Документация, содержащая описание функциональных характеристик экземпляра программного обеспечения, предоставленного для проведения экспертной проверки

Программного обеспечения «Amplicode»

г. Самара

Глоссарий

Бэкенд	(англ. backend) — программная часть сервиса, отвечающая за функционирование его внутренней части.
Open source	Программное обеспечение с открытым исходным кодом. Исходный код таких программ доступен для просмотра, изучения и изменения.
Фреймворк	Программная платформа, определяющая структуру и архитектуру приложения. “Каркас” приложения.
API	(англ. application programming interface) — программный интерфейс приложения. Описание способов взаимодействия различных частей приложения друг с другом.
REST API	(англ. Representational state transfer) — API, которое соответствует набору ограничений архитектурного стиля REST.
Визард (мастер)	(англ. wizard) — приём построения пользовательских интерфейсов, при котором для совершения какого-то действия пользователю необходимо последовательно пройти несколько форм, содержащих небольшое количество элементов управления.
Endpoint	Точка подключения к веб-сервису, инструменту или приложению, доступному по сети.
R&D	(англ. Research and design) — набор исследовательских активностей, предпринимаемых при разработке новых или улучшению существующих услуг или продуктов.
DDL	(англ. Data definition language) — семейство языков программирования, используемых для описания структуры базы данных.
NoSQL	Широкий класс систем управления баз данных, в которых делается попытка решить проблемы масштабируемости и доступности за счет полного или частичного отказа от требований атомарности и согласованности данных.

1. Введение

Назначение Amplicode предоставить инструменты максимально эффективной и комфортной разработки сервисов и web-приложений на Spring Boot. Amplicode не заставляет разработчиков пользоваться проприетарными абстракциями и узкоспециализированными языками предметной области (DSL). Напротив, концепция Amplicode заключается в максимальном использовании наиболее популярных библиотек и подходов.

Центральным компонентом Amplicode - является плагин для среды разработки IntelliJ IDEA. Плагин предоставляет разработчикам инструменты для создания бэкенд приложения на Spring Boot и связанных технологий, а также предоставляет инструменты для работы с Docker и Docker Compose файлами.

1.1. Роль и цели инструмента Amplicode

Плагин Amplicode для IntelliJ IDEA является центральной частью платформы. Этот инструмент призван увеличить продуктивность как начинающих, так и опытных разработчиков за счет того, что им не придется искать способы решения тех или иных типовых каждодневных задач, а также за счет помощи в навигации по проекту с учетом специфики вышеуказанных фреймворков.

Цели, которые ставятся перед Amplicode:

1. Снижение порога входа в технологию.
2. Ускорение разработки за счет:
    a. использования широко известных библиотек и фреймворков;
    b. богатых возможностей генерации кода;
    c. визуальных инструментов, помогающих ориентироваться в структуре кода и приложения, а также снижающих порог входа в технологию;
    d. минимального набора соглашений (conventions);
3. Контроль за применением лучших практик.
4. Целостный взгляд на проект:
    a. подключенные библиотеки и их конфигурации;
    b. источники данных;
    c. доменная модель;
    d. ролевая модель безопасности;
    e. события;
    f. API (MVC, REST);

В Amplicode базовым фреймворком для построения приложений на бэкенд части является Spring Boot.

1.2. Основные идеи и концепции

Для достижения поставленных целей предлагается подход с использованием визуальных инструментов, генераторов и анализаторов кода. В рамках такого подхода можно будет создавать и конфигурировать компоненты приложения путем указания нужных значений в диалоговых окнах или таблицах свойств.

1.2.1 Визуальные инструменты

Визуальные инструменты Amplicode призваны облегчить работу с кодом, а не заменить её. Цель – не скрыть сложность разработки за графическим интерфейсом, а избавить разработчика от поиска нужных классов, значений параметров конфигурации или нужного API в документации. Визуальные инструменты не «перекрывают» код, а помогают в навигации, редактировании и генерации типовых элементов. Графический пользовательский интерфейс не мешает опытным

разработчикам, которые могут написать код быстрее, чем перетащить элемент из палитры мышкой, и помогает менее опытным пользователям, которые не всегда знают, как правильно написать защищенный ролевым доступом REST API или обработать событие из Kafka.

Amplicode включает в себя следующие визуальных инструменты:

- Навигатор по структуре проекта с наглядным представлением конфигурации модулей доступа к данным, системы безопасности, API интеграций и пр.
- Палитры. Контекстно-зависимые палитры предоставляют выбор функционала, доступного для добавления в текущий контекст (класс, метод).
- Инспекторы. Показывают, что можно изменять в текущем контексте. Например, для метода контроллера это может быть выбор метода HTTP и URL, а для кнопки в пользовательском интерфейсе - надпись и имя функции-обработчика нажатия.
- Визарды (мастеры). Пошаговые визуальные помощники, которые помогают сгенерировать необходимые конфигурационные свойства, заготовки кода и подключить нужные библиотеки для приложения.

1.2.2. Генераторы кода

Генераторы кода автоматизируют рутинную работу, предоставляют возможность легко создать нужный код-заготовку для компонентов, который послужит основой для написания «полноценного» кода приложения. При помощи генераторов можно быстро создать REST контроллер, конфигурацию для источника данных или конфигурацию для Kafka.

1.2.3. Анализаторы кода

При написании кода важно не просто решать поставленную задачу, но и делать это в соответствии с лучшими практиками. Для решения этой задачи используются контекстно-зависимые анализаторы кода, которые выполняют не просто анализ правильности синтаксиса, а производят оценку кода с точки зрения применения практик разработки. Если выявлена какая-то проблема, то для нее предлагаются варианты исправления. Такой код будет проще поддерживать и в нем будет меньше потенциальных ошибок.

Требования к проектируемым визуальным инструментам, генераторам и анализаторам кода определяются отдельно для каждой конкретной функциональности и представлены в разделах ниже.

2. Инструментарий для разработки бэкенд

Spring Boot, служащий фундаментом бэкенд-части приложения – самый популярный фреймворк для разработки бэкенд приложений на языке Java. Типовое приложение, написанное на Spring Boot, состоит из следующих основных частей:

- Слой доступа к данным
- Сервисы бизнес-логики
- Слой безопасности и контроля доступа
- Слой API

Также могут присутствовать:

- Обработчики событий из очередей сообщений
- Сервисы хранения и обработки логов приложения
- Сервисы мониторинга состояния приложения
- и т.д.

Слой бизнес-логики - уникальный код, который пишется под требования бизнеса. Системные задачи, такие как работа с базами данных, обработка HTTP запросов и т.д., решаются с использованием возможностей библиотек Spring Boot.

Подключение библиотеки — это не просто добавление зависимости в файл сборки. Зачастую, это еще и задание необходимых значений параметров в конфигурации приложения, создание классов конфигурации и подключение связанных библиотек.

В Amplicode эти неявные зависимости между библиотеками продуманы таким образом, чтобы разработчик тратил меньше усилий при добавлении того или иного компонента приложения. Кроме того, для каждой библиотеки характерен уникальный набор процессов создания функциональности, которые дополнительно автоматизированы: такие как набор практик, инспекций кода и т.д.

Что касается ежедневных задач разработчика по написанию кода, то среды разработки предоставляют функциональность для увеличения продуктивности разработки при помощи этого фреймворка. В IntelliJ IDEA Ultimate это навигация по компонентам приложения, инъекция бинов, генерация и выполнение HTTP запросов для REST API и др. В Eclipse существует плагин Spring Tools Suite, который показывает URL для MVC контроллеров приложения, показывает статистику работы приложения во время исполнения и т.д. Таким образом, существующие инструменты обеспечивают ускорение решения задач разработчика в области написания кода. Но такие задачи обычно выполняются в рамках какого-то более крупного процесса, для которого нужны дополнительные знания о внутренней структуре приложения и API фреймворка.

Например, для создания защищенного REST API Endpoint при помощи фреймворка Spring Boot требуется:

1. Создать Java класс.
2. Поставить на него нужную аннотацию.
3. Заинжектить в него сервис или репозиторий.
4. Сделать метод и поставить на него аннотацию для обработки соответствующего HTTP метода, а также URL.
5. Делегировать вызов метода в сервис.
6. Обновить конфигурацию безопасности приложения в соответствующем классе.

В процессе у разработчика могут возникнуть следующие вопросы:

1. В каком пакете создать класс, чтобы при сканировании классов Spring Boot подключил новый контроллер?
2. Какие классы аннотаций нужны для маркирования контроллера и его методов по обработке запросов?
3. Где находится класс конфигурации системы безопасности и как правильно прописать в нем новый URL?

Среда разработки не дает ответов на эти вопросы, поскольку в ней отсутствует высокоуровневое, единое представление процесса разработки функциональности приложения. В итоге, для разработки новой функции разработчик вынужден обращаться к документации, сайтам с учебными пособиями или копировать код из существующего решения. Это приводит к потерям времени, кроме того, может оказаться, что код, скопированный из решения двухгодичной давности или устаревшего, но популярного учебника, не отвечает лучшим практикам современной версии фреймворка.

Кроме того, представление структуры кодовой базы не всегда позволяет понять структуру приложения в терминах фреймворка Spring Boot. Например, не всегда можно понять, какой класс отвечает за конфигурацию источников данных, какие роли имеют доступ к каким REST API, какие сервисы обрабатывают те или иные события Spring приложения. Это приводит к тому, что начало работы с существующим приложением начинается с поиска использований классами друг друга, что тоже ведет к потерям времени.

2.1. Управление проектными зависимостями и свойствами

Amplicode обеспечивает подключение следующих основных библиотек фреймворка Spring Boot:

- Spring Web
- Spring Security
- Spring Kafka
- Spring Cloud
- Spring Data JPA
- Spring Data Mongo DB

Как было отмечено выше, подключение библиотек Spring Boot может сопровождаться дополнительной работой по настройке функциональности данной библиотеки. В Amplicode эта задача должна решаться на этапе подключения. Сам процесс подключения библиотеки выглядит следующим образом:

- Выбор необходимой библиотеки.
- Заполнение обязательных значений свойств.
- Выбор связанных библиотек.
- Заполнение обязательных значений свойств для выбранных библиотек.
- Автоматическое обновление файла сборки.
- Автоматическое обновление файла настройки приложения.
- Автоматическая генерация классов конфигурации, при необходимости.

Для поддержки каждой подключаемой библиотеки выполняется следующий набор работ:

- Выявление минимально необходимых свойств конфигурации, которые позволят модулю работать.
- Подготовка шаблонов классов конфигурации.
- Создание сценариев подключения модуля и выявление «смежных» и необходимых библиотек.
- Сценарии настройки модуля и алгоритм изменения кода классов конфигурации и значений свойств при настройке.
- Анализ практик создания функциональности, поддерживаемой подключаемым модулем для последующей реализации в генераторах кода.
- Проектирование пользовательского интерфейса визуальных инструментов.
- Разработка контекстно-зависимых палитр и компонентов для всех видов классов, содержащихся в библиотеке.

Далее рассматривается функциональность, которая реализована в рамках упрощения и ускорения работы с конкретными библиотеками фреймворка Spring Boot.

2.2. Поддержка Spring Framework

Контейнер зависимостей Spring Framework является ядром Spring Boot и применяется во всех проектах, написанных с использованием этого фреймворка. Amplicode позволяет ускорить разработку за счет генерации кода типовых компонентов приложения и упрощения установки зависимостей между существующими компонентами.

В Amplicode реализована следующая функциональность:

- Палитра со списком компонентов Spring Boot приложения и обновление этого списка при создании или удалении компонентов. Компоненты приложения – это, в частности:
  - Сервисы
  - Репозитории
  - Конфигурации
- Генерация кода при установлении зависимостей между компонентами. Код для создания зависимостей зависит от режима создания компонента:
  - Singleton
  - Prototype
- Поддержка установки зависимостей через:
  - Конструктор
  - Поле класса
  - Lookup-метод
- Генерация кода для компонентов – контейнеров свойств (ConfigurationProperties)
  - Поддержка @ConfigurationProperties
  - Поддержка @ConfigurationPropertiesScan
  - Навигация по свойствам из ConfigurationProperties
  - Поддержка @PropertySource аннотации
  - Поддержка переменных среды в названиях свойств
- Генерация методов-обработчиков жизненного цикла компонентов
  - Post Construct (JSR-250)
  - Pre Destroy (JSR-250)
  - Генерация методов для init-method и destroy-method свойств
  - Навигация к методам жизненного цикла из метаданных бина
  - Поддержка интерфейсов InitializingBean и DisposableBean
- Генерация методов-обработчиков для наиболее часто используемых событий жизненного цикла Spring приложения
  - ContextRefreshedEvent
  - ContextStartedEvent
  - ContextStoppedEvent
  - ContextClosedEvent
  - RequestHandledEvent
  - ServletRequestHandledEvent
- Подключение интерфейсов для обработки фаз жизненного цикла приложения
  - ApplicationContextAware
  - BeanNameAware
  - ResourceLoaderAware
  - ServletConfigAware (для Spring Web)
  - ServletContextAware (для Spring Web)

Для реализации данной функциональности проведены следующие работы:

- Изучение алгоритмов конфигурации IoC контейнера Spring для обнаружения компонентов, заданных:
    - в программной конфигурации;
    - при помощи аннотаций.
- Выявление лучших способов создания зависимостей между компонентами и создание соответствующих анализаторов кода.
- Изучение спецификации JSR-330 для поддержки альтернативных способов установления зависимостей между компонентами приложения.
- Изучение спецификации JSR-250 для поддержки генерации кода обработки событий жизненного цикла компонентов приложения.

2.3. Поддержка Spring Web

REST API является самым распространенным способом не только для обмена данными с клиентскими приложениями, но и для интеграции приложений от разных вендоров. Поскольку современные приложения редко работают в изолированной среде, то наличие точек интеграции является обязательной частью спецификации приложения в большинстве случаев.

Для создания REST API в приложениях, написанных на Spring, в большинстве случаев используется библиотека Spring Web. Существующие инструменты незначительно ускоряют разработку с использованием Spring Web. Помощь разработчику сводится к облегчению поиска нужных вызовов API или генерации HTTP запросов для тестирования. Поэтому при создании REST контроллеров приходится тратить значительное время на изучение документации или готовых решений.

Решение Amplicode нацелено на то, чтобы облегчать реализацию типовых ежедневных задач по созданию REST API, с которыми сталкивается каждый разработчик.

В Amplicode реализована следующая функциональность:

- Подключение и конфигурирование Spring Web стартера Spring Boot
- Конфигурация параметров встроенного Web сервера
    - Вендор (выбор из поддерживаемых в Spring Boot)
    - Порт
- Генерация кода «пустых» контроллеров. При генерации присутствует возможность задавать параметры контроллера
    - URL
    - Тип контроллера (синхронный или асинхронный)
- Генерация контроллеров на основе сервисов и репозиториев
    - Добавление нужных компонентов приложения в контроллер
    - Делегирование вызовов методов
    - Реализация эвристических алгоритмов для обработки методов HTTP по умолчанию (GET для методов, начинающихся с find..., POST для методов save() и т.п.)
    - Генерация DTO классов для представления возвращаемых данных
    - Генерация CRUD REST методов на основании методов Spring Data репозиториев
- Инспектор для конфигурирования параметров контроллера:
    - Маппинг
    - Параметры запроса
    - Валидация

    - Формат получаемых/отдаваемых данных
    - Кэширование
    - Настройка CORS
- Генерация наиболее популярной функциональности общего назначения, реализуемой в контроллерах:
    - Загрузка файла
    - Обработка ошибок
    - Преобразование типов данных
    - Обработка заголовков запроса
- Генерация кода для вспомогательных объектов Spring Web
    - Фильтр
    - Обработчик ошибок
    - Валидатор
- Генерация заготовок для тестов REST API
    - Поддержка TestContextFramework
    - Поддержка MockMVC
    - Генерация тестов REST API с использованием RestTemplate

Помимо реализации данной функциональности проведены следующие работы:

- Исследование и анализ способов конфигурации контроллеров.
- Сбор статистики и выявление типовых задач общего назначения.
- Анализ лучших практик по конфигурированию дополнительных компонентов для маппинга, кэширования, валидации и т.д.

2.4. Поддержка Spring Security

Подсистема безопасности – одна из ключевых частей приложения. Для приложений, написанных с использованием Spring Boot, библиотека Spring Security – де-факто стандарт для реализации аутентификации и авторизации. Spring Security гибко настраивается и поддерживает все современные схемы аутентификации и авторизации. Но гибкость библиотеки имеет и обратную сторону – сложность в начале работы с библиотекой и сложность последующей настройки. Amplicode позволяет генерировать типовые конфигурации настроек безопасности и позволяет производить настройку при помощи визуальных инструментов.

В Amplicode реализована следующая функциональность:

- Подключение библиотеки Spring Boot Security и начальная настройка при помощи визуального редактора
- Генерация кода для провайдеров, работающих по протоколу OAuth2
    - Keycloak
    - Google
    - GitHub
    - Okta
- Генерация кода для собственного сервера OAuth2 и поддержка дополнительных свойств конфигурации, таких как:
    - Метод аутентификации клиента
    - Тип авторизации
    - Метод аутентификации пользователя
    - Сфера (scope) авторизации

- Генерация кода для поддержки Form-based аутентификации пользователей, в том числе:
    - Обработчиков успешной авторизации
    - Обработчиков ошибок авторизации
    - Поддержка URL для автоматического перенаправления пользователя
    - Переименование поля формы для передачи пароля
    - Кода для извлечения деталей авторизации при использовании собственной формы
- Генерация кода для провайдеров, работающих по протоколу LDAP, в том числе:
    - Шаблон кода для извлечения authorities из ответа сервера
    - Шаблон кода и инспектор для ContextSource
    - Генерация кода для bind и password аутентификации
- Генерация кода для аутентификации и авторизации при помощи JWT:
    - Задание свойств подключения к серверу
    - Генерация шаблона кода для создания собственного декодера JWT
- Интеграция с библиотекой Spring Web
    - Настройка безопасности REST API (добавление/отключение авторизации для endpoints)
    - Настройка CORS и CSRF
    - Генерация кода для обработки ошибок авторизации при вызове REST API
- Создание подключения к БД для случая использования JDBC хранилища пользователей и ролей
- Создание JPA сущностей для случая использования JPA хранилища пользователей и ролей
- Инспектор для просмотра и изменения настроек системы безопасности.
- Отображение ролей и разрешенных им вызовов REST API и методов сервисов.
- Генерация кода для отображения ролей из внешних серверов аутентификации/авторизации на роли приложения
- Детектирование настроек безопасности:
    - В одном классе конфигурации
    - В разных классах конфигурации
- Детектирование ролей безопасности:
    - В аннотациях методов
    - В программном коде
- Анализаторы кода:
    - Предупреждение о том, что не включен ролевой доступ к методам сервисов при использовании аннотации @Secured
    - Предупреждение о том, что не включен контроль доступа к REST контроллерам при использовании аннотаций доступа в обработчиках HTTP запросов

Помимо реализации данной функциональности проведены следующие работы:

- Анализ различий подключения к различным OAuth2 провайдерам.
- Сбор и анализ практик по написанию кода для конфигурации Spring Security.
- Изучение Spring Security DSL для написания кода по детектированию настроек безопасности во всех классах приложения.

2.5. Поддержка Spring Kafka

Kafka – широко распространенное решение для передачи сообщений между сервисами бизнес-приложений, построенных с использованием микросервисной архитектуры. Для фреймворка

Spring Boot существует библиотека для работы с Kafka, и Amplicode позволяет облегчить и ускорить работу с данной библиотекой.

В Amplicode реализована следующая функциональность:

- Подключение библиотеки Spring Boot Kafka с использованием визуальных инструментов
- Конфигурирование подключения к Kafka при помощи визарда:
    - Задание адреса сервера
    - Задание класса сообщения
    - Выбор сериализаторов и десериализаторов сообщений
    - Создание фабрик для генерации компонентов для приема сообщений из Kafka
    - Создание фабрик для генерации компонентов для отправки сообщений в Kafka
- Инспектор свойств конфигурации Spring Boot Kafka для тонкой настройки подключения и обработки сообщений
    - Разбор кода фабрики компонентов для отправки сообщений и отображение свойств в инспекторе
    - Разбор кода фабрики компонентов для приема сообщений и отображение свойств в инспекторе
- Отображение сконфигурированных подключений к очередям сообщений
- Генерация шаблонного кода для приема сообщений из Kafka
    - В сервисах
    - В контроллерах
- Генерация шаблонного кода для отправки сообщений
    - В сервисах
    - В контроллерах
- Компоненты палитры для генерации кода отправки и приема сообщений

Помимо реализации данной функциональности проведены следующие работы:

- Исследование практик написания кода для:
    - Высоконагруженных очередей
    - Минимизации случаев дублирования отправки и приема сообщений
    - Обработки ситуации недоступности очереди сообщений
- Анализ частоты использования проприетарных конвертеров сообщений и целесообразности генерации шаблонного кода для нестандартных сериализаторов и десериализаторов

2.6. Поддержка Spring Boot Cloud

При написании практически любого современного корпоративного приложения нужно учитывать тот факт, что с большой вероятностью оно будет разворачиваться в облачной среде. Развёртывание в облачной среде имеет ряд особенностей, одна из которых – широкое использование сервисов среды для выполнения «сервисных» задач: журналирование событий, управление конфигурациями, хранение настроек для доступа к базам данных и пр.

Для работы с облачными сервисами используется набор библиотек Spring Boot Cloud. Amplicode позволяет облегчить подключение, конфигурирование и использование некоторых библиотек из этого набора в приложениях.

В Amplicode реализована следующая функциональность:

- Подключение библиотек Spring Boot Cloud при помощи визуальных инструментов
    - Spring Cloud Config
    - Micrometer Tracing
    - Spring Cloud Stream
- Визуальные инструменты, учитывающие текущую конфигурацию приложения и генерирующие код для интеграции уже подключенных библиотек в облачную среду:
    - Работа с Kafka - для Spring Cloud Stream
- Генерация кода для хранения и конфигураций приложения. Поддерживаются следующие хранилища:
    - Файловая система
    - HashiCorp Vault
    - Хранилище данных с доступом по JDBC
- Визуальные инструменты для простой замены типа хранилища конфигураций
- Визуальные инструменты для конфигурирования системы логирования (журналирования) приложения
    - В том числе - конфигурирование Micrometer Tracing
- Компоненты палитры для генерации кода по чтению конфигурации из различных хранилищ

Помимо реализации данной функциональности проведены следующие работы:

- Изучение способов развертывания серверов конфигурации в облачной среде
- Изучение лучших практик логирования в облачных приложениях

2.7. Поддержка Spring Data JPA

Amplicode использует JPA как основной слой доступа к данным в целевом приложение на Spring Boot и позволяет облегчить и ускорить работу с JPA, Hibernate и Spring Data JPA. В Amplicode реализована следующая функциональность:

- Подключение и конфигурация библиотеки Spring Data JPA
- Визуальный инструмент для настройки ORM Hibernate
- Поддержка возможностей JPA, ORM Hibernate и Spring Data JPA
- Подключение и конфигурация инструментов для миграции БД:
    - Flyway
    - Liquibase
- Подключение дополнительных библиотек:
    - Hibernate Validator
    - Lombok
    - Hibernate Types
    - MapStruct
    - Hibernate Envers
    - Spring Data Audit
    - Blaze Persistence
- Создание и конфигурирование JPA Data Sources:
    - В файле настроек
    - Программно – в компонентах
- Анализаторы кода:
    - Предупреждение об отсутствии драйвера БД в списке зависимостей проекта
    - Предупреждение о том, что нет конфигурационных свойств для источника данных

В рамках функциональности по генерации и анализу кода выполнен ряд дополнительных задач.

- Поддержка:
  - Спецификации JPA 3.1
  - Hibernate 6
  - Hibernate Envers
  - Spring Data Audit
- Дополнительные средства для эффективной работы с кодом
  - Интеллектуальные преобразования кода JPA сущностей и репозиториев
  - Улучшение генерации разностных DDL скриптов
  - Улучшение генерации кода DTO
  - Поддержка собственных стратегий именования объектов баз данных
  - Развитие функционала создания JPA сущностей из базы данных

Далее рассматривается детальная спецификация для расширения функциональности по генерации и анализу кода для работы с JPA.

2.7.1. Поддержка спецификации JPA 3.1 и Hibernate 6

В марте 2022 года вышел релиз Hibernate 6. В него включено большое количество изменений и улучшений API. Также новая версия фреймворка поддерживает последнюю версию спецификации JPA 3.1. Amplicode сохраняет актуальность для последних версий одного из основополагающих фреймворков по работе с реляционными данными, поддерживает как последнюю версию Hibernate, так и последнюю версию спецификации.

В Amplicode реализована следующая функциональность:

- Добавлена поддержка @GeneratedValue(strategy=GenerationType.UUID), для первичных ключей типа UUID в соответствии со спецификацией JPA 3.1
  - В дизайнере сущностей (инспекторе) добавлена поддержка UUID
  - Генераторы поддерживают UUID тип ID атрибута, в том числе при генерации сущности из базы данных
- Поддержана аннотация @JdbcTypeCode в соответствии с документацией Hibernate 6, в том числе маппинг XML и JSON в окне создания атрибутов, генерации поля из колонки базы данных и при генерации скриптов версионирования базы данных
- Поддержаны типы данных ZoneOffset, InetAddress в окне создания атрибутов, генерации поля из колонки базы данных и при генерации скриптов версионирования базы данных
- Поддержан новый способ определения пользовательских типов через аннотацию @JavaType для генерации скриптов версионирования базы данных, а также при генерации полей из колонки базы данных
- Поддержаны аннотации @JdbcTypeRegistration и @JdbcType для генерации скриптов версионирования базы данных, а также при генерации полей из колонки базы данных в случае использования нестандартного SQL типа
- Поддержана аннотация @Type в части настроек маппингов для генерации полей из колонки базы данных и создания скриптов обновления базы данных
- Поддержаны аннотации @TimeZoneStorage и @TimeZoneColumn в части создания атрибута, отображающего дату и время и для генерации корректных скриптов обновления базы данных
- Адаптирована работа с большими типами данных (Lob) при создании полей соответствующих типов (String, sql.CLOB, sql.BLOB), а также при генерации скриптов

версионирования базы данных и при генерации полей соответствующих типов из существующих колонок базы данных
- Поддержана, представленная в Hibernate6 стратегию CamelCaseToUnderscoresNamingStrategy, предназначенная для вычисления "физического" имени колонки по “логическому” имени атрибута

2.7.2. Интеграция с Envers

Hibernate Envers (https://hibernate.org/orm/envers/) — это популярный модуль, работающий в связке с JPA и Hibernate, который позволяет отслеживать и сохранять изменения значений полей сущностей, а также работать со старыми состояниями этих сущностей (восстанавливать их значения из прошлых ревизий). Такие требования выдвигаются практически к любому корпоративному приложению.

Упрощение работы с Hibernate Envers, а также автоматизация рутинных операций, таких как генерация Flyway и Liquibase скриптов для таблиц, хранящих изменения данных, сохранит большое количество времени на изучение и применение этой технологии.

В рамках интеграции с Hibernate Envers решены следующие задачи:

- Изучена документация Hibernate Envers, а также популярные кейсы использования технологии из открытых источников, таких как GitHub
- Проведен анализ исходного кода решения hbm2ddl для разбора логики создания DDL скриптов
- Предоставлены средства генерации DDL для изменения таблиц аудита в Flyway versioned migration и Liquibase diff changelog, учитывая множество разнообразных настроек, возможных в Hibernate Envers
- Добавлена возможность расширять репозитории Spring Data JPA с помощью RevisionRepository, представленного в проекте Spring Data Envers
- Предоставлены средства генерации для объекта @RevisionEntity с различными опциями, поддерживаемыми Hibernate Envers
- Расширен Entity Inspector для визуального управления аннотацией @Audited, как на уровне класса, так и на уровне полей сущности.

2.7.3. Интеграция с Spring Data Audit

Аудит — важная функция для корпоративных приложений, позволяющая обеспечить прозрачность и адресность изменения данных. В то время как Hibernate Envers занимается вопросами версионирования записей, Spring Data Audit позволяет отслеживать и журналировать авторов и время этих изменений.

В рамках интеграции с Spring Data Audit решены следующие задачи:

- Изучена документации Spring Data Audit, а также популярные кейсы использования технологии из открытых источников, таких как GitHub
- Добавлена в JPA Palette группа для полей аудита: Created By, Created Date, Last Modified By, Last Modified Date
- Поддержка всех указанных в документации типов данных для полей Created Date и Last Modified Date
- Помощь при генерации классов, реализующих AuditAware и ReactiveAuditorAware интерфейсы

- Возможность включения @EntityListeners(AuditingEntityListener.class) из панели JPA inspector
- Поддержка AbstractAuditable родительского класса для JPA сущностей в плане корректной генерации DDL и Liquibase Changelog-ов

2.7.4. Интеграция с Entity View из библиотеки Blaze-Persistence

Для корпоративных приложений характерна работа со сложными графами данных. При интенсивной работе с данными через JPA разработчики сталкиваются с проблемами, такими как LazyInitializationException или N+1, связанные с "ленивой" загрузкой данных.

Существует два традиционных решения упомянутых проблем:

- Выделение слоя DTO и отказ от ленивой загрузки ассоциаций. Этот подход приводит к большому количеству шаблонного кода, который необходимо поддерживать, а также к проблемам с производительностью системы. Проблемы производительности связаны с тем, что Hibernate всегда выбирает все поля, представленные простыми типами данных. В итоге при запросе с join мы получаем мультипликацию пересылаемых данных, которые не используются в логике или отображении.
- Использование анти-паттерна "open-session in view". Это простой в реализации подход, однако он негативно влияет на производительность и масштабируемость приложения.

Blaze Persistence Entity Views — это фреймворк, нацеленный на решение вышеуказанных проблем через выделение слоя DTO, однако без характерных для этого негативных эффектов. В рамках интеграции с Blaze Persistence Entity Views решены следующие задачи:

- Изучена документацию и примеры кода Blaze Persistence Entity Views
- Предоставлены визуальные средства для генерации объектов EntityView из существующих сущностей в части полей простых типов, @Embedded полей и ассоциаций ManyToOne, OneToOne, OneToMany и ManyToMany
- Поддержана возможность создания Updateble Views
- Предоставлены средства для создания Spring Data репозиториев для объектов EntityView
- Реализованы инспекции при расхождении маппингов в процессе рефакторинга имен полей в JPA сущностях

2.7.5. Поддержка расширенных возможностей JPA, Hibernate. Интеллектуальные преобразования JPA кода.

Разработчик много работает с кодом. По разным оценкам, осмысление задачи занимает от 20 до 40% рабочего времени, оставшаяся же часть времени уходит именно на написание и отладку кода. С этой точки зрения, сокращение времени на физическое написание кода, а также отладку, кратно повысит бизнес-показатели эффективности всего процесса разработки программного обеспечения. В части работы с данными количество шаблонного кода особенно велико. JPA сущности и DTO фактически повторяют друг друга, DDL скрипты имеют однозначное отображение на код JPA сущностей, и т.д.

Функциональность Amplicode позволяет снизить издержки на непосредственную разработку кода.

2.7.6. Предоставление дополнительных средств эффективной работы с кодом

Корпоративные приложения характеризуются моделью данных с большим количеством сущностей. Для определения такой модели широко применяются некоторые паттерны и приемы, которые и предлагается автоматизировать:

- Извлечение некоторого набора полей в @MappedSuperclass с последующей генерализацией модели. Фактически это означает выделение некоторых полей в модели, таких как id, createdBy, tenantId, а также общих методов hashCode(), equals(), toString(), prePersist() и т.д. и перемещение их в класс предка. Такая функция позволяет сделать код более лаконичным и читабельным. В ручном режиме генерализация модели занимает много времени и является процессом, в котором легко допустить ошибку. Автоматизация этого процесса разгрузит разработчика от ручной работы и устранит потенциальные ошибки.
- Генерацию Spring Data JPA репозиториев для множества сущностей. В большинстве корпоративных систем требуется обеспечение CRUD операций практически для каждой сущности. Множественная генерация репозиториев с настройкой и применением единообразного подхода является удобной функцией, которая упростит написание исходного кода, особенно на начальный стадиях проекта.
- Конвертацию POJO объектов в JPA сущности в соответствии с заданными правилами. Часто определение объекта, который впоследствии будет сохранен в базу данных, приходит из сторонних сервисов. Преобразование этих объектов в JPA сущность может быть трудоемкой операцией, если объект содержит ссылки на другие объекты. Amplicode позволяет максимально автоматизировать эту задачу.

Кроме шаблонных операций, существуют распространенные ошибки при использовании JPA. В рамках задачи повышения качества кода и снижения человеческого фактора реализованы сделать следующие инспекции:

- Каскадное удаление приводит к непредсказуемым последствиям в ManyToMany связях.
- Ленивая загрузка не работает для OneToOne ассоциаций, если объявлена не на стороне владельца связи.

Помимо реализации инспекций предоставляются корректные исправления кода для его правильного функционирования.

2.7.7. Расширенные возможности генерации разностных DDL скриптов

Amplicode предоставляет возможности генерации скриптов версионирования баз данных. Возможна генерация скриптов с продвинутыми настройками, к которым относятся:

- Генерация скриптов в случае композитного первичного ключа, заданного посредством @IdClass
- Возможность задавать шаблоны для генерации новых Liquibase changeset-ов
- Возможность генерировать разностные скрипты не только по всей модели целиком, но и по ее части
- Возможность учитывать конфигурационные параметры для ID типа Sequence из аннотации @GenericGenerator
- Добавления скрипта для создания Sequence при отображении DDL для сущностей с соответствующим типом ID
- Поддержка генерации Liquibase скриптов для ассоциаций типа OneToOne с указанием JoinTable

2.7.8. Улучшение функционала генерации DTO

Работа с DTO – одна из самых рутинных частей при разработке приложений, активно взаимодействующих с данными. Предоставление инструментов для создания, рефакторинга и управления этими объектами безусловно ускорит сам процесс разработки и снизит количество возможных ошибок, продиктованных человеческим фактором.

В рамках функционала генерации DTO Amplicode решены следующие задачи:

- Поддержана возможность создавать DTO в виде Java record, для пользователей, использующих Java 17+
- Предоставлена возможность генерировать "fluent setter" методы, т.е. методы возвращающие сам объект после простановки значения в поле
- Предоставлена возможность маркировать DTO классы аннотацией @JsonIgnoreProperties(ignoreUnknown = true), при использовании популярной библиотеки Jackson
- Создан механизм сопоставления DTO объектов, а также навигация с соответствующими им JPA сущностями посредством
    a. Комментария определенного паттерна, который можно настраивать
    b. Настраиваемой конвенции именования класса DTO
    c. Наличия метода маппинга при использовании интерфейсов библиотеки MapStruct
- Предоставлены средства рефакторинга, которые будут позволять синхронизировать изменения полей сущностей с соответствующими полями в DTO

2.7.9. Поддержка собственных стратегий именования объектов баз данных

Часто в корпоративных приложениях используют нестандартные стратегии преобразование полей JPA сущностей в колонки базы данных. Это связано с устоявшимися в компании конвенциями именования. Hibernate предоставляет механизм для переопределения стандартных конвенций, а также предлагает набор базовых реализаций. В Amplicode помимо поддержки базовые стратегий именований, существует возможность использовать стратегии, определенные на уровне проекта. Amplicode предоставляет возможности:

- Механизма определения существующих реализацию стратегий именования из классов проекта
- Вычисления “физическое” имя из "логического" имени
- Вычисления обратного преобразование из "физического" имени в "логическое" для решения задачи генерации сущностей из таблиц БД

2.7.10. Развитие функционала создания JPA сущностей из базы данных

Часто разработчики практикуют подход, при котором изначально создаются таблицы в базе данных. Затем разрабатываются JPA сущности, соответствующие изменениям в базе данных. Этот процесс также может быть автоматизирован так, чтобы ручная работа сводилась к минимуму.

Amplicode предоставляет генератор сущностей из таблиц в базе данных, а также атрибутов уже имеющихся сущностей из колонок таблиц. К продвинутым возможностям генерации скриптов относятся:

- Возможность задавать параметры генерации ID в отсутствии значения по умолчанию в определении колонки первичного ключа
- Широкие настройки при генерации сущностей и колонок из базы данных, а именно:

    - Префикс и/или суффикс, который не надо учитывать для именования поля, соответствующему колонке в базе данных
    - Префикс и/или суффикс, который не надо учитывать для именования класса сущности, соответствующему таблице в базе данных
- Оповещение о том, что некоторые обязательные (not null) колонки без указания значений по умолчанию не включены в маппинг
- возможность мигрировать индексы и ограничения

2.8. Поддержка Spring Data MongoDB

Корпоративные приложения имеют дело с различными видами данных, и для хранения и обработки некоторых лучше подходят NoSQL решения. Типичный пример - хранение слабо структурированных документов. Обычно для этих целей используется MongoDB – популярное документно-ориентированное, масштабируемое хранилище данных. В Spring Boot для работы с MongoDB применяется библиотека Spring Data Mongo, по философии сходная со Spring Data JPA.

Для поддержки работы с MongoDB в Amplicode реализован следующий набор задач:

- Подключение библиотеки Spring Data Mongo и начальная конфигурация соединения с MongoDB
- Генерация классов – документов
- Генерация Spring Data Mongo репозиториев
- Создание и конфигурирование MongoDB Data Sources
    - В файле настроек
    - Программно – в компонентах
- Отображение источников данных MongoDB в проектном навигаторе
- Отображение MongoDB репозиториев в проектном навигаторе
- Инспектор для редактирования свойств документа
- Палитра компонентов для редактирования кода класса-документа

3. Общее развитие инструмента разработки Amplicode

3.1. Поддержка языка программирования Kotlin

Говоря о разработке бэкенд приложений на Amplicode, всюду в этом документе подразумевается использование языка Java - основного языка программирования для фреймворка Spring Boot. Java — это язык, изначально поддержанный в Amplicode, т.к. он более распространен и проще по своей синтаксической структуре.

Для увеличения потенциальной пользовательской базы платформы Amplicode и упрощения ее продвижения необходимой мерой стало добавление поддержки языка Kotlin.

Kotlin — это современный мультипарадигменный язык программирования. Он содержит множество современных языковых конструкций, таких как "properties", "value classes", "nullability", содержит базовую поддержку легковесных потоков исполнения (coroutines). Этот язык считается более лаконичным и экспрессивным, чем Java. Он поддерживается всеми базовыми технологиями, лежащими в основе Amplicode и ее инструментов: IntelliJ IDEA, Spring, Maven.

Для поддержки работы Kotlin в Amplicode выполнены следующие задачи:

- Проведен анализ синтаксических структур и уникальных особенностей языка Kotlin в плане возможности их применения к различным элементам кода, которые генерируются в Amplicode.

- Доработаны визуальные дизайнеры, генераторы кода, фоновые проверки, контекстно-зависимые действия и другие возможности Amplicode, чтобы они функционировали в файлах исходного кода на языке Kotlin дополнительно к Java. В том числе:
    - Средства по работе с JPA
    - Средства по работе с Spring Core, Spring MVC, Spring Security, Spring Boot Cloud, Spring Kafka, Spring Data, MongoDB.

3.2. Поддержка системы сборки Gradle

Основной системой сборки, изначальной поддерживаемой в инструментах Amplicode, является Maven. Maven — это наиболее популярная система сборки Java-проектов, используемая 76% разработчиков (согласно Snyk JVM Ecosystem Report 2021).

Однако для повышения охвата потенциальной аудитории продукта, более эффективного продвижения было принято решение о поддержка и другой, альтернативной системы сборки, также популярной в Java экосистеме - Gradle.

Gradle используется около 38% разработчиков (согласно тому же отчету; один разработчик может использовать несколько систем сборки). Gradle имеет ряд преимуществ по сравнению с Maven, например возможность инкрементальной сборки, инкрементальной компиляции проектов, а также более простая кастомизация скриптов сборки.

В рамках данной задачи усовершенствована логика различных подсистем Amplicode, для чтения структуры Gradle проекта или изменения содержимого Gradle скрипта. В том числе:

- Функции установки библиотек и Spring Boot стартеров в подсистемах поддержки соответствующих технологий.
- Функции анализа структуры проекта, его модулей.

3.3. Поддержка многомодульной структуры проекта

Базовая структура проекта, которая в первую очередь поддерживается в Amplicode — это простой одномодульный проект. Для небольших приложений одномодульная структура оптимальна. Тем не менее, в сфере разработки бизнес-приложений на Spring Boot обычным делом является деление приложений на несколько модулей. Разделение проекта на модули позволяет повысить структурированность содержимого проекта, четко выделить API между модулями, что повышает качество кода.

В многомодульном проекте каждый модуль может иметь свой дескриптор сборки (pom.xml или build.gradle), содержащий свойства модуля, зависимости от внешних библиотек и т.д. В рамках поддержки многомодульной структуры проекта Amplicode проведены следующие работы:

- Добавлен выбор модуля во все действия по созданию элементов кода: сущностей, классов передачи данных, контроллеров, конфигураций подсистемы безопасности и др.
- Переработана панель структуры проекта (окно инструментов Amplicode). Панель структуры отображает файлы и классы проекта с учетом структуры модулей.
- Переработаны дизайнеры кода и других визуальные средств Amplicode, включая автодополнения, списки опций и прочие подсказки для учета зависимостей модулей при поиске классов. Опциями в текущем контексте могут служить только классы, расположенные в текущем модуле или в модулях, от которого данный модуль зависит по компиляции.
- Доработаны механизмы, автоматически добавляющих зависимости от внешних библиотек в проект (например, Liquibase, JDBC драйверы и др.), чтобы они корректно выбирали

нужный дескриптор сборки, соответствующий конкретному модулю, либо корневой дескриптор.