**1. Общая идея и постановка задачи**

В рамках учебного проекта нужно добавить в существующую систему компонент, который **отвечает за хранение антивирусных сигнатур** и даёт возможность **получать** их:

1. Целиком (полная база).
2. Выборочно (только изменённые или добавленные с определённой даты).

В компонентесистемы должна быть реализована реальная (или максимально приближённая к реальной) схема **электронной подписи** каждого записываемого в базу объекта, чтобы можно было проверять целостность и отсутствие несанкционированных изменений.

Управлять логикой добавления и изменения записей должен пользователь с ролью «администратор». Пользователи с другими ролями могут только получать записи.

**2. Структура таблицы сигнатур**

В БД должна быть создана таблица для хранения сигнатур. Ниже приводится описание.

| **Поле** | **Тип данных (пример)** | **Описание** |
| --- | --- | --- |
| id | GUID (PK) | Уникальный идентификатор записи |
| threat\_name | VARCHAR / TEXT | Название угрозы |
| first\_bytes | BINARY(8) / VARCHAR | Первые 8 байт сигнатуры |
| remainder\_hash | VARCHAR | Хэш от «хвоста» |
| remainder\_length | INT | Количество байт в «хвосте» |
| file\_type | VARCHAR | Тип файла, для которого актуальна сигнатура |
| offset\_start | INT | Смещение начала сигнатуры в файле |
| offset\_end | INT | Смещение конца сигнатуры в файле |
| digital\_signature | VARCHAR / BLOB | Электронная цифровая подпись |
| updated\_at | DATETIME / TIMESTAMP | Дата и время последнего обновления (исп. для механизма «диффа») |
| status | VARCHAR | Статус записи: ACTUAL, DELETED, CORRUPTED |

**3. Описание и требования к компоненту, обрабатывающему сигнатуры и ЭЦП**

Предполагается, что существует **единый сервис**, отвечающий за:

1. **Приём** сигнатуры (добавление в базу).
2. **Обновление** существующей сигнатуры.
3. **Формирование** электронной подписи (ЭЦП) при любом создании или изменении записи.
4. **Проверку** корректности ЭЦП при необходимости (например, при выдаче или в ходе периодических проверок).
5. **Выдачу** списка сигнатур либо полной базы, либо только диффа по дате.

Когда сервис получает запрос на создание/обновление сигнатуры, он:

* + Формирует «подписываемые данные»;
  + Вычисляет их хэш;
  + Подписывает полученный хэш приватным ключом;
  + Полученную подпись сохраняет в колонку digital\_signature.

При получении или использовании сигнатуры можно проверить подпись, расшифровав (проверив) её публичным ключом и сравнив с вычисленным хэшем тех же полей.

**4. Эндпоинты (REST API) для выгрузки данных**

**4.1. Получение всей базы**

Реализовать эндпоинт для получения полного списка сигнатур, данные должны быть возвращены в формате JSON. При реализации учесть возможность смены формата возвращаемых данных таким образом, чтобы изменения в основной логике были минимальны. При получении всей базы, должны возвращаться только актуальные записи.

**4.2. Получение «диффа»**

У каждой записи есть поле updated\_at. При создании или обновлении записи сервис проставляет текущее время в updated\_at. Клиентский запрос может содержать параметр since (дата/время). Сервис возвращает только те записи, у которых updated\_at **позднее**, чем since. В данном случае возвращаются все записи с указанной даты из таблицы сигнатур.

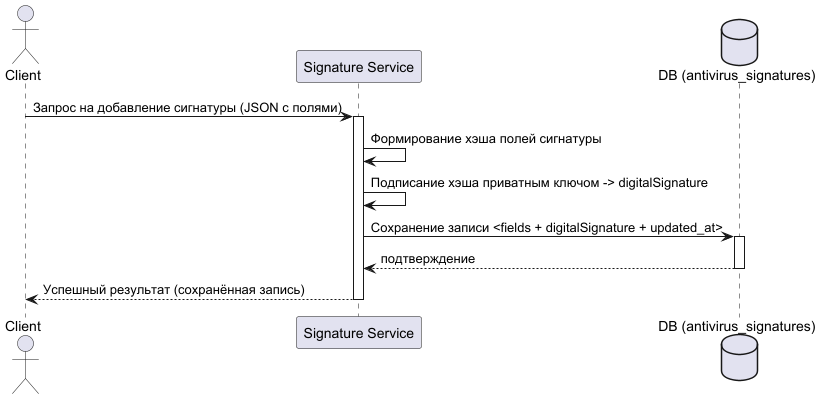
**4.3. Получение записей по списку GUID**

Предоставить возможность для клиента получения сигнатур по списку GUID. На сервер будет приходить запрос, в теле которого будет находиться список GUID. В ответ должен быть возвращен список сигнатур в формате JSON.

**4.4. Добавление новой сигнатуры**

Примерная схема обработки запроса при добавлении записи:

1. Клиент отправляет запрос (например, JSON с данными о сигнатуре).
2. Сервис вычисляет хэш, подписывает, получает ЭЦП.
3. Сервис сохраняет всё в БД (новая запись).
4. Сервис возвращает клиенту успешный результат.

****

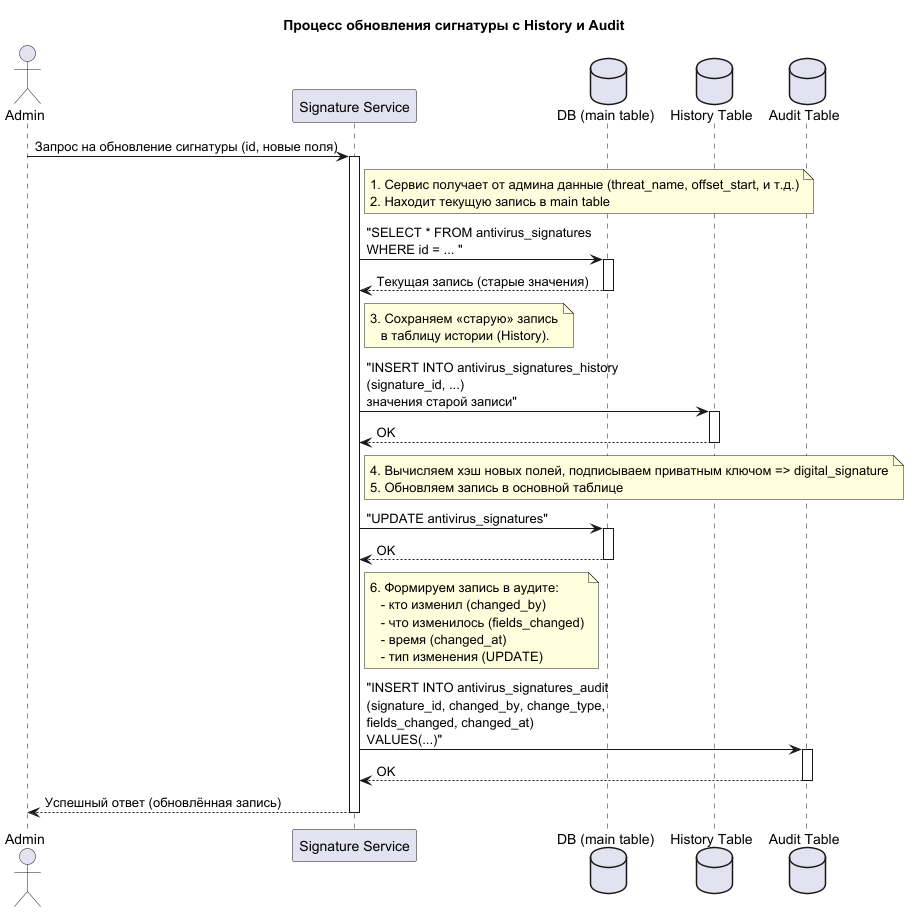
**4.5. Удаление записи**

При удалении сигнатуры из базы данных фактически запись не удаляется, ей устанавливается статус **DELETED**. Такие записи не возвращаются клиенту при запросе всех записей, но возвращаются при запросе «диффа», чтобы клиент предпринял действия на своей стороне.

**4.6. Обновление записи**

При обновлении записи, текущее значение не должно теряться, оно будет записываться в специальную таблицу с историей изменений. Т.е. при обновлении записи администратором, текущая запись записывается в историю, а затем обновляется в таблице сигнатур, при этом в таблице аудита фиксируется, кто и какие поля изменял.

Ниже приведена упрощенная схема процесса обновления записи сигнатуры.



**4.7. Просмотр записей по статусам и получение аудита**

Для проведения анализа ошибок и инцидентов необходимо предоставить администратору доступ к записям по статусам (условно добавляется в панель администратора).

**5. Управление версиями и аудит изменений**

Когда мы говорим о версионировании записей в БД, мы хотим хранить предыдущие состояния записи, чтобы можно было отслеживать, как менялась сигнатура со временем, откатываться при ошибочном/несанкционированном изменении или анализировать, кто и когда менял запись.

При этом основная таблица сигнатур хранит только актуальную версию каждой сигнатуры, а все предыдущие версии сохраняются в «исторической» таблице.

**Структура таблицы History**

| **Поле** | **Тип данных (пример)** | **Описание** |
| --- | --- | --- |
| history\_id | BIGINT (PK) | Уникальный идентификатор **записи в истории** |
| signature\_id | GUID | Ссылка на запись в таблице сигнатур (тот же id, что и в основной таблице) |
| version\_created\_at | DATETIME / TIMESTAMP | Момент времени, когда появилась эта версия |
| threat\_name | VARCHAR / TEXT | Название угрозы (копия поля на момент сохранения в истории) |
| first\_bytes | BINARY(8) / VARCHAR | Первые 8 байт (копия поля) |
| remainder\_hash | VARCHAR | Хэш «хвоста» (копия поля) |
| remainder\_length | INT | Длина «хвоста» (копия поля) |
| file\_type | VARCHAR | Тип файла (копия поля) |
| offset\_start | INT | Смещение начала сигнатуры (копия поля) |
| offset\_end | INT | Смещение конца сигнатуры (копия поля) |
| digital\_signature | VARCHAR / BLOB | Копия ЭЦП |
| status | VARCHAR | Копия статуса записи на момент сохранения версии |
| updated\_at | DATETIME / TIMESTAMP | Копия значения updated\_at |

**Структура таблицы Audit**

| **Поле** | **Тип данных (пример)** | **Описание** |
| --- | --- | --- |
| audit\_id | BIGINT (PK) | Уникальный идентификатор записи в аудите |
| signature\_id | GUID | Ссылка на запись в таблице сигнатур |
| changed\_by | VARCHAR / GUID / INT | Указатель на пользователя, совершившего изменение |
| change\_type | VARCHAR | Тип изменения (CREATED, UPDATED, DELETED, CORRUPTED и т.д.) |
| changed\_at | DATETIME / TIMESTAMP | Время, когда произошло изменение |
| fields\_changed | TEXT | Список изменённых полей, можно хранить в виде JSON |

**Взаимодействие статуса (ACTUAL, DELETED, CORRUPTED) с History и Audit**

1. **DELETED**. Когда администратор удаляет запись, на самом деле мы делаем устанавливаем status = 'DELETED'. В history уходит «старое» состояние, которое было до этого. В audit записываем change\_type = DELETE.
2. **CORRUPTED**. Если при сверке ЭЦП мы обнаружили проблему, мы присваиваем status = CORRUPTED. Создаём версию в history. В audit пишем, что система присвоила CORRUPTED из-за сбоя подписи.
3. **ACTUAL**. Обычное состояние записи. Любые изменения также фиксируются, как описано выше.

Таким образом, **каждое изменение статуса** (равно как и изменение других полей) ведёт к созданию версии в history и события в audit.

**6. Регулярная сверка ЭЦП**

Необходимо создать периодическую задачу (Scheduled Task), с заданной в константе частотой выполнения (по умолчанию раз в сутки). Задача должна вызывать сервис, отвечающий за проверку ЭЦП.

Регулярная проверка должна проводиться только для тех записей, которые были изменены за время, прошедшее с предыдущей проверки.

Реализовать запуск проверки рекомендуется при помощи инструментов Spring Boot, а конкретно - Spring Scheduler.

Если во время проверки было обнаружено несоответствие ЭЦП, то необходимо залогировать данное событие со статусом ERROR, запись сигнатуры при этом должна быть помечена, как повреждённая.

Ниже приведена упрощенная схема процесса работы задачи по расписанию.

