ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ДИЗАЙН-ТОКЕНОВ

Лазебная Е.А., старший преподаватель,
Присухин Б.Р., студент,
БГТУ им. В.Г. Шухова, г. Белгород, Россия

Аннотация: В статье рассматривается проектирование информационной системы для централизованного управления дизайн-токенами. Представлены функциональные модели IDEF0, описывающие входы, выходы, управление и механизмы системы. Также спроектирована логическая модель данных в нотации IDEF1X, включающая сущности коллекций, токенов, истории изменений, версий коллекций и пользователей. Обоснован выбор реляционной СУБД MySQL.

Ключевые слова: дизайн-токены, проектирование ис, IDEF0, IDEF1X, модель данных, версионирование, реляционная БД.

Концепция дизайн-токенов возникла как закономерный этап эволюции взаимодействия между дизайном и разработкой. С ростом сложности цифровых продуктов и появлением атомарного дизайна компании столкнулись с проблемой масштабирования визуальных языков. К 2018 году эта проблема достигла критической массы — крупнейшие игроки (GitHub, Adobe, Shopify) совместно с сообществом W3C инициировали стандартизацию форматов обмена дизайн-токенами. Согласно спецификации W3C Design Tokens Community Group, дизайн-токены — это «наименьшие неделимые единицы дизайн-системы, такие как цвета, отступы, масштабы типографики» [1]. Впервые концепция была предложена командой Salesforce для выражения дизайнерских решений платформонезависимым способом, что позволяет использовать их в различных инструментах, технологиях и платформах.

При проектировании моделей предметной области целесообразно

использовать смешанный подход, который позволит описать функции системы с разных сторон. Например, методология IDEF0 позволяет описать все бизнес-процессы, присутствующие в системе [2]. На рисунке представлена контекстная диаграмма IDEF0, которая дает общее представление о деятельности веб-ориентированной информационной системы управления жизненным циклом дизайн-токенов (Рис. 1):

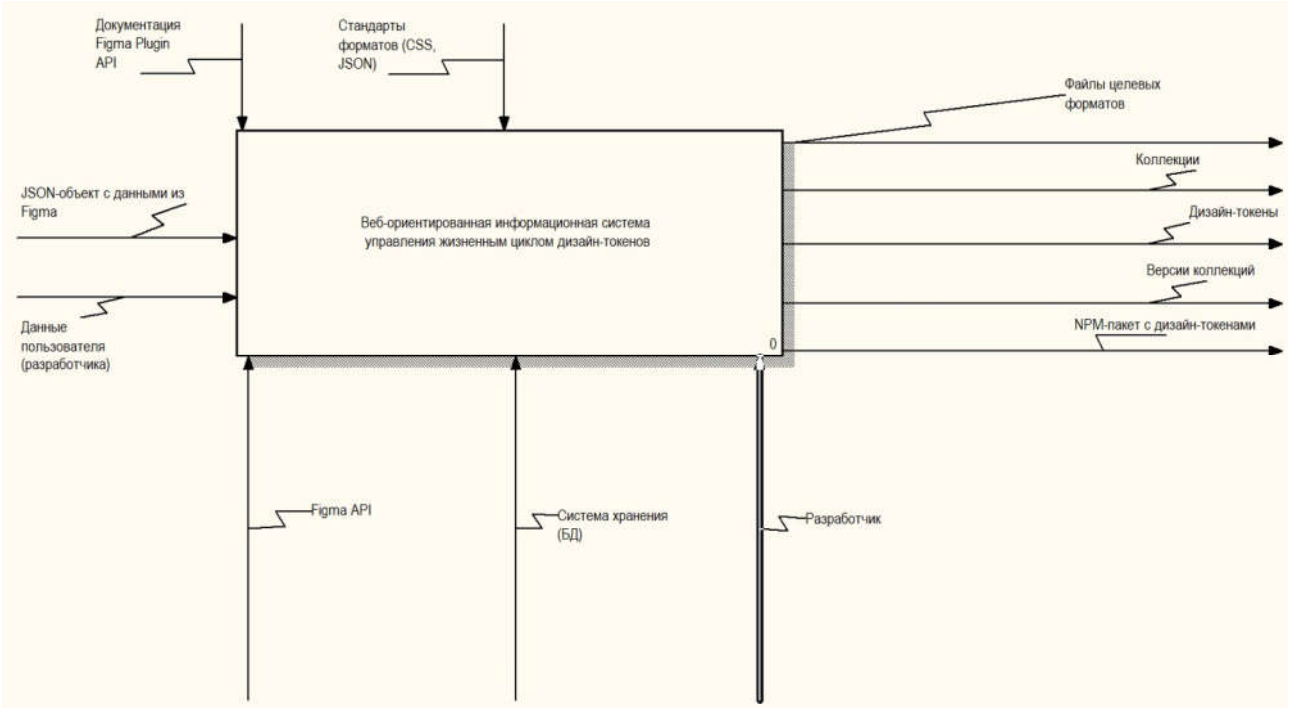

Рис. 1 Контекстная диаграмма информационной системы

Входными данными являются JSON-объект с данными из Figma, содержащий коллекции переменных, их значения, типы и метаданные, полученные через плагин, а также данные пользователя (разработчика) — имя и email, которые используются для подписи создаваемых версий и фиксации автора в истории изменений. Управляющее воздействие на систему оказывают документация Figma Plugin API [3], регламентирующая способы интеграции плагина с Figma, и стандарты форматов CSS и JSON, определяющие требования к структуре генерируемых файлов. В роли механизмов выступают Figma API, обеспечивающий доступ к переменным и коллекциям, реляционная база данных, отвечающая за сохранность токенов, коллекций, версий и истории изменений, а также разработчик как конечный пользователь системы,

взаимодействующий с веб-интерфейсом. На выходе система формирует файлы целевых форматов (CSS, JSON) для фронтенд-разработки, NPM-пакет с дизайн-токенами для интеграции в JavaScript-проекты, а также структурированные данные (токены, коллекции и версии коллекций), доступные через веб-интерфейс для просмотра, сравнения и управления.

Далее, с целью детального уточнения, диаграмма декомпозируется. Результат декомпозиции представлен на рис. 2.

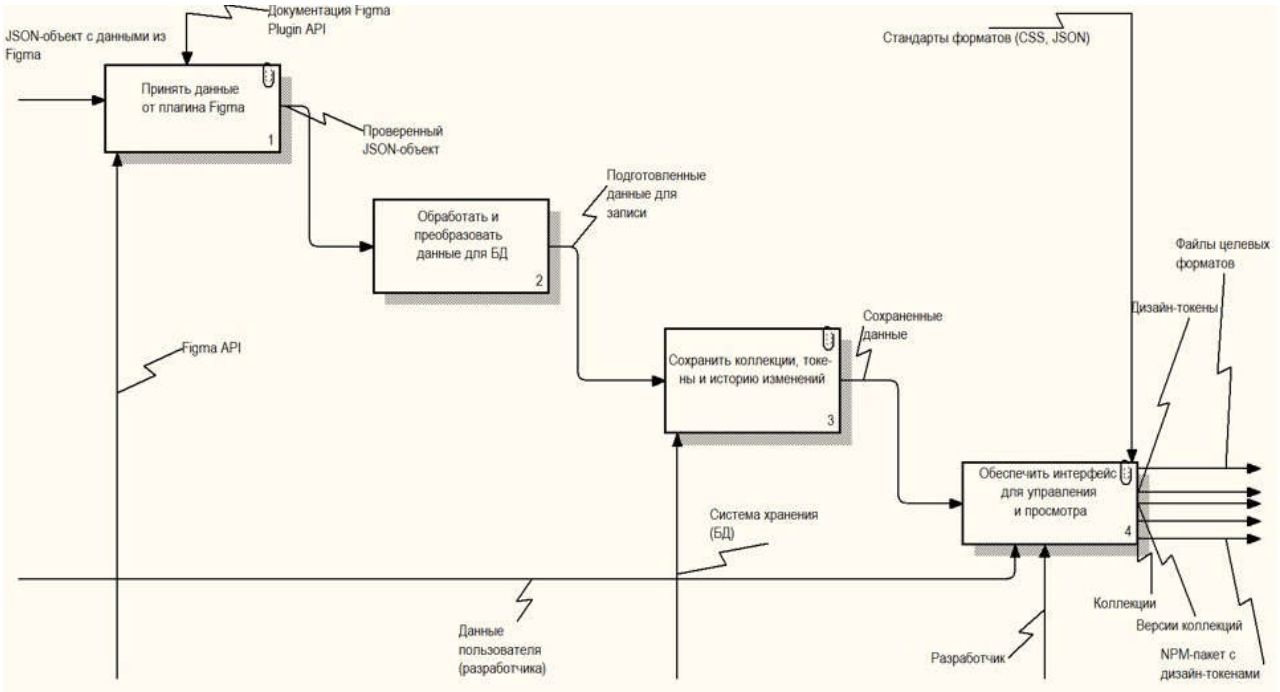

Рис. 2 Диаграмма декомпозиции информационной системы

Процесс управления жизненным циклом дизайн-токенов состоит из пяти основных блоков:

- принять данные от плагина Figma;
- обработать и преобразовать данные для БД;
- сохранить коллекции, токены и историю изменений;
- обеспечить интерфейс для управления и просмотра;
- сгенерировать и предоставить файлы целевых форматов.

На вход блока «Принять данные от плагина Figma» поступает JSON-объект с данными из Figma, содержащий коллекции переменных, их значения и

метаданные. Процесс приёма данных должен соответствовать документации Figma Plugin API, которая регламентирует способы взаимодействия с плагином.

На выходе из этого блока получаем проверенный JSON-объект, который будет передан в блок обработки и преобразования.

В ходе блока «Обработать и преобразовать данные для БД» выполняется валидация структуры данных, приведение типов (например, преобразование RGB-значений в hex-формат) и подготовка данных для записи в базу данных. Результатом являются подготовленные данные для записи, которые поступают в блок сохранения.

Блок «Сохранить коллекции, токены и историю изменений» отвечает за запись данных в систему хранения (БД). Здесь фиксируются актуальные значения токенов, создаются или обновляются коллекции, а также ведётся история изменений каждого токена. Выходом из этого блока являются сохранённые данные, которые могут быть использованы в дальнейшем для просмотра и экспорта.

Имея сохранённые коллекции, токены и их версии, система переходит к блоку «Обеспечить интерфейс для управления и просмотра». Здесь, с учётом данных пользователя (разработчика), через веб-интерфейс предоставляется доступ к просмотру коллекций, версий коллекций и истории изменений. Разработчик может управлять версиями, просматривать различия между ними и инициировать экспорт файлов для использования в разработке.

На рис. 3 представлена диаграмма декомпозиции процесса приёма данных от плагина Figma. Данный процесс состоит из трёх последовательных этапов.

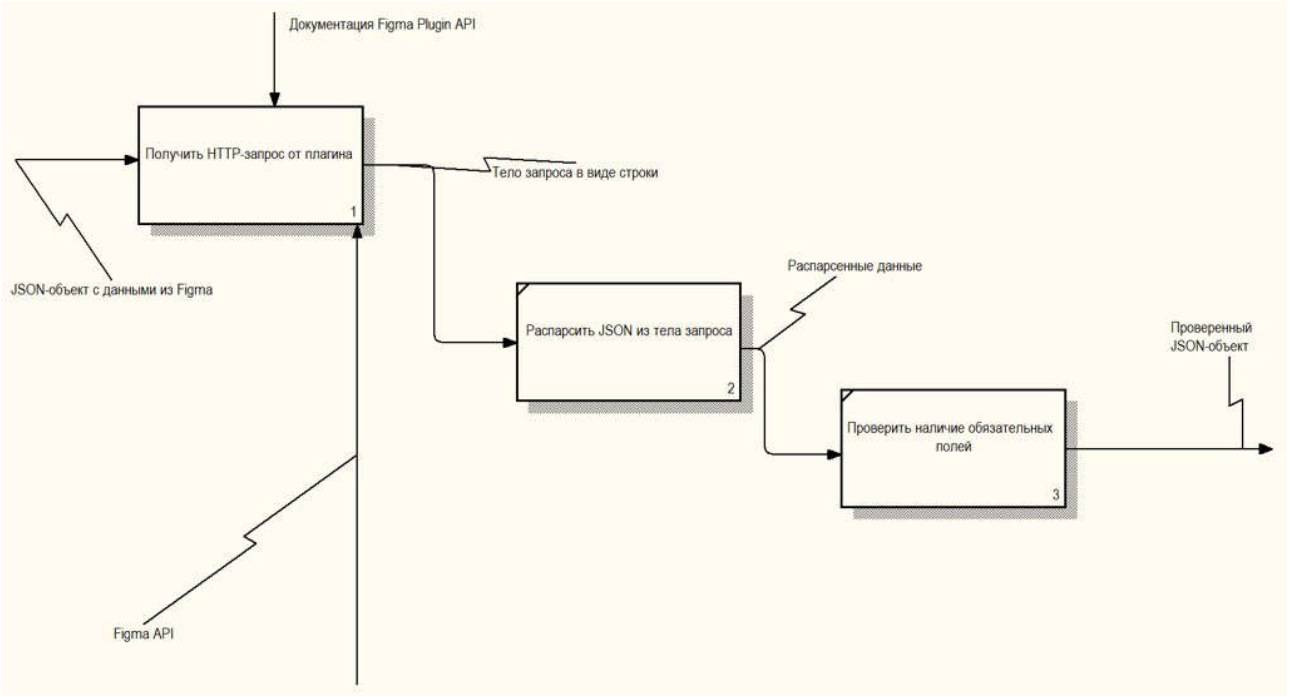

Рис. 3 Декомпозиция блока «Принять данные от плагина Figma»

Блок 1 «Получить HTTP-запрос от плагина» обеспечивает приём входящего запроса от клиентского приложения. На вход блока поступает JSON-объект с данными из Figma, содержащий информацию о коллекциях переменных, их значениях и метаданных. Также блок использует документацию Figma Plugin API как управляющее воздействие, регламентирующее формат и структуру передаваемых данных. В результате работы блока формируется тело запроса в виде строки, которое передаётся на следующий этап обработки.

Блок 2 «Распарсить JSON из тела запроса» выполняет преобразование строкового представления JSON в структурированные данные, пригодные для программной обработки. На выходе блока формируются распарсенные данные, представляющие собой объектную структуру с информацией о дизайн-токенах.

Блок 3 «Проверить наличие обязательных полей» осуществляет валидацию полученных данных. Блок проверяет присутствие всех необходимых атрибутов в JSON-объекте. Результатом работы блока является проверенный JSON-объект, который может быть передан на дальнейшую обработку в систему.

На рис. 4 представлена диаграмма декомпозиции процесса обработки и преобразования данных для последующей записи в базу данных. Данный процесс состоит из двух последовательных этапов, обеспечивающих трансформацию входящих данных из Figma в формат, пригодный для хранения в реляционной СУБД.

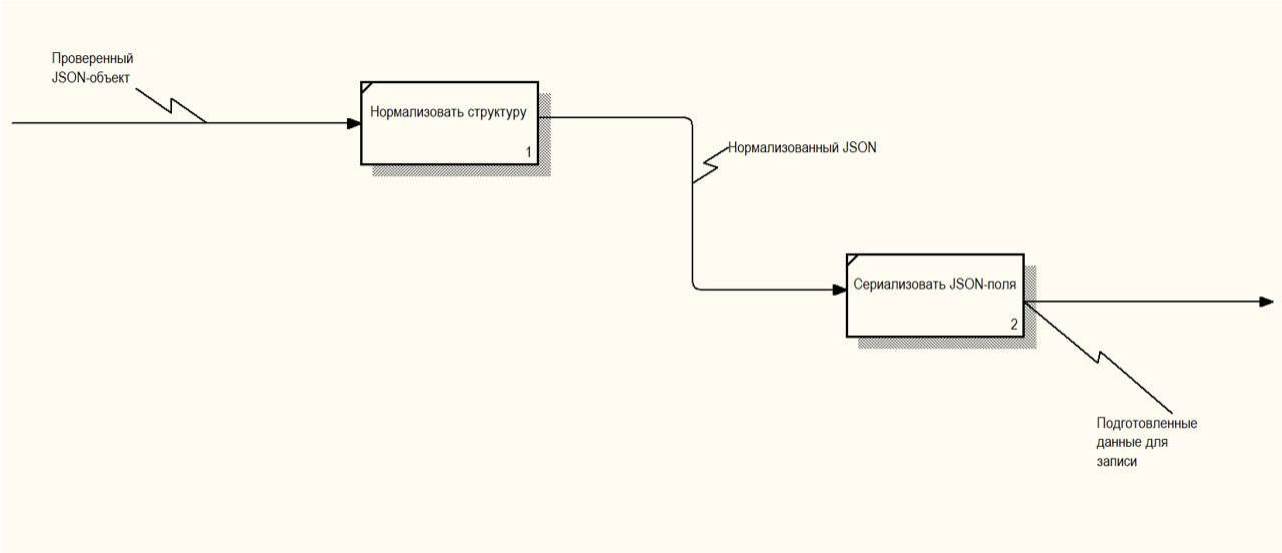

Рис. 4 Декомпозиция блока «Обработать и преобразовать данные для БД»

Блок 1 «Нормализовать структуру» выполняет первичную обработку проверенного JSON-объекта, поступившего от плагина Figma. На этом этапе производится преобразование сырых данных Figma API во внутреннюю структуру системы, которая включает:

- группировку переменных по коллекциям — переменные (variables) распределяются по соответствующим коллекциям на основе поля collectionId;
- извлечение метаданных — для каждой коллекции извлекаются идентификатор, название и список режимов (modes);
- стандартизацию типов данных — переменные приводятся к единому формату с сохранением их типа (COLOR, FLOAT, STRING, BOOLEAN), имени, идентификатора и значений по режимам (valuesByMode).

Результатом работы блока является нормализованный JSON-объект, содержащий две основные сущности: массив коллекций с их метаданными и массив переменных, сгруппированных по коллекциям.

Блок 2 «Сериализовать JSON-поля» выполняет финальную подготовку данных для совместимости с реляционной базой данных MySQL. Поскольку сложные структуры данных (например, значения переменных по режимам) не могут быть напрямую сохранены в отдельные колонки таблицы, выполняется их сериализация, которая включает:

- преобразование valuesByMode — объект, содержащий значения переменной для каждого режима Figma, преобразуется в строку формата JSON с помощью JSON.stringify();
- подготовку к записи — все поля приводятся к типам данных, совместимым с СУБД (строки, числа, логические значения);
- сохранение структуры — несмотря на сериализацию, полная структура данных сохраняется и может быть восстановлена при чтении из БД.

На выходе процесса формируются окончательно подготовленные данные для записи, которые включают:

- массив коллекций с полями: id, name, modes (сериализованный JSON);
- массив переменных с полями: id, name, type, valuesByMode (сериализованный JSON), collectionId.

Эти данные передаются в следующий блок «Сохранить коллекции, токены и историю изменений», где будет выполнена логика сравнения с существующими записями, определение изменений (новые/обновленные/удаленные переменные) и запись в базу данных в рамках единой транзакции.

На рис. 5 представлена диаграмма декомпозиции процесса сохранения данных в базе данных. Данный процесс обеспечивает атомарную запись всех изменений в рамках единой транзакции и состоит из пяти последовательных этапов.

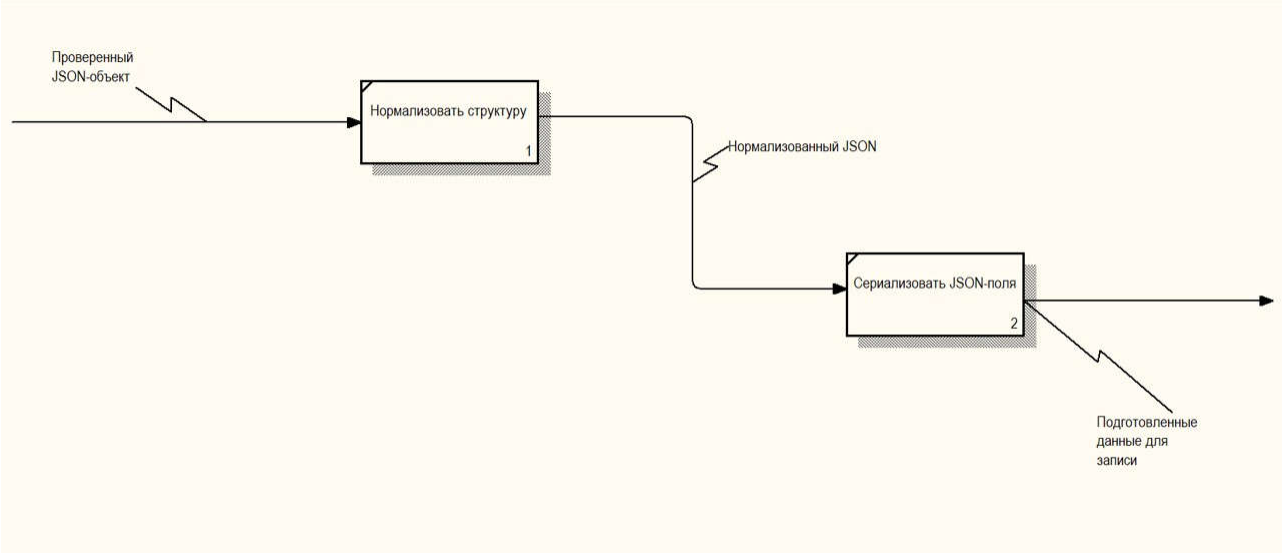

Рис. 5 Декомпозиция блока «Сохранить коллекции, токены и историю изменений»

Блок 1 «Открыть соединение с БД и начать транзакцию» инициализирует процесс сохранения. На вход блока поступают подготовленные данные для записи, прошедшие предварительную обработку и валидацию. Блок устанавливает соединение с системой хранения (базой данных MySQL) и инициирует новую транзакцию. Это критически важный этап, обеспечивающий целостность данных: либо все изменения будут сохранены, либо ни одно из них не будет применено в случае возникновения ошибки. На выходе блока формируются активное соединение с базой данных и подготовленные данные для записи, которые передаются на следующий этап.

Блок 2 «Сравнить новые данные с существующими записями БД» выполняет ключевую логику синхронизации. Для каждой коллекции из входящих данных блок загружает все активные (не удалённые) переменные из таблицы variables с фильтром по collection_id и флагом is_deleted = FALSE. На основе загруженных данных формируется карта существующих переменных (existingVarMap). Затем выполняется сравнение идентификаторов новых и существующих переменных, в результате чего вычисляются три подмножества:

- новые переменные — отсутствуют в БД (определяются по отсутствию id в существующих записях);

- изменённые переменные — присутствуют в БД, но изменились значения или метаданные (определяются с помощью функции hasVariableChanged, которая сравнивает значения по режимам с учётом типов данных: цвета, алиасы, числа, строки);
- удалённые переменные — присутствуют в БД, но отсутствуют во входящем пакете (определяются как разность множеств идентификаторов).

Блок 3 «Выполнить запись коллекций» осуществляет синхронизацию метаданных коллекций. Для каждой коллекции из входящих данных выполняется операция UPSERT (вставка или обновление) в таблице collections с использованием SQL-конструкции INSERT ... ON DUPLICATE KEY UPDATE. Если коллекция с указанным идентификатором уже существует, обновляются её метаданные: название и список режимов (модов), представленных в виде JSON-массива. Если коллекция новая — она создаётся с фиксацией временной метки создания. Результатом работы блока являются обновлённые данные о коллекциях, которые сохраняются в базе данных.

Блок 4 «Выполнить запись токенов и истории изменений» реализует основную логику синхронизации переменных. Для каждой коллекции блок выполняет последовательную обработку трёх категорий токенов:

- новые токены: для каждого токена, отсутствующего в базе данных, выполняется вставка записи в таблицу variables с указанием идентификатора, имени, типа, значений по режимам (в формате JSON) и ссылки на коллекцию. Параллельно в таблицу variable_history добавляется запись типа created с полным снимком значений токена;
- изменённые токены: для токенов, существующих в базе, но имеющих изменённые значения или метаданные, выполняется обновление записей в таблице variables. В историю изменений добавляется запись типа updated с новым снимком значений;
- удалённые токены: для токенов, отсутствующих во входящих данных, но существующих в базе, устанавливается флаг is_deleted = TRUE и

фиксируется дата удаления. В историю добавляется запись типа deleted.

Блок 5 «Завершить транзакцию и сформировать ответ» финализирует процесс сохранения. Если все предыдущие этапы выполнены успешно, блок фиксирует транзакцию командой commit(), делая все изменения постоянными. В случае возникновения любой ошибки на предыдущих этапах выполняется откат транзакции командой rollback(), возвращающий базу данных в исходное состояние. После успешной фиксации блок формирует ответ для клиента, содержащий статистику выполненных операций (количество созданных, обновлённых и удалённых токенов) и подтверждение успешного завершения. На выходе процесса формируются сохранённые данные, доступные для последующего использования через веб-интерфейс и API.

На рис. 6 представлена диаграмма декомпозиции процесса предоставления пользовательского интерфейса для работы с дизайн-токенами. Данный процесс состоит из шести взаимосвязанных этапов, обеспечивающих полный цикл взаимодействия пользователя с системой.

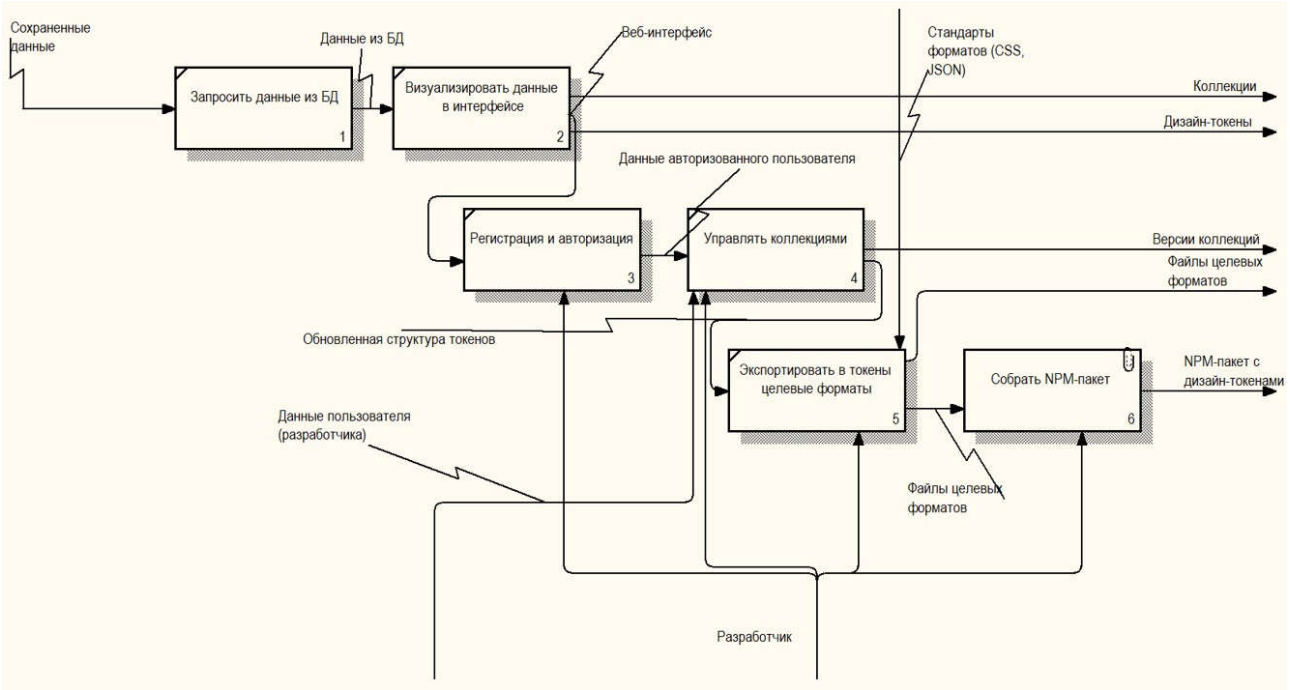

Рис. 6 Декомпозиция блока «Обеспечить интерфейс для управления и просмотра»

Блок 1 «Запросить данные из БД» инициирует процесс получения информации для отображения в интерфейсе. На вход блока поступают сохранённые данные из системы хранения. Блок формирует SQL-запросы к базе данных для извлечения актуальной информации о коллекциях, токенах, версиях и истории изменений. На выходе блока формируются данные из БД, которые передаются на визуализацию.

Блок 2 «Визуализировать данные в интерфейсе» отвечает за отображение полученной информации в веб-интерфейсе. Блок преобразует сырые данные из базы данных в удобную для восприятия форму: таблицы с токенами, иерархические деревья групп, временные шкалы истории изменений. Визуализация учитывает тип токена: цветовые токены отображаются с цветовыми превью, числовые — с форматированными значениями. На выходе блока формируются коллекции и дизайн-токены, представленные в веб-интерфейсе.

Блок 3 «Регистрация и авторизация» обеспечивает управление доступом к системе. Блок обрабатывает данные пользователя (разработчика), включая email, пароль и имя пользователя. При регистрации выполняется хеширование пароля и сохранение учётной записи в базе данных. При авторизации проверяются учётные данные и генерируется JWT-токен для последующей аутентификации запросов. Блок обеспечивает разграничение прав доступа: только авторизованные пользователи могут создавать версии коллекций и инициировать экспорт. На выходе блока формируются данные авторизованного пользователя, которые передаются в блок управления коллекциями.

Блок 4 «Управлять коллекциями» предоставляет функционал для работы с коллекциями дизайн-токенов. Блок позволяет: просматривать список всех коллекций с метаданными (название, количество токенов, дата создания); переключаться между коллекциями для детального просмотра; создавать новые версии коллекций с присвоением имени, тега и описания; просматривать список существующих версий с возможностью переключения между ними. При создании версии блок инициирует процесс фиксации текущего состояния всех

токенов коллекции. На выходе блока формируются версии коллекций и файлы целевых форматов.

Блок 5 «Экспортировать в токены целевые форматы» отвечает за преобразование внутреннего представления токенов в форматы, пригодные для использования в разработке. Блок поддерживает генерацию следующих форматов:

- CSS Custom Properties — переменные CSS для веб-разработки;
- JSON — файлы, совместимые со спецификацией W3C Design Tokens и инструментом Style Dictionary;
- SCSS — переменные Sass/SCSS для препроцессоров.

Блок учитывает типы токенов при экспорте: цветовые токены преобразуются в соответствующие форматы (hex, rgb), числовые — в единицы измерения (px, rem). На выходе блока формируются файлы целевых форматов, которые передаются на сборку NPM-пакета.

Блок 6 «Собрать NPM-пакет» выполняет финальную упаковку дизайн-токенов для распространения через npm registry. Блок создаёт структуру пакета с необходимыми файлами: package.json с метаданными, сгенерированными файлами токенов в директории dist/, документацией README.md. Пакет готов к публикации и может быть установлен в проекты разработки командой npm install. На выходе блока формируется NPM-пакет с дизайн-токенами, который передаётся разработчику для интеграции в проекты.

База данных основана на реляционной модели и содержит основные сведения о сущностях системы управления жизненным циклом дизайн-токенов (Рис. 7).

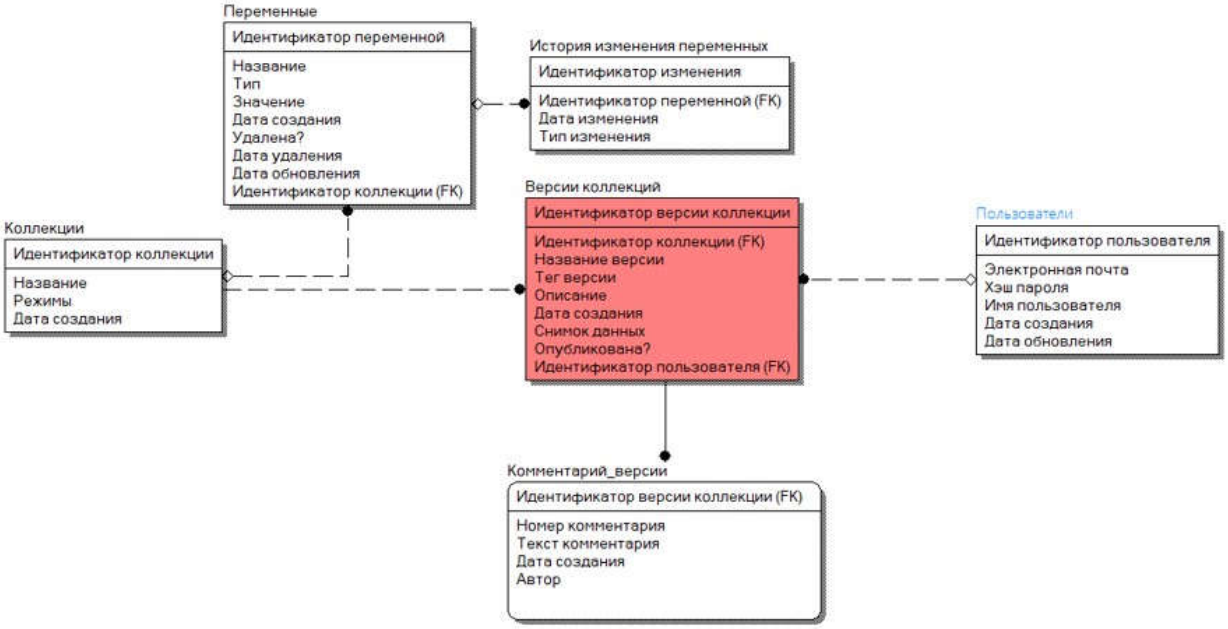

Рис. 7 Логическая модель данных информационной системы

В таблице «Коллекции» хранится информация о коллекциях дизайн-токенов, представляющих собой логические группировки, такие как примитивы или семантические токены. Для каждой коллекции фиксируется её название. Временная метка позволяет определить дату создания коллекции в системе.

Таблица «Токены» является центральной в системе и хранит информацию о каждом дизайн-токене. Для токена фиксируются его название, тип (цвет, число, строка или логическое значение), а также значения для различных режимов, что позволяет гибко поддерживать множественные состояния. Каждый токен обязательно принадлежит одной коллекции, что обеспечивает связь между таблицами: одна коллекция может содержать множество токенов, но каждый токен относится только к одной коллекции. Для поддержки мягкого удаления предусмотрены флаг удаления и дата удаления, а также автоматически обновляемая временная метка последнего изменения.

Для обеспечения аудита всех изменений, происходящих с токенами, предназначена таблица «История изменений токенов». Каждая запись в этой таблице соответствует одному изменению конкретного токена и содержит снимок его значений на момент изменения, временную метку и тип изменения

— создание, обновление, удаление или восстановление. Данная таблица связана с таблицей «Токены» таким образом, что одному токену может соответствовать множество записей истории, но каждая запись истории относится ровно к одному токену.

Таблица «Версии коллекций» предназначена для хранения снимков состояния коллекций на определенные моменты времени. Для каждой версии указывается, к какой коллекции она относится, её название, тег, описание, дата создания, а также полный снимок данных коллекции, содержащий все токены этой коллекции с их значениями на момент создания версии. Дополнительно фиксируется, опубликована ли версия для использования в разработке. Одна коллекция может иметь множество версий, но каждая версия относится ровно к одной коллекции. Также предусмотрена связь с пользователями, создавшими версию.

Таблица «Пользователи» содержит информацию для идентификации и авторизации пользователей системы. В ней хранятся электронная почта, хеш пароля, имя пользователя, а также временные метки создания и обновления записи. Электронная почта имеет уникальное значение для предотвращения дублирования учетных записей.

Построены функциональные модели IDEF0 и логическая модель данных IDEF1X, включающая таблицы коллекций, токенов, истории, версий и пользователей. Модели поддерживают версионирование и полную историю изменений. Выбрана реляционная СУБД MySQL. Результаты проектирования готовы к реализации.

Литература

1. Design Tokens Technical Reports 2025.10 [Электронный ресурс] / Design Tokens Community Group. – 2025. – Режим доступа: https://www.designtokens.org/tr/2025.10/ (дата обращения: 06.05.2026).

2. Figma. Plugin API Reference [Электронный ресурс] // Figma Developers. – 2026. – Режим доступа: https://developers.figma.com/docs/plugins/api-reference/ (дата обращения: 06.05.2026).
3. Р 50.1.028-2001. Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования [Электронный ресурс]. – Введ. 2002-07-01. – Режим доступа: https://docs.cntd.ru/document/1200028629 (дата обращения: 06.05.2026).