data: &SaftData) -> Result<String, quick_xml::Error> {
    use quick_xml::Writer;
    let mut x = Writer::new_with_indent(Cursor::new(Vec::new()), b' ', 2);

    // XML декларация
    x.write_event(Event::Decl(BytesDecl::new("1.0", Some("UTF-8"), None)))?;

    // Root с namespace на НАП
    let mut root = BytesStart::new("nsSAFT:AuditFile");
    root.push_attribute(("xmlns:nsSAFT", "mf:nra:dgti:dxxxx:declaration:v1"));
    x.write_event(Event::Start(root))?;

    build_header(&mut x, data)?;          // Заглавна част
    build_master_files(&mut x, data)?;     // Сметки, клиенти, доставчици
    build_general_ledger_entries(&mut x, data)?; // Главна книга
    build_source_documents(&mut x, data)?; // Фактури, плащания

    close(&mut x, "nsSAFT:AuditFile")?;
    Ok(String::from_utf8_lossy(&x.into_inner().into_inner()).to_string())
}
```

6.5. Двойната идентификация на сметките (Мапинг в действие)

Ето как baraba.org реализира мапването `AccountID` ↔ `TaxpayerAccountID` в реално време:

```
// При генериране на всеки ред в Главната книга
for (i, line) in je.lines.iter().enumerate() {
    // Търсим сметката на потребителя
    let standard_code = if let Some(acc) = data.accounts.iter()
        .find(|a| a.id.as_deref() == Some(&line.account_id))
    {
        // Ако има SAF-T mapping — вземаме кода от НАП номенклатурата
        if let Some(ref saft_id) = acc.saft_account_id {
            data.saft_accounts.iter()
                .find(|s| s.id.as_deref() == Some(saft_id))
                .map(|s| s.code.as_str())    // "411" от НАП
                .unwrap_or(&acc.code)         // fallback
        } else {
            &acc.code  // Ако няма mapping — ползваме вашия код
        }
    } else {
        line.account_code.as_deref().unwrap_or(&line.account_id)
    };

    // В XML: AccountID = код на НАП, TaxpayerAccountID = ваш код
    w(x, "nsSAFT:AccountID", standard_code)?;
    // TaxpayerAccountID се попълва от acc.code
}
```

Счетоводният аспект: Вашата сметка `411.01.002` отива в `TaxpayerAccountID`. Стандартната

`411` отива в `AccountID`. Софтуерът прави превода автоматично, стига сметката да има mapping в настройките.

6.6. Трите лица на SAF-T (Monthly / OnDemand / Annual)

baraba.org генерира различен XML в зависимост от типа отчет:

```
// Избор на MasterFiles секция според типа
let master_element = match data.report_type.as_str() {
    "annual"   => "nsSAFT:MasterFilesAnnual",    // ДМА + амортизации
    "ondemand" => "nsSAFT:MasterFilesOnDemand",   // Складови наличности
    _          => "nsSAFT:MasterFilesMonthly",     // Стандартен месечен
};

open(x, master_element)?;
build_gl_accounts(x, data)?;    // Сметки — винаги
build_customers(x, data)?;      // Клиенти — винаги
build_suppliers(x, data)?;      // Доставчици — винаги
build_tax_table(x)?;            // Данъчни кодове — винаги
build_uom_table(x)?;            // Мерни единици — винаги

if data.report_type == "ondemand" {
    build_physical_stock(x, data)?;    // Само при On Demand
}
if data.report_type == "annual" {
    build_assets(x, data)?;            // Само при годишен
}
close(x, master_element)?;
```

6.7. SurrealDB: Заявките за събиране на данните

Преди генерацията, baraba.org събира всичко необходимо с няколко заявки:

```
// Журнални статии С вложени редове (subquery)
let je_sql = format!(
    "SELECT *, \
       (SELECT * FROM journal_entry_line WHERE journal_entry_id = $parent.id) AS lines \
     FROM journal_entry \
     WHERE company_id = '{}' AND status = 'POSTED' AND date CONTAINS '{}' \
     ORDER BY date ASC",
    company_id, period  // period = "2026-01"
);
let entries: Vec<JournalEntry> = query(&je_sql).await?
    .into_iter()
    .filter_map(|v| serde_json::from_value(v).ok())
    .collect();
```

В класически SQL това би изисквало JOIN + GROUP BY. В SurrealDB вложената заявка (`$parent.id`) директно връща редовете вътре в записа.

А ето и слоят за комуникация с базата (`src/db.rs`):

```
// HTTP клиент към SurrealDB с retry логика
pub async fn query(sql: &str) -> Result<Vec<Value>> {
    let resp = HTTP
        .post(&CONFIG.url)           // http://127.0.0.1:8000/sql
```

```
        .basic_auth(&CONFIG.user, Some(&CONFIG.pass))
        .header("surreal-ns", &CONFIG.ns)  // namespace: test
        .header("surreal-db", &CONFIG.db)  // database: test
        .body(sql.to_string())
        .send()
        .await?;
    // ... parse JSON response, handle errors, retry on 5xx
}
```

6.8. Валидацията преди експорт (Pre-flight check)

baraba.org не чака НАП да отхвърли файла. Валидацията се случва **преди** генерацията:

```
#[server]
async fn validate_saft_export(company_id: String, year: i32, month: i32)
    -> Result<SaftValidationResult, ServerFnError>
{
    let mut errors = Vec::new();
    let mut warnings = Vec::new();

    // 1. Проверка на фирмени данни
    let company: Company = /* fetch from DB */;
    if company.eik.is_empty() {
        errors.push("Компанията няма попълнен ЕИК".into());
    }
    if !company.is_vat_registered {
        warnings.push("Компанията не е регистрирана по ДДС".into());
    }

    // 2. Проверка на сметки без SAF-T mapping
    let accounts: Vec<Account> = /* fetch active accounts */;
    let without_saft = accounts.iter()
        .filter(|a| a.saft_account_id.is_none()).count();
    if without_saft > 0 {
        warnings.push(format!("{} сметки нямат SAF-T mapping", without_saft));
    }

    // 3. Проверка на контрагенти без ЕИК
    let counterparts: Vec<Counterpart> = /* fetch all */;
    let cp_no_id = counterparts.iter()
        .filter(|c| c.eik.is_none() && c.vat_number.is_none()).count();
    if cp_no_id > 0 {
        warnings.push(format!("{} контрагенти нямат ЕИК или ДДС номер", cp_no_id));
    }

    Ok(SaftValidationResult {
        valid: errors.is_empty(),
        errors,
        warnings,
        journal_entries_count: entries.len() as i32,
        invoices_count: invoices.len() as i32,
        counterparts_count: counterparts.len() as i32,
        accounts_count: accounts.len() as i32,
    })
}
```

Потребителят вижда резултата в UI — грешки в червено, предупреждения в жълто, статистика в карти.

6.9. Данъчната таблица (Hardcoded номенклатура)

Данъчните кодове са стандартни за всички фирми — вградени са директно в кода:

```
fn build_tax_table<W: std::io::Write>(x: &mut quick_xml::Writer<W>)
    -> Result<(), quick_xml::Error>
{
    open(x, "nsSAFT:TaxTable")?;
    open(x, "nsSAFT:TaxTableEntry")?;
    w(x, "nsSAFT:TaxType", "100010")?;     // ДДС
    w(x, "nsSAFT:Description", "ДДС")?;

    let codes = [
        ("100211", "20.00", "ДО със ставка 20%"),
        ("100213",  "9.00", "ДО със ставка 9%"),
        ("100214",  "0.00", "ДО със ставка 0%"),
        ("100219",  "0.00", "Освободени доставки"),
    ];
    for (code, rate, desc) in &codes {
        open(x, "nsSAFT:TaxCodeDetails")?;
        w(x, "nsSAFT:TaxCode", code)?;
        w(x, "nsSAFT:Description", desc)?;
        w(x, "nsSAFT:TaxPercentage", rate)?;
        close(x, "nsSAFT:TaxCodeDetails")?;
    }
    close(x, "nsSAFT:TaxTableEntry")?;
    close(x, "nsSAFT:TaxTable")?;
    Ok(())
}
```

6.10. Python за одит на генерирания XML

Преди да „запечатате" месечния отчет, можете да пуснете одитни скриптове. Ето пример как Python проверява дали ДДС-то съответства на данъчната основа:

```
import xml.etree.ElementTree as ET

def audit_saft_xml(file_path):
    tree = ET.parse(file_path)
    root = tree.getroot()
    ns = {'ns': 'mf:nra:dgti:dxxxx:declaration:v1'}

    # Проверка 1: Баланс на Главната книга
    total_d = float(root.find('.//ns:GeneralLedgerEntries/ns:TotalDebit', ns).text)
    total_c = float(root.find('.//ns:GeneralLedgerEntries/ns:TotalCredit', ns).text)
    if abs(total_d - total_c) > 0.01:
        print(f"ГРЕШКА: Дебит ({total_d}) != Кредит ({total_c})")

    # Проверка 2: ДДС съответствие
    for line in root.findall('.//ns:TransactionLine', ns):
        tax_info = line.find('.//ns:TaxInformation', ns)
        if tax_info is not None:
            base = float(tax_info.find('ns:TaxBase', ns).text)
            rate = float(tax_info.find('ns:TaxPercentage', ns).text)
            vat = float(tax_info.find('.//ns:TaxAmount/ns:Amount', ns).text)
            expected = round(base * rate / 100, 2)
            if abs(expected - vat) > 0.01:
                rec_id = line.find('ns:RecordID', ns).text
                print(f"ГРЕШКА: Ред {rec_id}: база {base} × {rate}% = {expected}, но ДДС = {vat}

    # Проверка 3: Всяка фактура има TransactionID
    for inv in root.findall('.//ns:Invoice', ns):
        tid = inv.find('ns:TransactionID', ns)
        if tid is None or not tid.text:
            inv_no = inv.find('ns:InvoiceNo', ns).text
            print(f"ПРЕДУПРЕЖДЕНИЕ: Фактура {inv_no} няма TransactionID")
```

```
audit_saft_xml("output_saft.xml")
```

6.11. Миграциите (Идемпотентна инициализация)

baraba.org използва SurrealDB миграции, които се изпълняват автоматично при стартиране:

```
-- migrations/001_init.surql — 360 сметки от НАП номенклатурата
DEFINE TABLE saft_account SCHEMAFULL;
DEFINE FIELD OVERWRITE code ON saft_account TYPE string;
DEFINE FIELD OVERWRITE name ON saft_account TYPE string;
DEFINE FIELD OVERWRITE account_type ON saft_account TYPE string;
DEFINE INDEX idx_saft_code ON saft_account FIELDS code UNIQUE;

-- Seed: Зареждане на стандартните сметки
UPSERT saft_account:101 SET code = '101', name = 'Основен капитал', account_type = 'Liability';
UPSERT saft_account:411 SET code = '411', name = 'Клиенти', account_type = 'Bifunctional';
UPSERT saft_account:702 SET code = '702', name = 'Приходи от стоки', account_type = 'Sale';
-- ... общо 360 реда
```

Ключов урок: Използваме `UPSERT` (не `CREATE`), за да може миграцията да се изпълни многократно без дубликати. `DEFINE FIELD OVERWRITE` гарантира идемпотентност.

6.12. Бъдещето: Автоматизация през API

SAF-T не е крайна цел, а само етап. Софтуерът на бъдещето ще предава данните към НАП чрез защитени API канали в реално време. Това ще премахне нуждата от месечни „кампании" по подаване на файлове.

Извод за техническите лица: Целият SAF-T генератор на baraba.org е ~1300 реда Rust код. Той покрива Monthly, On Demand и Annual отчети. Всичко е type-safe, streaming, и валидирано преди генерация. Ако вашият софтуер е изграден върху „кръпки", SAF-T ще ги разкрие.

Глава 7: Епилог — Краят на търговията със страх

През последните месеци бяхме свидетели на невиждана кампания по насаждане на страх. Адвокатски кантори, които не могат да напишат един SQL SELECT, обясняваха на счетоводителите как НАП ще ги глоби за "неправилно тълкуване". Консултанти продаваха уебинари, на които четоха на глас екселски таблици.

Време е да сложим край на това.

Истината е в кода, не в параграфа

SAF-T не е правна революция. Той е дигитална санитария. Ако вашата система е изградена правилно – ако използвате релационни бази данни (или модерни NoSQL решения като SurrealDB), ако имате строга типизация (като в Rust), ако сте мапнали вашите 411 и 702 към номенклатурата на НАП и ако не изписвате "кашони" в склад за "литри" – **вие сте в безопасност.**

XSD валидаторът няма адвокат. Той не приема подкупи, не се впечатлява от титли и не се интересува от правни становища. Той чете тагове. `<nsSAFT:DebitAmount>` трябва да е равно на `<nsSAFT:CreditAmount>`. Ако е така – минавате.

Счетоводителят на бъдещето

Вие вече не сте просто оператори на данни. Вие сте **Data Engineers**. Вие управлявате потоци от информация, гарантирате интегритета на базите данни и работите в реално време.

Не позволявайте на хора извън вашата професия да ви учат на счетоводна архитектура. Изисквайте от създателите на вашия софтуер да ви дадат инструменти за мапинг, за локална валидация и за автоматизация, вместо да ви прехвърлят отговорността.

Ние в baraba.org направихме точно това. Пренаписахме системата си, за да отговаря на стандарта, без да ви превръщаме в програмисти. Защото вярваме, че добрият софтуер работи за вас, а не вие за него.

Action Plan: 3 стъпки от хаос към готовност

Стига приказки. Ето конкретно какво правите — днес, този месец и оттук нататък.

Стъпка 1: Веднага (тази седмица)

Целта е да разберете **колко зле стоите** — без илюзии, без оправдания.

#	Действие	Какво очаквате	Глава
1	**Извадете пълния сметкоплан** от софтуера си — списък на всички сметки с наименования, нива, аналитичности	Плосък CSV/Excel файл. Ако софтуерът не може да го даде — това е първият червен флаг	Гл. 3

2	**Проверете дали имате 360-те сметки на НАП** — отворете Приложение А и сравнете. Колко от вашите сметки нямат съответствие?	Число: „47 от 120-те ни сметки нямат мапинг"	Гл. 3
3	**Извадете списък на контрагентите** — всички клиенти и доставчици с ЕИК/ДДС номер	Колко нямат ЕИК? Колко са дублирани? Колко са „Клиент 1", „Разни" или „Каса"?	Гл. 1
4	**Проверете мерните единици** — какви UOM използвате в складовия модул	Колко са „бр.", колко „бройки", колко „Бр" — и дали изобщо можете да ги конвертирате до UN/ECE кодове	Гл. 4, 5
5	**Поискайте от софтуерния доставчик** писмен отговор: „Кога ще имате SAF-T експорт по Заповед № ЗМФ-...?"	Ако отговорът е неясен — започнете да търсите алтернатива. Сега, не след 6 месеца	Гл. 0, 0А

Резултат след Стъпка 1: Имате 5 файла на бюрото си и реалистична оценка на мащаба на проблема. Нищо не е поправено, но знаете какво трябва да се поправи.

Стъпка 2: В рамките на 30 дни

Целта е **да затворите критичните дупки** — тези, които ще провалят първото подаване.

#	Действие	Как го правите	Глава
1	**Мапнете сметкоплана** — всяка ваша сметка трябва да има `AccountID` (от НАП) и `TaxpayerAccountID` (ваш)	Ръчно в Excel или с инструмент за мапинг. Ако имате 500 сметки, очаквайте 2-3 работни дни. Без това — няма SAF-T	Гл. 3
2	**Изчистете контрагентите** — слейте дублираните, попълнете липсващите ЕИК, генерирайте SAF-T ID (10XXXXXXXXX за БГ, 11XXXXXXXXX за ЕС)	Ползвайте Python скрипта за дедупликация от Гл. 11.1 или ръчно в софтуера	Гл. 1, 11
3	**Настройте данъчните кодове (TaxTable)** — заменете свободните текстове („ДДС 20%", „без ддс", „ддс вкл.") с кодовете от Приложение Б	Вътрешна номенклатура в софтуера. Ако софтуерът не поддържа структурирани данъчни кодове — виж Стъпка 1, точка 5	Гл. 3
4	**Стандартизирайте мерните единици** — „бр." → C62, „кг" → KGM, „кашон" → CT. Без изключения	Python скрипт от Гл. 5 или ръчна таблица за конверсия	Гл. 4, 5
5	**Направете пробно подаване** — генерирайте XML за последния приключен месец и валидирайте	Ако нямате SAF-T генератор — използвайте ръчен шаблон от Приложение Е и попълнете поне	Гл. 2, 6, 10

	срещу XSD	Header + MasterFiles	
6	**Пуснете risk score анализа** от Гл. 11.6 върху данните си — амортизации, вечни салда, залежали запаси, смесени разходи	Python скриптовете от секции 11.6.1–11.6.5. Ако резултатът е HIGH/CRITICAL — имате данъчни проблеми, които SAF-T само ще направи видими	Гл. 11

Резултат след Стъпка 2: Имате работещ мапинг, чисти номенклатури и поне един валидиран XML файл. Знаете кои данъчни политики са рискови.

Стъпка 3: Постоянен контрол (всеки месец)

Целта е **да не се върнете назад** — защото данните се замърсяват с всяка нова операция.

#	Контрол	Честота	Как
1	**Нов контрагент = попълнен ЕИК + SAF-T ID**	При всяко въвеждане	Правило в софтуера: нов контрагент без ЕИК → предупреждение. Без SAF-T ID → грешка
2	**Съгласуване: Оборотна ведомост ↔ Журнал ↔ ДДС дневници**	Месечно, преди подаване	`TotalDebit == TotalCredit` в GeneralLedgerEntries. Сума на фактурите по ДДС дневник == сума в SourceDocuments. Ако не бият — не подавайте
3	**Нови сметки = веднага мапнати**	При откриване на нова сметка	Ако счетоводителят открие нова подсметка — тя трябва да има `AccountID` от НАП в рамките на същия ден
4	**Складова дисциплина: без отрицателни количества**	Месечно	Справка за артикули с qty < 0. Ако има — коригирайте преди приключване на периода
5	**XSD валидация на генерирания файл**	Преди всяко подаване	Локална валидация (Rust/Python от Гл. 10), не на сървъра на НАП. Руска рулетка не се играе с данъчни декларации
6	**Risk score мониторинг**	Тримесечно	Пуснете анализа от Гл. 11.6. Следете тренда: ако score расте — имате системен проблем

Контролен списък „Готовност за SAF-T"

Преди първото подаване, всяка точка трябва да е ✓:

```
НОМЕНКЛАТУРИ
  [ ] Сметкоплан: всяка сметка има AccountID (НАП) + TaxpayerAccountID (фирмен)
  [ ] Контрагенти: всеки има SAF-T ID (10/11/12/13/15 префикс)
  [ ] Контрагенти: няма дублирани записи за един и същ ЕИК
```

```
  [ ] Данъчни кодове: TaxTable е попълнена, няма свободни текстове
  [ ] Мерни единици: всички конвертирани до UN/ECE кодове (C62, KGM, LTR…)

ДАННИ
  [ ] Оборотна ведомост: TotalDebit == TotalCredit
  [ ] Журнал: всяка статия има поне един ред с Debit и един с Credit
  [ ] Фактури: всяка има InvoiceNo, InvoiceDate, CustomerID/SupplierID
  [ ] Плащания: всяко плащане е свързано с документ (фактура/РКО/ПКО)
  [ ] Складови наличности: няма артикули с qty < 0

ВАЛИДАЦИЯ
  [ ] XML файлът минава XSD валидация без грешки
  [ ] NumberOfEntries съвпада с реалния брой записи
  [ ] TotalDebit/TotalCredit в Header съвпадат със сумите в журнала
  [ ] Всички CustomerID/SupplierID в журнала съществуват в MasterFiles
  [ ] Всички AccountID в журнала съществуват в GeneralLedgerAccounts

ДАНЪЧНИ РИСКОВЕ (Гл. 11.6)
  [ ] Амортизации: счетоводен полезен живот не е агресивно по-кратък от данъчния
  [ ] Разчетни сметки: няма салда без движение > 3 години
  [ ] Запаси: няма залежали артикули > 12 месеца без обезценка/отписване
  [ ] Смесени разходи: има аналитичност лично/служебно за горива, наеми, телеком
  [ ] Обобщен risk score: LOW или MEDIUM
```

Какво да изискате от софтуерния доставчик

Ако софтуерът ви не може да направи нещо от горния списък — проблемът не е във вас. Ето минималните изисквания към софтуера:

1. **Експорт на SAF-T XML** — месечен, годишен, On Demand. Не „скоро", не „в следващата версия". Сега.
2. **Двоен сметкоплан** — поле за AccountID (НАП) до всяка фирмена сметка
3. **Структурирана TaxTable** — данъчни кодове с ставки, не свободен текст
4. **SAF-T ID генератор** — автоматично генериране на идентификатор за контрагенти по правилата 10/11/12/13/15
5. **Вградена XSD валидация** — преди подаване, не след
6. **Стандартни мерни единици** — UN/ECE кодове, не „бр." и „кашони"
7. **Складова интеграция** — PhysicalStock и MovementOfGoods да се генерират от складовия модул, не от Excel
8. **Одитна следа** — кой, кога, какво е променил. Без това SAF-T е безсмислен

Ако доставчикът ви не може да предостави поне точки 1-5 — **сменете доставчика**. Не утре. Тази седмица.

> **Игнорирайте шума. Подредете си данните. И продължавайте напред.**

Глава 8: Data Engineering — От хаоса към структурирани данни

> *Пълното съдържание на тази глава е в CHAPTER_8.md*

8.1. Счетоводителят като Data Engineer

SAF-T трансформира счетоводството от "цифрова писане" в "данъчен ETL процес". Всяка фирма вече трябва да има **Data Pipeline**:

```
[Източници] → [Extract] → [Transform] → [Validate] → [Load] → [SAF-T Export]
```

8.2. Архитектура на данните за SAF-T

Пълен SurrealDB схематичен модел с DEFINE TABLE за всички сущности: company, counterpart, account, journal_entry, invoice.

8.3. Rust: Типобезопасен модел на данните

Пълни struct дефиниции за Company, Account, Counterpart, JournalEntry, Invoice с Serde serialization.

8.4. Python: Data Wrangling и анализ

Класове SaftDataCleaner, CounterpartImporter, AccountMapper за почистване и трансформация на данни.

8.5. SQL: Релационен подход

Пълен PostgreSQL schema с tables, indexes, triggers, materialized views и stored procedures.

8.6. Data Quality Framework

Rust валидационен engine със SaftValidator и comprehensive field validation.

8.7. ETL Pipeline с Python

Async pipeline с Extractor, Transformer, Loader абстракции.

Глава 9: SQL vs NoSQL — Избор на база данни за SAF-T

> *Пълното съдържание на тази глава е в CHAPTER_9.md*

9.1. Дилемата на архитектора

Критерий	SQL (PostgreSQL)	NoSQL (SurrealDB)
Схема	Строга	Гъвкава
Транзакции	ACID	Различни нива
Скалируемост	Вертикална	Хоризонтална

9.2. PostgreSQL: Релационният стандарт

Пълен schema с ENUMs, functions, triggers, materialized views.

9.3. SurrealDB: Документен + Графов подход

DEFINE TABLE с SCHEMAFULL, Graph relations за автоматични кореспонденции.

9.4. Хибриден подход

Rust implementation за синхронизация между PostgreSQL и SurrealDB.

9.5. Performance Optimization

Indexing strategy, VACUUM ANALYZE, query optimization за големи обеми данни.

Глава 10: Тестване и QA на SAF-T файлове

> *Пълното съдържание на тази глава е в CHAPTER_10.md*

10.1. Защо тестването е критично

Пет слоя валидация: 1. XSD Schema Validation 2. Data Type Validation 3. Referential Integrity 4. Cross-module Consistency 5. Business Logic

10.2. XSD Валидация (Layer 1)

Rust и Python XSD validators с lxml.

10.3. Data Type Validation (Layer 2)

SaftFieldTypeValidator за EIK, VAT, UOM, decimals, dates.

10.4. Referential Integrity (Layer 3)

SaftIntegrityValidator за cross-reference проверки.

10.5. Cross-module Consistency (Layer 4)

CrossModuleValidator за invoice-journal links, VAT reconciliation.

10.6. Business Logic Validation (Layer 5)

AccountingRulesValidator за double-entry principle, trial balance.

10.7. Automated Test Suite

Pytest framework с fixtures за всички типове тестове.

10.8. CI/CD Integration

GitHub Actions workflow за автоматична валидация.

Глава 11: Реални казуси от практиката

> *Пълното съдържание на тази глава е в CHAPTER_11.md*

11.1. Казус: Търговец на едро с 3 склада

Предизвикателства: Дублирани продукти, различни мерни единици **Решение:** ProductDeduplicator + WarehouseConsolidator в Rust

11.2. Казус: Производствено предприятие с ВОП

Предизвикателства: Вътрешнообщностни доставки, сложни ДДС схеми **Решение:** VatClassification engine с VIES integration

11.3. Казус: Холдинг с множество дружества

Предизвикателства: Междуфирмени транзакции, консолидация **Решение:** ConsolidationEngine с elimination entries

11.4. Казус: Едноличен търговец

Предизвикателства: Ограничени ресурси, опростена структура **Решение:** SimplifiedSaftGenerator за бърз старт

11.6. Данъчни рискови индикатори: Какво вижда НАП, когато отвори файла

SAF-T е рентген на счетоводната политика — всяко решение (полезен живот на актив, обезценка на вземане, отписване на запас) е машинночетимо и сравнимо. Секцията покрива:

- **Активи и амортизации** — сравнение счетоводен vs данъчен полезен живот (ЗКПО чл. 55), напълно амортизирани активи с приходи
- **Несъбираеми вземания и „вечни" салда** — анализ по сметки 41х/40х, салда без движение > 3 години, контрагенти без ЕИК
- **Запаси: отписвания и обезценки** — залежали артикули, декемврийски чистки, отрицателни количества, разлика склад/книга
- **Разходи със смесено ползване** — горива, наеми, телеком без аналитичност лично/служебно (ЗКПО чл. 204)
- **Обобщен Risk Score** — композитен профил LOW/MEDIUM/HIGH/CRITICAL с Python скриптове за self-assessment

11.7. Голямата стратегическа грешка: Каруцата пред коня

Анализ на липсващото звено — електронното фактуриране — и защо SAF-T без централизиран хъб за обмен на структурирани данни е обречен на „дигитален боклук".

11.8. Заключение

Няма едно решение за всички. SAF-T имплементацията трябва да се адаптира към спецификите на бизнеса.

Приложения

> *Пълното съдържание е в APPENDICES.md*

Приложение А: Пълна таблица на SAF-T сметките (360 кода)

Приложение Б: Данъчни кодове (TaxTable)

Приложение В: Мерни единици (UOM)

Приложение Г: Типове движения за склад (On Demand)

Приложение Д: Префикси за SAF-T ID

Приложение Е: Примерен SAF-T XML (Monthly)

Приложение Ж: Rust пълен генератор (boilerplate)

Приложение З: Често задавани въпроси (FAQ)

Заключение: Голямата стратегическа грешка

SAF-T не е просто край на класическото счетоводство. Той е начало на **Data-Driven Accounting**, но в българския контекст той е и огледало на административната недалновидност.

Основният извод е ясен: НАП сложи каруцата пред коня.

Опитът да се въведе SAF-T (детайлен одит на данните) без фундамента на задължителното електронно фактуриране е обречен на „дигитален хаос“. Докато фактурите не станат структурирани данни в момента на тяхното издаване (през централизиран хъб), SAF-T ще остане една невъзможна фантазия, захранвана от OCR грешки и ръчно робство.

Въпреки това, счетоводителят на бъдещето трябва да бъде: - **Data Architect** — проектира модела на данните - **Quality Engineer** — гарантира целостта на информацията - **Automation Specialist** — използва инструменти вместо ръчен труд

Тази книга ви даде техническите и архитектурни оръжия, за да не бъдете погълнати от този хаос. **Следващата стъпка е вашата.** Подредете данните си при източника, преди системата да ги е поискала при отчета.

Книга за SAF-T BG v1.0.1 От екипа на baraba.org 2026