Оглавление

[Введение 3](#_Toc138121012)

[Глава 1. Структура предприятия 4](#_Toc138121013)

[Глава 2. Выполнение индивидуального задания 5](#_Toc138121014)

[2.1 Подготовительный этап работы. Знакомство с предприятием 7](#_Toc138121015)

[2.2 Исправление несоответствий в диаграммах классов 8](#_Toc138121016)

[2.3 Внесение исправлений в диаграмму с созданием новых классов 11](#_Toc138121017)

[2.4 Использование продвинутых возможностей Flexberry Designer. 13](#_Toc138121018)

[2.5 Работа с системой управления жизненным циклом 15](#_Toc138121019)

[2.6 Поездка на конференцию. Публикация научной статьи 15](#_Toc138121020)

[2.7 Изучение возможностей фреймворка Ember.js 17](#_Toc138121021)

[Глава 3. Выводы и предложения по улучшению 19](#_Toc138121022)

[Заключение 22](#_Toc138121023)

[Список литературы 23](#_Toc138121024)

[Приложение А 25](#_Toc138121025)

# Введение

Для прохождения производственной практики, предусмотренной учебным планом, я был направлен в ООО «Траектория времени», занимающееся разработкой аналитических систем, мобильных приложений, решений на базе блокчейн-технологий. На данный момент компания находится на IT-рынке более 8 лет. Самой известной разработкой «Траектории Времени» является цифровая платформа  
«Т-Мобис», использующаяся для технического аудита [1]. Известными партнерами компании является ООО «Скайори», АО «ЭнергосбыТ Плюс», энергетический холдинг «Сибирская генерирующая компания» [2].

Задачи, цели и график практики расписаны в индивидуальном задании и в рабочем графике. Кратко резюмируя содержание перечисленных документов, можно выделить в качестве целей практики изучение прикладных технологий в области программной инженерии, получение практических навыков разработки и поддержки программного обеспечения, получение производственного опыта сотрудничества в рамках производственной команды.

В комплекс основных технологий, используемых на предприятии, входит инструмент для создания программного обеспечения Flexberry Designer, использующийся для облегчения и ускорения процесса разработки программного обеспечения. Данный инструмент разработан на языке C# с применением фреймворка Windows Forms и системы управления базами данных (СУБД) Microsoft SQL Server. Инструмент позволяет создавать модели программного обеспечения в формате UML-диаграмм, из которых впоследствии возможна генерация программного кода. Также на производстве используется среда разработки Microsoft Visual Studio, редактор кода Microsoft Visual Studio Code, языки программирования С# (фреймворк ASP.Net) и JavaScript (Ember.js).

# Глава 1. Структура предприятия

ООО «Траектория времени» функционирует в основном в формате удаленных команд, взаимодействующих посредством мессенджеров (Telegram) и средств видео-конференц-связи (Discord, Zoom). Команды распределены согласно решаемым ими задачам, в каждой из них есть так называемый тимлид (то есть лидер команды), отвечающий за работы вверенного ему коллектива. Работа команд ведется преимущество по модели SCRUM с использованием таких ритуалов, как «дэйлики» (то есть встречи для планирования и краткого подведения итогов), «ретроспектива» (подведения итогов за определенный длительный период).

В целях сохранения конкретных преимуществ и защиты от рисков предприятие заключает с сотрудниками договоры о неразглашении конфиденциальной информации.

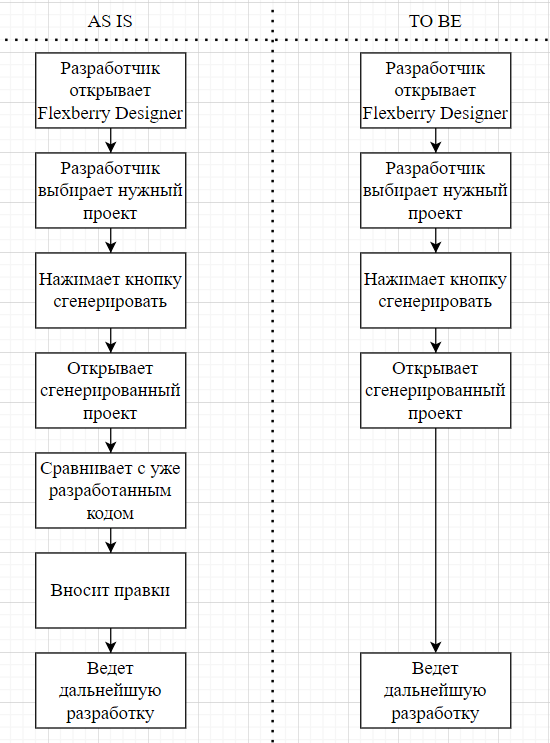
Для распределения задач используется встроенные в веб-систему управления жизненным циклом GitLab дашборды, по которым распределяются и перемещаются задачи (виртуальный аналог Kanban-доски). Каждая задача как правило имеет срок, ответственное лицо и статус выполнения. Таким образом, мое взаимодействие с руководителем от практики и прочими сотрудниками происходило преимущественно в удаленном формате, поскольку большинство из них не присутствовало (физически) на предприятии.

На предприятии имеется полностью оборудованная небольшая кухня, на которой можно провести обеденное время. «Траектория времени» организует так называемые книжных клуб, на котором происходит обмен профильных и непрофильных знаний, обсуждение актуальной литературы, присутствуют летние и зимние выездные мероприятия, командные обеды, встречи «один на один» с руководителем для обсуждения профессионального развития, имеются собственные корпоративные курсы.

# Глава 2. Выполнение индивидуального задания

Основной задачей производственной практики было приведение в соответствие диаграмм классов, встроенных в инструмент Flexberry Designer, с разработанным программным кодом, в разной степени (в зависимости от конкретной диаграммы) отличавшихся в силу нарушения подхода разработки Model First (то есть разработка на основе модели). Необходимость решения задачи обусловлена нарушением стандартного производственного бизнес-процесса (см. схему клиентского пути на рис. 1) в силу описанных причин, согласно которому сгенерированный по модели программный код должен быть сразу же пригоден к дальнейшему применению в разработке.

***Рисунок 1. Карта клиентского пути разработчика  
AS IS—TO BE***



Однако перед проведением необходимых работ сгенерированный программный код требовал «ручной» актуализации (путем сравнения сгенерированного и продуктового кода), в процессе которой накапливались ошибки (по большей части, опечатки) и «излишки» (то есть неиспользуемый программный код). Данную проблему раскроем в виде диаграммы активностей (рис. 2), желтым выделены шаги, которые необходимо устранить.



***Рисунок 2. Диаграмма активностей процесса использования CASE-инструмента Flexberry Designer***

Проблема возникла с процессе модификации и поддержки программного проекта. Изменения, внесенные в программный код, не всегда находили свое отражение в диаграмах. Таким образом, модель приложения устаревала. Суть поставленной задачи заключается в изменении диаграмм, путем добавления и удаления атрибутов и введения новых классов, так, чтобы они соответствовали измененному в процессе разработке программному коду.

В силу отсутствия программных интерфейсов для автоматического приведения диграмм в соответствие разработанному коду, данную работу пришлось выполнять вручную.

## 2.1 Подготовительный этап работы. Знакомство с предприятием

Работа с корпоративными информационными системами невозможна без предварительной настройки окружения и получения необходимого доступа к ресурсам компании.

Таким образом, первой задачей, возникшей во время практики, было подключение к удаленной корпоративной сети (VPN), установка необходимых сертификатов в доверенные корневые центру сертификации, авторизация на рабочем сервере (GitLab), ознакомление с назначенной задачей (было сделано с помощью фильтра по параметру Assignee — т.е. назначенное лицо). В процессе решения вышеописанных задач было изучено несколько статей (написанных сотрудниками предприятия) о старте работы в компании [3–5]. В них раскрывались тонкости работы, такие как генерация ssh-ключей для дальнейшей работы с системой, значение корпоративного стандарта именований так называемых «лэйблов» (статусов) задач.

В хода выполнения задачи оборвалось соединение с удаленной корпоративной сетью, после чего было произведено повторное подключение.

Также для выполнения следующих заданий по актуализации диаграмм классов и программного кода были созданы две ветки в системе контроля версий Git. Теперь, согласно инструкциям руководства, требовалось обратиться к инструменту Flexberry Designer [6]. После успешной установки и изменения конфигурации приложения была произведена автоматическая генерация C# кода.

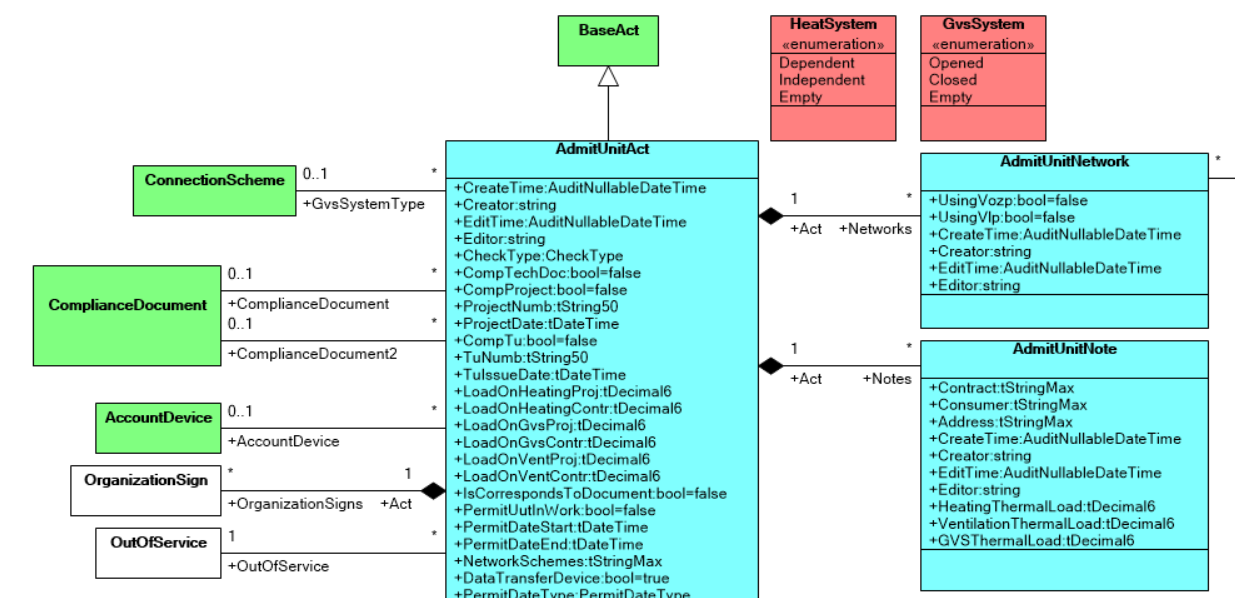
Для автоматической генерации Frontend-части кода требовалось установить ряд пакетов, таких как yarn, ember (с помощью команды *npm install[[1]](#footnote-1)*) и их настройка для дальнейшей работы[[2]](#footnote-2).

На данном этапе мы с коллегами отвлеклись, чтобы познакомиться со всей командой предприятия. Это заняло примерно 10 минут. Нам было рассказано о задачах, которые решаются в компании, произошло знакомство с сотрудниками предприятия, ее корпоративной культурой и историей. Далее мы перешли непосредственно к генерации программного кода, которая после еще около двух часов работы с настройками начала исправно работать и выдавать результат.

## 2.2 Исправление несоответствий в диаграммах классов

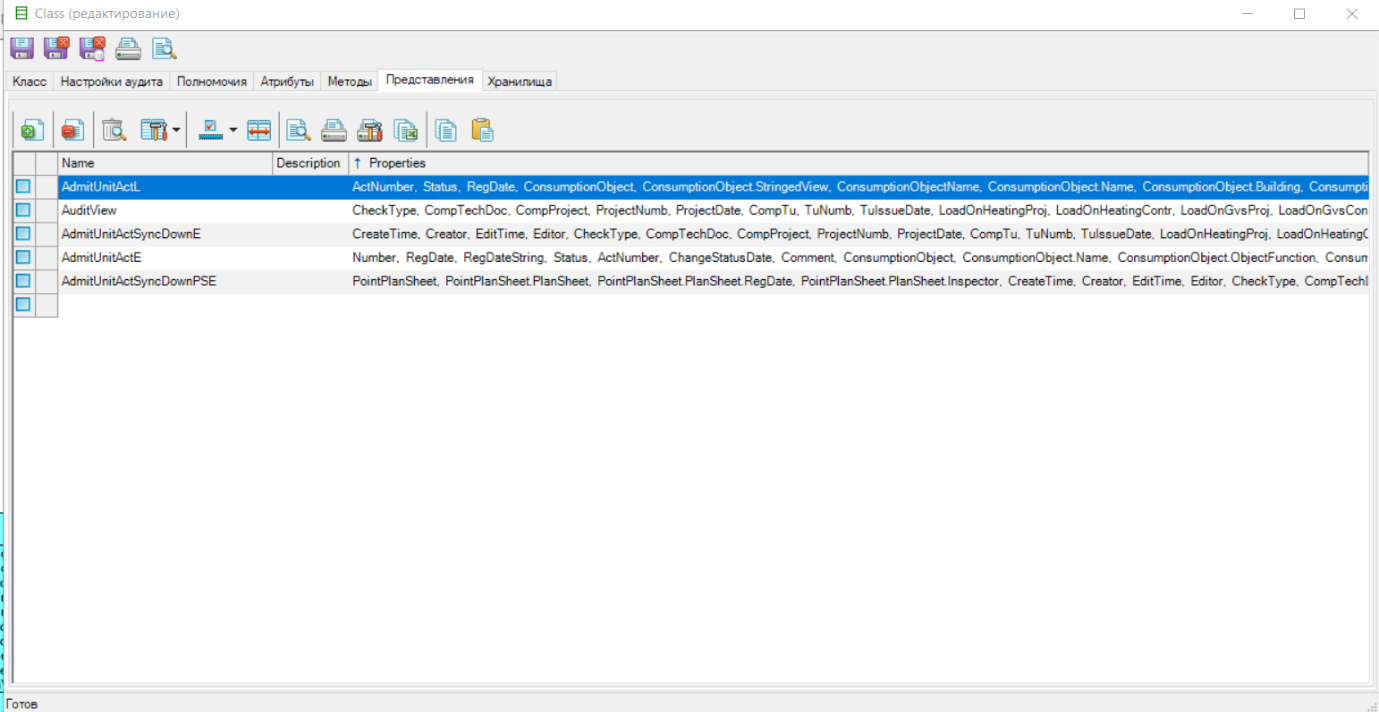
Для наглядности приведем часть диаграмм из корпоративной системы (рисунки 3, 4).

Всего подобных диаграмм в работе было восемь (всего же таких диаграмм около сорока), каждую из них было необходимо привести в соответствие с разработанным программным кодом. Фактически в контексте диаграмм такой подход можно охарактеризовать как реверсивную разработку.

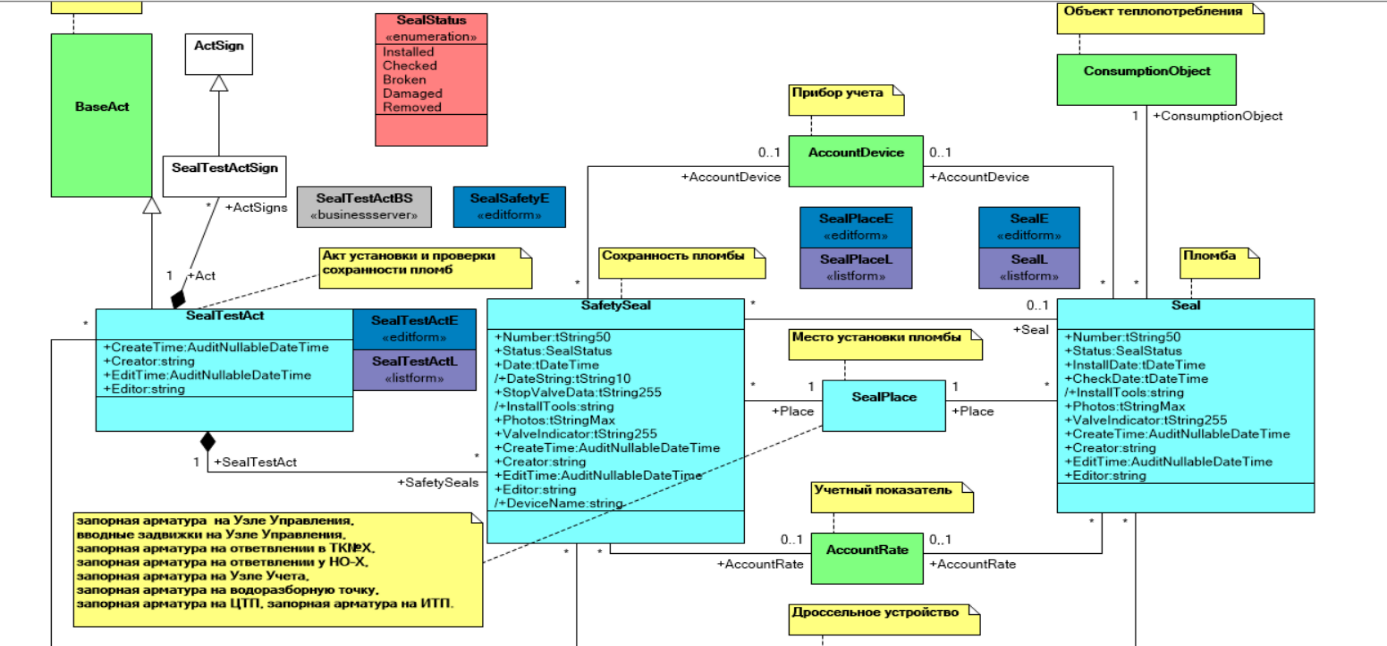


***Рисунок 3. Часть диаграммы «Акт допуска в эксплуатацию УЭТТ»***

Для этого необходимо нажать на нее правой кнопкой компьютерной мыши, после чего выбрать в меню пункт «Редактировать свойства». Откроется соответствующее окно. После этого переходим во вкладку «Представления» (рис. 5). По условиям задач, требовалось обработать все неактуальные представления каждого класса.



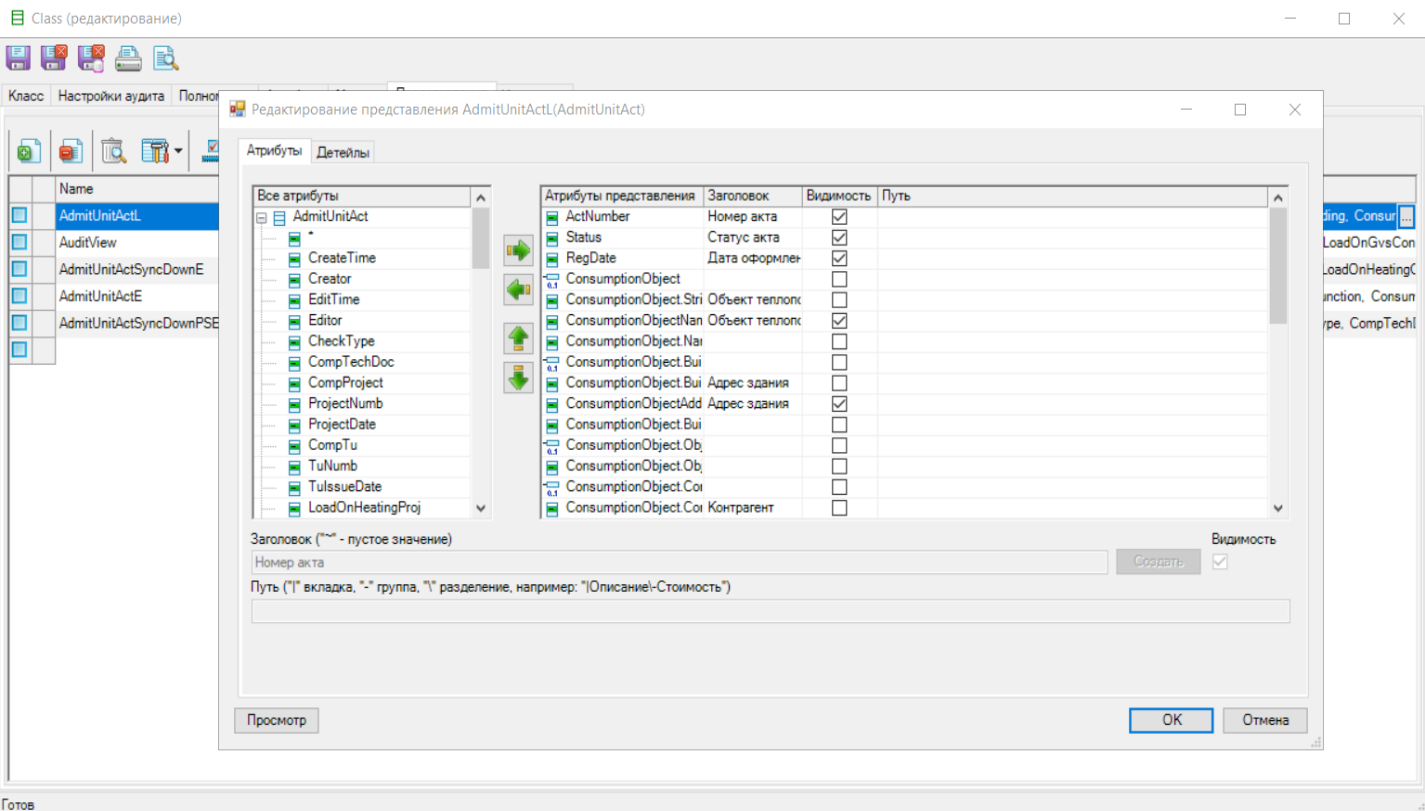
***Рисунок 5. Окно выбора представления***



***Рисунок 4. Часть диаграммы «Акт установки и проверки сохранности пломб»***

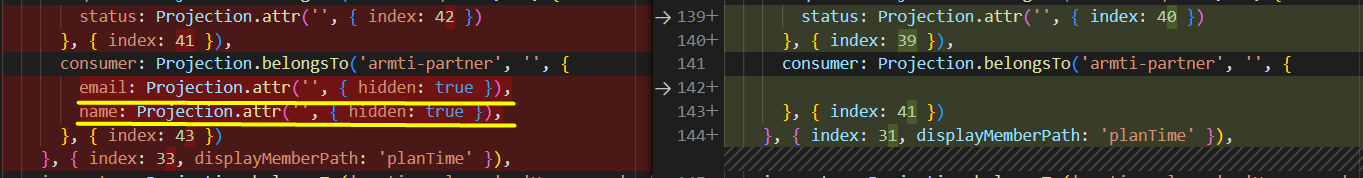
Затем для выполнения задачи нажатием на кнопку «…» в правой части окна необходимо открыть редактор представления (см. рис. 6). С помощью кнопок соответствующих кнопок (зеленые стрелки) осуществляется управление атрибутами. Так, стрелки, направленные вправо и влево, позволяют добавлять и удалять атрибуты из представления, стрелки, направленные вверх и вниз, дают возможность менять очередность атрибутов. Их порядок особенно важен для корректного (с пользовательской точки зрения) отображения «видимых» (то есть отображаемых на экране) атрибутов страницы.

Для нахождения мест, в которых требует редактирование обратимся к редактору кода Visual Studio Code. С помощью системы контроля версий сравним программный код, сгенерированный по диаграмме (см. правую половину рис. 7), с тем, что находится на корпоративном сервере GitLab (слева на рис. 7). Найдем несоответствия (как правило они выделены ярко-красным цветом).



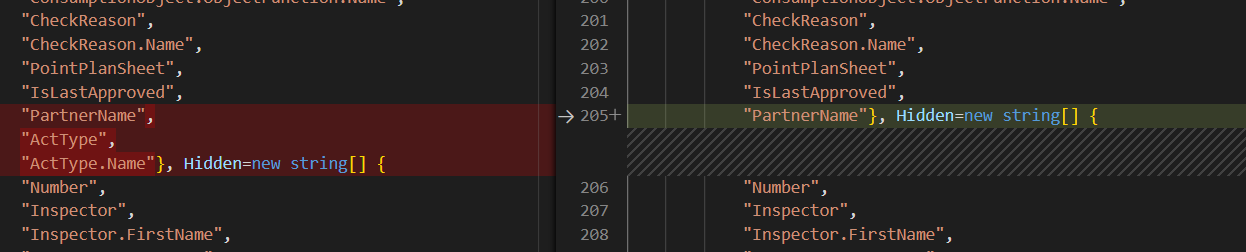
***Рисунок 6. Редактор представлений***

***Рисунок 7. Пример несоответствия в программном коде***



Исправим его в соответствующей диаграмме путем добавления нужных полей в диаграмму. Сделаем то же самое для других несоответствий (в том числе в других диаграммах), после чего сгенерируем и проверим результат.

Аналогичным образом велась актуализация диаграммы для приведения ее в соответствие с программным кодом на языке C# (пример несоответствия можно увидеть на рис. 8), так как платформа поддерживает генерацию как frontend (описано выше), так и backend. В случае отсутствия backend-атрибута необходимо добавить его как «невидимый», в таком случае расположением элементов можно пренебречь.



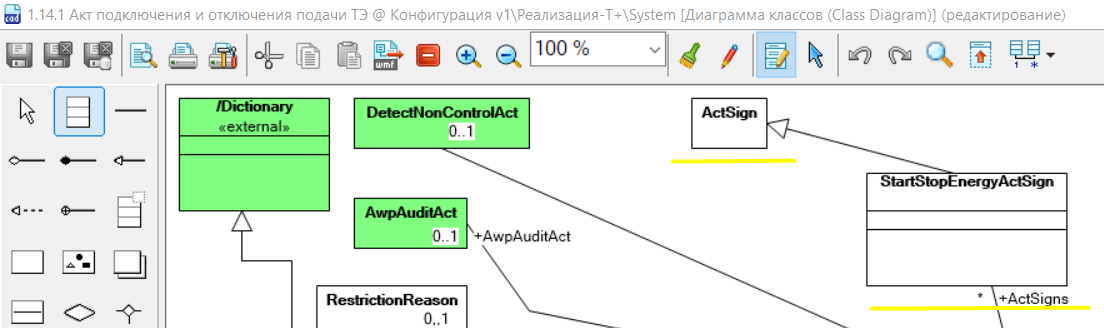
***Рисунок 8. Пример несоответствия в программном коде (Backend)***

## 2.3 Внесение исправлений в диаграмму с созданием новых классов

Некоторые изменения, в силу их основательности, оказалось невозможным решить путем модификации уже существующих классов. В частности, это произошло из-за замены в продуктовом коде базового класса на класс-наследник (не существующий в диаграмме) для уменьшения нагрузки на базу данных (кратко говоря, было произведено разбиение одной таблицы на несколько таблиц).

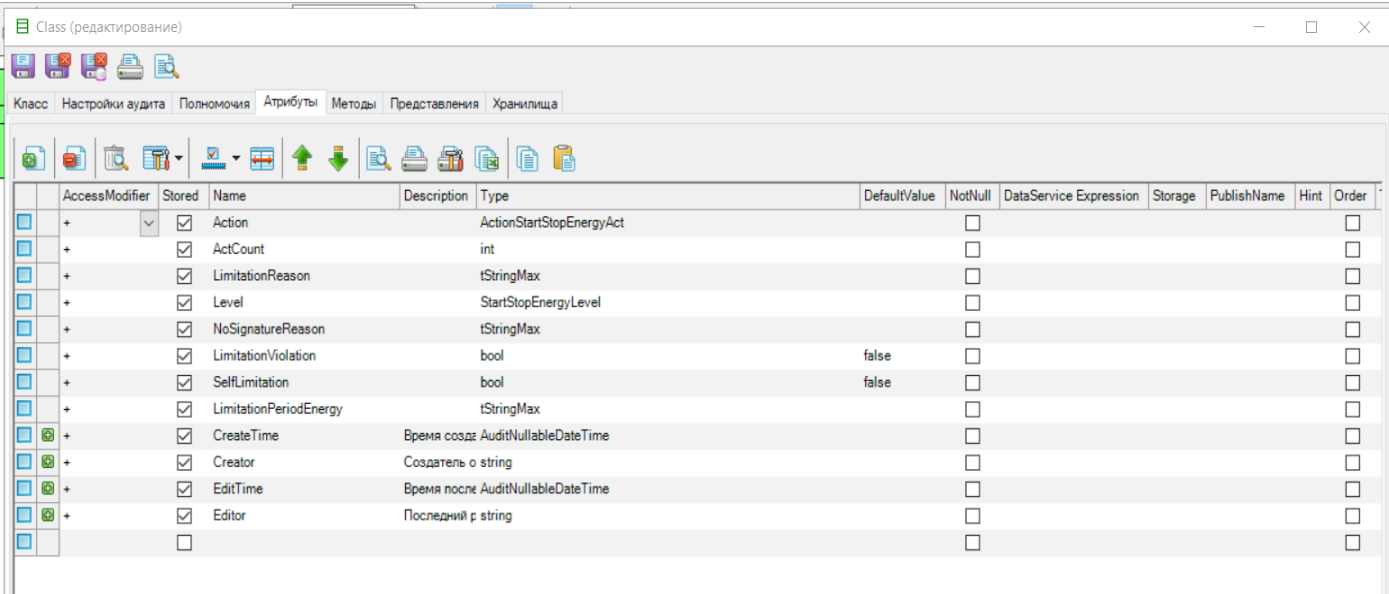
Таким образом, требовалось создать новый класс, делается это способом, похожим на аналогичное взаимодействие в visio или draw.io (рис. 9).

***Рисунок 9. Пример создания новых классов***



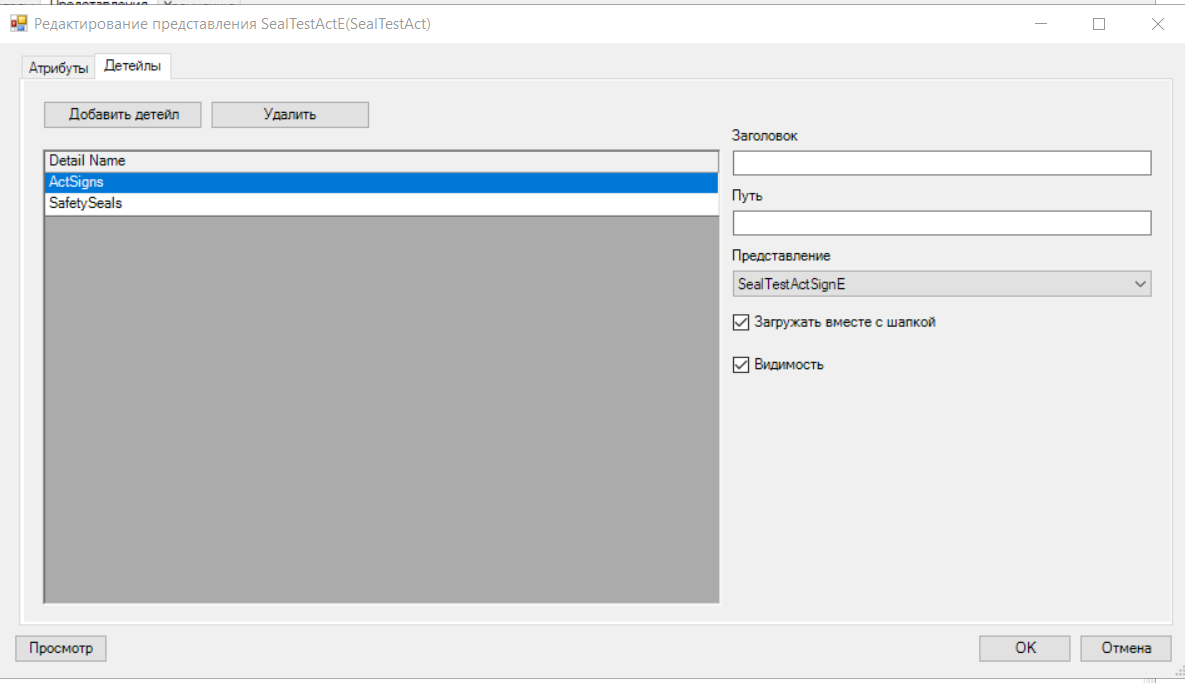
Наполнение данных классов производилось с чистого листа с помощью редактора представлений. В ряде случаев требовалось создавать новые поля, так как они отсутствовали в родительских классах. Для этого в системе Flexberry Designer имеется соответствующая вкладка (см. рис. 10).

Встроенный в систему редактор атрибутов позволяет их удалять, изменять, добавлять. Атрибутам можно добавлять описание, задавать тип и стандартное значение.



***Рисунок 10. Окно добавления атрибутов***

Ранее было рассмотрено добавление атрибутов и связей «один-к-одному», для организации связи «один-ко-многим» и «многие-ко-многим» используется вкладка «детейлы» (рис. 11). Здесь можно выбрать требуемое представление компонента, задать его имя.



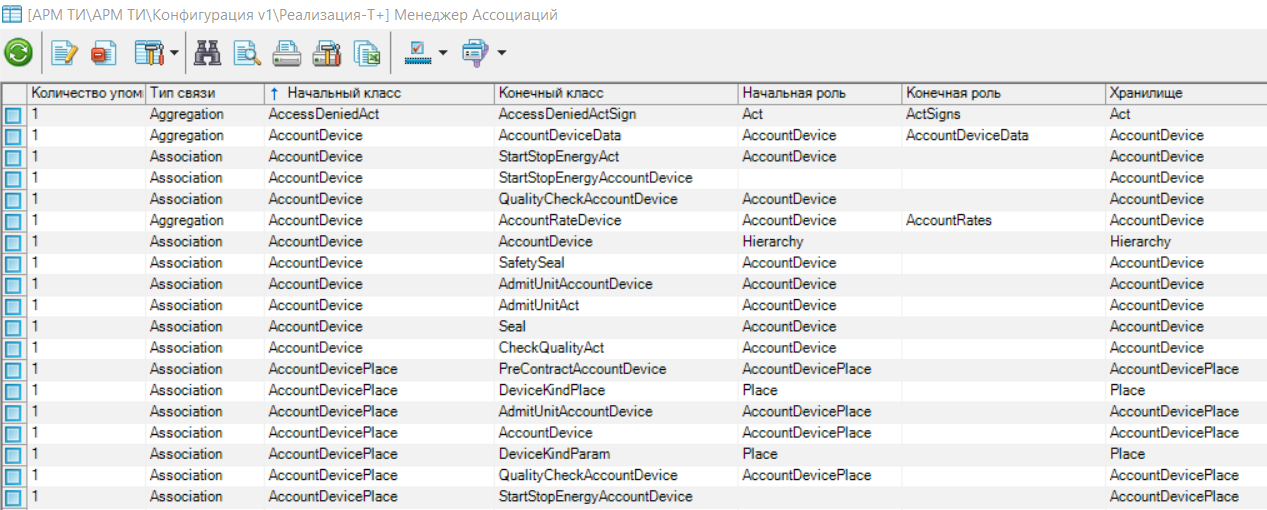
***Рисунок 11. Окно добавления детейлов***

## 2.4 Использование продвинутых возможностей Flexberry Designer.

Самым сложным случаем, возникшим во время выполнения производственной практики, стало исправление ошибок, возникавших в результате сбоев в запросах к базе данных. Данный тип ошибок является критичным, поскольку препятствует сохранению результатов работы с диаграммой. В ходе консультации с руководителем практики от предприятия и ведущим техническим экспертом компании было выяснено, что проблема является следствием так называемых «фантомных связей» (данный термин был употреблен техническим экспертом), заключающихся в лишних и неучтенных связях, возникающих в результате сбоев работы инструмента.

Во избежание порчи данных было предпринято создание резервной копии. Данная потребность полноценно реализуется встроенными средствами, однако воспользоваться резервной копией в итоге не пришлось.

Для проверки употреблений и связей классов воспользуемся встроенной утилитой инструмента Flexberry. Выделим проект «Т Плюс», перейдем по контекстному меню в менеджер ассоциаций (рис. 12).



***Рисунок 12. Окно утилиты «Менеджер ассоциаций»***

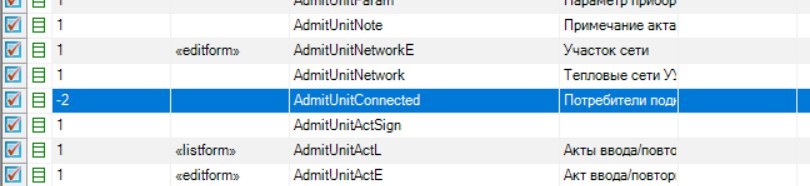
Нажмем на иконку «бинокля» в «шапке» инструмента, введем в поле «что искать» название класса (в нашем случае «AdmitUnit»), с помощью нажатия на кнопку пометить выделим все употребления этого класса (см. рис. 13).

Совершим аналогичное действие в менеджере классов. Введем в поиск то же самое название класса. Найдем случай ошибки в базе данных (на рисунке 14). В выделенной строке число в первой колонке показывает число употреблений данного класса, очевидно, что оно не может быть отрицательным. Данное значение было исправлено вручную.



***Рисунок 13. Пример работы с менеджером ассоциаций***

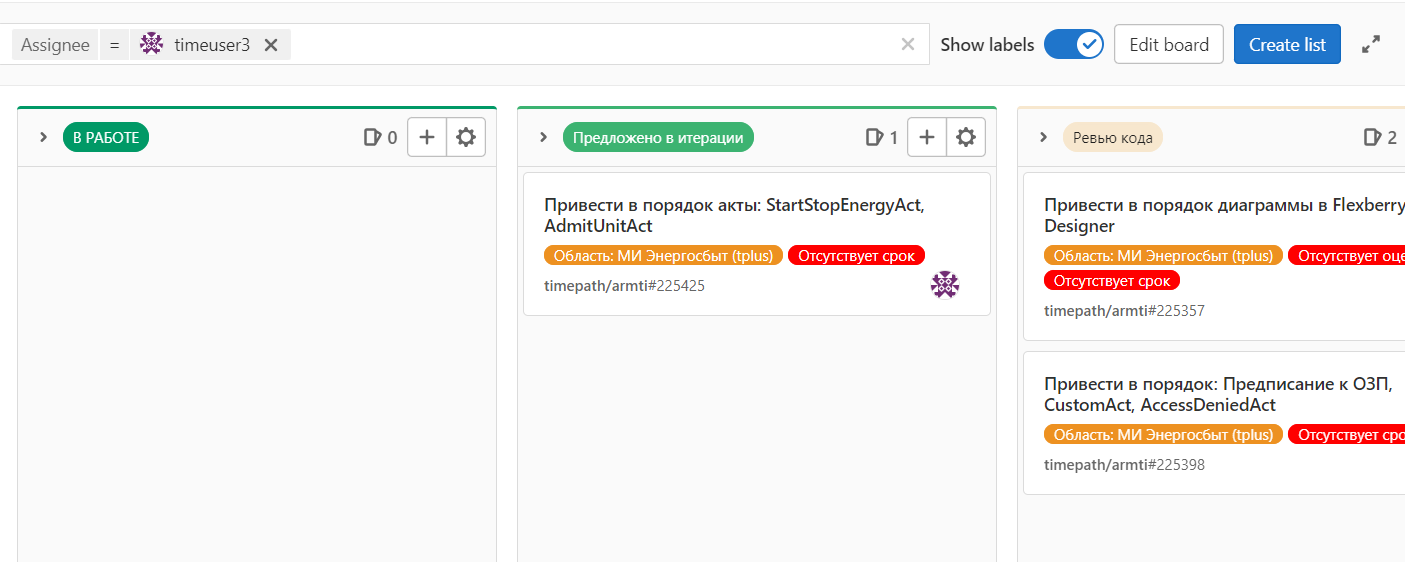
Исправление данной ошибки позволило продолжить работы над диаграммами, поскольку проблема с ее обновлением была решена.



***Рисунок 14. Пример ошибки в количестве употреблений класса***

## 2.5 Работа с системой управления жизненным циклом

Принятие, отслеживание и проверка задач в компании, как уже было сказано, происходит с помощью корпоративного сервера с системой GitLab. (Демонстрация работы с сервисом на рисунке 15, приведено состояние задач на начало середины предпоследней рабочей недели.)



***Рисунок 15. Окно с задачами в системе GitLab***

Изменение состояния задачи происходит путем ее перемещения указателем мыши по соответствующим панелям. При нажатии на задачу открывается окно с ее подробным объяснением.

Для контроля версий программного обеспечения на предприятии также используется GitLab. В ходе выполнения практики использовалось и взаимодействие посредством командной строки, и с помощью графического интерфейса инструментов разработки.

## 2.6 Поездка на конференцию. Публикация научной статьи

Не последним событием, произошедшим во время практики, стала поездка на конференцию SYRCoSE 2023 с представлением доклада на ней. Программная разработка, представленная на конференции, была инициирована руководителем от предприятия еще до прохождения производственной практики (в рамках курсовой работы), однако презентована была во время ее прохождения.

Конференция проводилась в Пензенском государственном технологическом университете. Рассматривались проблемы управления жизненным циклом программного обеспечения и формальных методов, в частности в рамках построения моделей программного обеспечения. Основной проблемой разработки (как было высказано доктором Д. Козновым) на основе моделей была названа их «поддерживаемость» (перевод английского термина maintainability). Зачастую разработчики попросту забывают вносить все изменения, что подтверждается и опытом производственной практики.

Не менее важной проблемой является и избыточность сопутствующей базы компонентов (как в виде программного кода, так и в виде диаграмм). Зачастую новые компоненты создаются путем дублирования старых, что также подтвердилось практическим опытом.

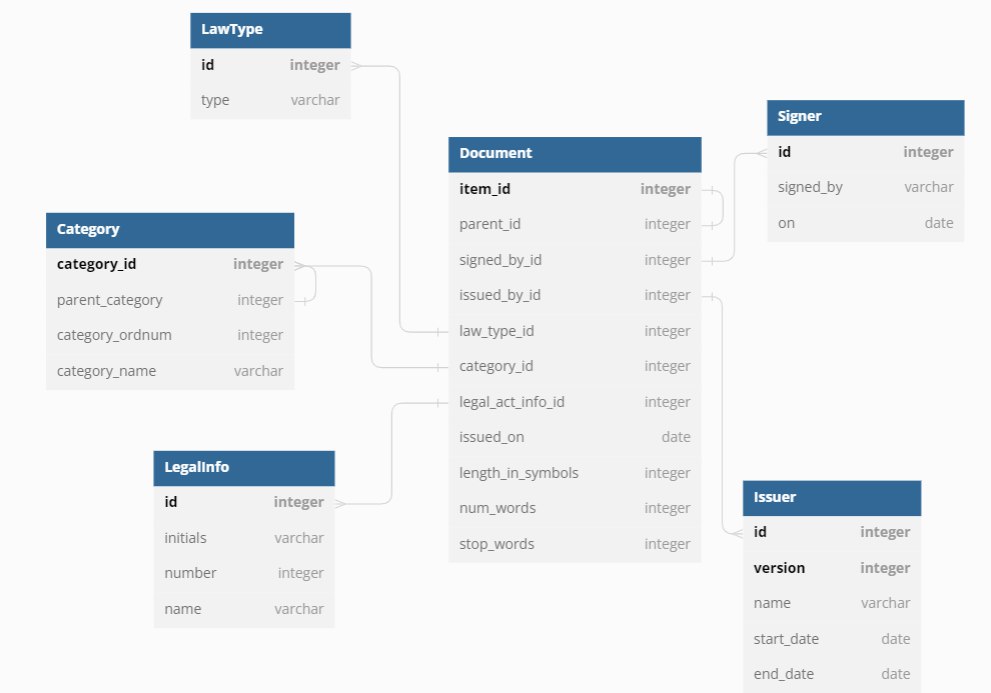
Следует отметить показавшийся теоретическим доклад о паттернах «Домино» в регулярных выражениях и сопутствующим атакам на них. Кратко объясняя данную проблему, следует обратиться к теории автоматов, на которых базируется обработка регулярных выражений. В ряде случаев одно и то же предложение может неоднозначно распознаваться одним и тем же паттерном, особенно в случае повторов. Простым примером «домино» является слово «ababa», которое может быть распознано регулярным выражением «(ab)\*a(ba)\*» и как «ab a ba» (пробелы в этом и следующем проблеме поставлен для наглядности разбиения)   
и как «ab ab a».

Следует отметить, что паттерны «Домино» встречаются и в продуктовом программном коде, что может приводить к ухудшению производительности и сбоям программного обеспечения в силу возрастания времени на обработку слова (или предложения) в силу конструктивных особенностей ряда обработчиков регулярных выражений.

Представлен и доклад, написанный по опубликованной статье [7], созданной в продолжении работы над разработкой, созданной в рамках выполнения курсовой работы. Во время обсуждения доклада присутствовавшими экспертами было предложено внести улучшение для хранения закрытых для публичного просмотра статей нормативно-правовых актов. Данное улучшение можно внести путем частичной модификации базы данных, путем внесения дополнительного поля и разграничения прав пользователей системы.

Было предложено проанализировать возможность реализации классификации нормативно-правового акта путем анализа ссылок на ранее принятые документы. Для анализа применимости данного подхода начато проектирование и сбор аналитического хранилища данных (рис. 16). В целом, идея разработки была оценена положительно.

***Рисунок 16. Схема разработанного хранилища данных***

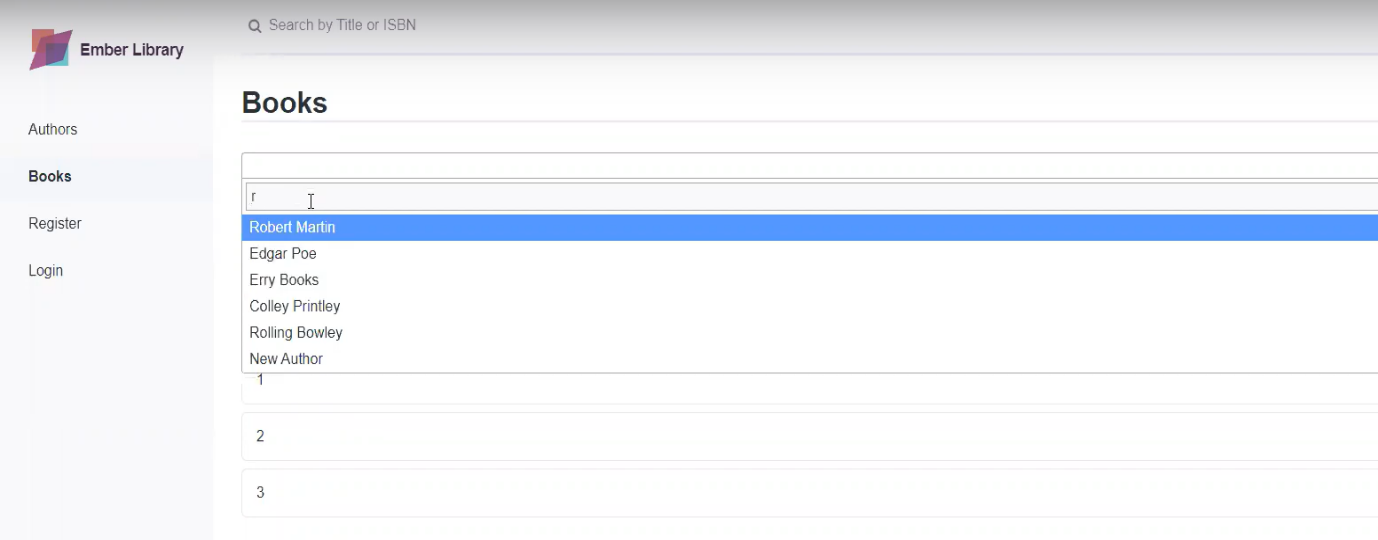


## 2.7 Изучение возможностей фреймворка Ember.js

Для введения нас в возможности фреймворка Ember.js была выделена последняя неделя производственной практики. Обучение велось по курсу Е.А. Хаберева [8], на момент окончания практики было освоено первые два модуля (не считая нулевого).

В ходе выполнения заданий была изучены основные аспекты архитектуры одностраничных (SPA) и многостраничных (MPA) приложений, установка и базовое взаимодействие с json-сервером, использование хостинга GitHub Pages для размещения приложений, применение библиотеки bootstrap для быстрой разработки веб-приложений.

На рисунке 17 можно увидеть окно разработанного по видео-уроку прототипа веб-приложения электронной библиотеки. Более подробно ознакомиться с основным программным кодом можно в Приложении А.



***Рисунок 17. Схема разработанного хранилища данных***

# Глава 3. Выводы и предложения по улучшению

К сожалению, платформа Flexberry Designer является не самой простой в установке. Она рассчитана на употребление разработчиками, имеющими высокую техническую экспертизу, требует предварительной установки и настройки менеджера пакетов yarn и других программных компонентов. Процесс подготовки к работе с платформой занял около 5 часов работы с наставником (сотрудником предприятия). Сказалась нехватка обучающих видео или статей, зависимость платформы на внешние компоненты. Решением проблемы может стать расширение набора обучающих статей или составление руководства программиста. Альтернативным решением видится создание инсталлятора или Docker-образа.

Еще одним недостатком системы является отсутствие в ней статического анализатора, с помощью которого можно было бы предотвратить появление банальных ошибок, таких как использование букв другого алфавита (русского вместо английского и наоборот) в названиях переменных, диаграмм и опечаток, вкупе приводящих к ошибкам генерации, ошибкам работы программы и длительной отладке.

Сама генерация программного кода является достаточно сложным процессом, на данный момент занимающим как минимум полчаса рабочего времени, поскольку происходит полная генерация. Однако процесс можно значительно ускорить, если ввести возможность генерации только измененных диаграмм.

Необходимость решения поставленной нам задачи является своего рода анти-примером Model First (т.е. разработка по разработанной модели) подхода к разработке программного обеспечения. Разумеется, требования заказчика могут меняться, однако зачастую на предприятии их удовлетворение происходит только в продуктовом программном коде и игнорируется модификация модели, что в конечном итоге приводит к накоплению неучтенных изменений и бесполезности моделей.

Более того, сама система построена на дублировании программного кода, таким образом, ухудшается качество и читаемость программного кода, приводит к неучтенным изменениям.

Системы, разрабатываемые компанией, отличаются монолитностью, вследствие чего требуют значительных ресурсов (временных и технических) для компиляции, отладки, тестирования, введения в эксплуатацию. Как стало известно, на предприятии рассматривается вопрос перехода к более гибкой (так называемой «микросервисной») архитектуре, обновления средств разработки (в том числе переход с языка программирования JavaScript на язык программирования TypeScript). Однако следует отметить, что использование языка программирования TypeScript потребует обновления платформы Flexberry Designer.

Улучшить качество работы может использование статического анализатора программного кода. Это может сократить количество ошибок, допущенных по невнимательности разработчиков (неправильное написание терминов, смешение символов разных языков). В рамках импортозамещения рассматривается замена системы управления базами данных (СУБД) Microsoft SQL Server на аналогичные решения от Postgre. Однако смена средств разработки и программного обеспечения требует повышения квалификации коллектива, дополнительные трудовые факторы производства. Видится наиболее практичным обновление технологий в рамках новых проектах, оставляя старые в имеющемся виде.По итогам работы на предприятии были выполнены все поставленные задачи, получен положительный отзыв от руководителя практики от предприятия и прочих сотрудников компании, ассистировавших мне и консультировавших меня при решении наиболее сложных заданий. Следует отметить, что технологии, освоенные во время практики, являются уникальными и ранее не рассматривались мной в контексте прикладной разработки.

За время прохождения производственной практики было приведено в надлежащий вид восемь диаграмм классов, внесен ряд незначительных правок и в другие. На данный момент данные диаграммы приведены в актуальное состояние. Таким образом, были актуализированы навыки построения диаграмм в нотации UML, навыки проектирования архитектуры программных систем.

В ходе поездки на конференцию получен опыт представления исследовательских работ, подготовлен задел, который может быть использован для написания выпускной квалификационной работы, получены рекомендации по развитию разработки.

На последней неделе разработке было произведено краткое знакомство с фреймворком Ember.js, изучено его основные концепции и способы применения в промышленной разработке.

Во время прохождения производственной практики была подготовлена отчетная документация. Ее разработка велась постепенно на протяжении всей практики и была доработана на завершающем этапе производственной практики.

# Заключение

Подводя итог, следует отметить выполнение индивидуального задания на производственную практику и развитие компетенций, указанных в программе практике. Поскольку цели практики имеют критерии их достижения, перечислим их.

Так, в ходе работы на предприятии были улучшены навыки применения систем контроль версий в корпоративной разработке (в частности, Git), систем контроля жизненного цикла программного обеспечения (главным образом, возможности платформы GitLab). Получен опыт продуктовой разработки в удаленной команде и участия в Scrum-ритуалах. В ходе практики, таким образом, были актуализированы компетенции, полученные во время освоения дисциплин «Групповая динамика и коммуникации в профессиональной практике программной инженерии», «Введение в программную инженерию», «Проектный семинар».

При выполнении задач, непосредственно связанных с рабочим процессом, были актуализированы знания, полученные в ходе изучения дисциплины «Программирование» (чтение, отладка и исправление программного кода), «Конструирование программного обеспечения» (использование CASE-инструментариев, чтение и редактирование UML-диаграмм).

Во время настройки рабочего окружения были актуализированы знания, полученные на дисциплинах «Веб-программирование» (установка node.js, ember.js и необходимых программных пакетов) и «Операционные системы» (использование консоли для управления выполняющимися процессами).

Свой опыт работы на предприятии и свои результаты оцениваю положительно, был выполнен ряд заданий, имевших для предприятия непосредственную важность (например, приведение в требуемое состояние UML-диаграмм для последующей генерации по ним программного кода).

Презентована ранее разработанная система на конференции SYRCoSE 2023, таким образом, получен опыт публичной защиты проекта перед неизвестной аудиторией и написания исследовательских статей. Также освоены возможности фреймворка ember.js, ранее мною не использовавшегося.

# Список литературы

1. Траектория времени: [сайт]. 2022.  
   URL: <https://timepath.ru/> (дата обращения 17.06.2023);
2. О Траектории // Траектория времени: [сайт]. 2022.  
   URL: https://timepath.ru/about.html (дата обращения 17.06.2023);
3. Памятка для разработчиков // Корпоративный сервер GitLab компании «Траектория времении». 2021.  
   URL: https://gitlab.timepath.ru/timepath/timepath/-/wikis/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0/%D0%9F%D0%B0%D0%BC%D1%8F%D1%82%D0%BA%D0%B0-%D0%B4%D0%BB%D1%8F-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA%D0%BE%D0%B2#%D0%BA%D0%B0%D0%BA-%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE-%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D1%8F%D1%82%D1%8C-%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D1%83-%D0%B2-%D1%80%D0%B0%D0%B1%D0%BE%D1%87%D0%B5%D0%BC-%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%B5 (дата обращения 17.06.2023);
4. Путеводитель для нового сотрудника разработчиков // Корпоративный сервер GitLab компании «Траектория времении». 2021.  
   URL: <https://gitlab.timepath.ru/timepath/timepath/-/wikis/%D0%9E%D0%B1%D1%89%D0%B5%D0%B5/%D0%9F%D1%83%D1%82%D0%B5%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8C-%D0%B4%D0%BB%D1%8F-%D0%BD%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE-%D1%81%D0%BE%D1%82%D1%80%D1%83%D0%B4%D0%BD%D0%B8%D0%BA%D0%B0#vpn-%D0%B> (дата обращения 17.06.2023);
5. Памятка по лейблам задач // Корпоративный сервер GitLab компании «Траектория времении». 2021.  
   URL: https://gitlab.timepath.ru/timepath/timepath/-/wikis/%D0%9E%D0%B1%D1%89%D0%B5%D0%B5/%D0%9F%D0%B0%D0%BC%D1%8F%D1%82%D0%BA%D0%B0-%D0%BF%D0%BE-%D0%BB%D0%B5%D0%B9%D0%B1%D0%BB%D0%B0%D0%BC-%D0%B7%D0%B0%D0%B4%D0%B0%D1%87 (дата обращения 17.06.2023);
6. Flexberry Designer // Flexberry Platform: [официальный сайт].  
   URL: https://flexberry.github.io/ru/fd\_flexberry-designer.html
7. *Насу Ю., Ланин В.В*. Разработка системы классификации нормативно-правовых документов на основе метода опорных векторов. *Труды Института системного программирования РАН*. 2023;35(2):49-56. <https://doi.org/10.15514/ISPRAS-2023-35(2)-4>
8. *Хаберев Е.А.* Программа обучения Flexberry Ember.  
   URL: <https://docs.google.com/document/d/1NuJf9AfKO-zHHl2zeL9NlngB9lxZZLoYv12wObVybZg/edit> (дата обращения 17.06.2023).

# Приложение А

import { computed } from '@ember/object';

import DS from 'ember-data';

export default DS.Model.extend({

firstName: DS.attr('string'),

lastName: DS.attr('string'),

books: DS.hasMany('book'),

fullName: computed('firstName', 'lastName', function() {

return `${this.get('firstName')} ${this.get('lastName')}`;

}),

});

import DS from 'ember-data';

export default DS.Model.extend({

title: DS.attr('string'),

isbn: DS.attr('string'),

publishDate: DS.attr('date-string'),

author: DS.belongsTo('author'),

reviews: DS.hasMany('review'),

user: DS.belongsTo('user')

});

import DS from 'ember-data';

export default DS.Model.extend({

user: DS.attr('string'),

body: DS.attr('string'),

createdAt: DS.attr('date-string'),

book: DS.belongsTo('book')

});

import DS from 'ember-data';

export default DS.Model.extend({

email: DS.attr('string'),

username: DS.attr('string'),

password: DS.attr(),

passwordConfirmation: DS.attr()

});

import Controller from '@ember/controller';

export default Controller.extend({

queryParams: ["search"],

search: ''

});

import Controller from "@ember/controller";

import { inject as service } from '@ember/service';

export default Controller.extend({

session: service(),

currentUser: service(),

actions: {

async logout(e) {

e.preventDefault();

this.get('session').invalidate();

}

}

});

import { computed } from '@ember/object';

import Controller from '@ember/controller';

export const PER\_PAGE = 2;

export default Controller.extend({

queryParams: ['search', 'page', 'author'],

search: '',

page: 1,

author: '',

pages: computed('model.books.meta.total', function() {

const total = Number(this.get('model.books.meta.total'));

if (Number.isNaN(total) || total <= 0) {

return [];

}

return new Array(Math.ceil(total / PER\_PAGE))

.fill()

.map((value, index) => index + 1);

}),

selectedAuthor: computed('author', function() {

const author = this.get('author');

return author ? this.get('model.authors').findBy('id', author) : null;

}),

actions: {

changeAuthor(author) {

this.set('author', author ? author.get('id') : '');

}

}

});

import Controller from '@ember/controller';

import { inject as service } from '@ember/service';

export default Controller.extend({

session: service(),

actions: {

async login(user) {

try {

await this.get('session').authenticate('authenticator:jwt', {

email: user.email,

password: user.password

});

}

catch(e) {

this.send('error', e);

}

},

error(error, transition) {

if (error instanceof Error) {

return true;

}

this.set('errors', error.json.errors);

return false;

}

},

resetErrors() {

this.set('errors', {});

}

});

import Controller from '@ember/controller';

export default Controller.extend({

actions: {

async saveUser(user) {

let newUser;

try {

newUser = this.get('store').createRecord('user', user);

await newUser.save();

this.transitionToRoute('index');

}

catch(e) {

e.user = newUser;

this.send('error', e);

}

},

error(error, transition) {

this.set('errors', error.user.errors);

return false;

}

},

resetErrors() {

this.set('errors', {});

}

});

import Component from '@ember/component';

export default Component.extend({

actions: {

submitForm(e) {

e.preventDefault();

this.onsubmit({

id: this.get('idAuthor'),

firstName: this.get('firstName'),

lastName: this.get('lastName')

});

}

},

didReceiveAttrs() {

this.\_super(...arguments);

// this.set('firstName', this.get('author.firstName'));

// this.set('lastName', this.get('author.lastName'));

this.setProperties({

idAuthor: this.get('author.id') ? this.get('author.id') : undefined,

firstName: this.get('author.firstName'),

lastName: this.get('author.lastName')

});

},

});

import Component from '@ember/component';

import { inject as service } from '@ember/service';

export default Component.extend({

store: service(),

currentUser: service(),

actions: {

async saveBook(e) {

e.preventDefault();

this.get('onSubmit')({

title: this.get('title'),

isbn: this.get('isbn'),

publishDate: this.get('publishDate'),

author: this.get('author'),

user: this.get('currentUser.user')

});

},

searchAuthor(query) {

return this.get('store').query('author', { q: query })

}

},

didReceiveAttrs() {

this.setProperties({

title: this.get('book.title'),

isbn: this.get('book.isbn'),

publishDate: this.get('book.publishDate'),

author: this.get('book.author')

});

},

});

import Component from '@ember/component';

export default Component.extend({

actions: {

login(e) {

e.preventDefault();

this.get('onSubmit')({

email: this.email,

password: this.password

});

}

},

didReceiveAttrs() {

this.setProperties({

email: this.get('user.email'),

password: this.get('user.password')

});

}

});

import Component from '@ember/component';

import fetch from 'fetch';

import ENV from 'books-demo/config/environment';

export default Component.extend({

iAmRobot: true,

reset: false,

actions: {

async saveUser(e) {

e.preventDefault();

this.get('onSubmit')({

email: this.email,

username: this.username,

password: this.password,

passwordConfirmation: this.passwordConfirmation

});

},

async verified(key) {

try {

const { success } = await (await fetch(`${ENV.backendURL}/recaptcha?key=${key}`)).json();

this.set('iAmRobot', !success);

} catch (error) {

this.set('reset', true);

}

},

expired() {

this.set('iAmRobot', true);

}

},

didReceiveAttrs() {

this.setProperties({

email: this.get('user.email'),

username: this.get('user.username'),

password: this.get('user.password'),

passwordConfirmation: this.get('user.passwordConfirmation'),

});

}

});

1. Например, команда для установки Ember.js: «npm install -g ember-cli@2.4.3». [↑](#footnote-ref-1)
2. К примеру, команда конфигурирования ssl-политики: «yarn config set strict-ssl false –global». [↑](#footnote-ref-2)