Задание №2 по курсу
«Проектирование Программных Систем»
Defect Prediction (облачный CI/CD)

Выполнили:
Калошин Павел, 491а
Болотин Петр, 491а
Бобровская Наталия, 491б

1. Постановка задачи

1.1. Состав второго задания

- Согласование с семинаристом списка классов платформы реализации
- Реализация модели на выбранной платформе
- Предоставить описание проекта, описания и обязанности основных сущностей (классы, интерфейсы и т.п.), основные сценарии работы с продуктом
- Модель реализации (диаграммы вариантов использования, диаграммы классов)

- Интерпретация ключевых взаимодействий/конечных автоматов динамической модели (2-3 штуки)
- Обоснование применения шаблонов проектирования и других проектировочных решений

1.2. Описание задачи

Планируется реализовать приложение для облачных систем CI/CD, которое будет нести в себе **инструмент расчета метрик и общаться с интерфейсом сервера. Сервер имеет метод** для предсказания, в каких модулях проекта вероятнее всего появятся ошибки к следующему его релизу (т.е. стабильной версии проекта). Поскольку пока что рассматриваются только проекты, написанные на java, модулем системы считаем отдельный класс. Приложение должно интегрироваться в процесс сборки/тестирования, считать метрики по модулям кода, **передавать их на сервер**, получать предсказания и выводить их в лог.

Также необходимо реализовать интерфейс поддержки этого приложения. Мы хотим иметь возможность дообучать модель на данных, собранных с открытых репозиториев и затем интегрировать ее в приложение. Для этих целей у нас должен быть **инструмент разметки**, позволяющий автоматически выкачивать репозитории с открытым кодом и формировать из них датасеты для нашей модели, а так же **инструмент дообучения,** позволяющий **обновлять веса ML модели и сохранять их** для дальнейшего использования на сервере при предсказаниях**.**

1.3 Распределение обязанностей

- **Пётр**: проектирование и обоснование архитектуры системы
- **Наталия**: проработка вариантов использования, требований к системе

- **Павел**: реализация прототипа, тестирование.

2. Описание предметной области

2.1. Логическая модель

Приводится последняя версия диаграммы классов на рис.1.

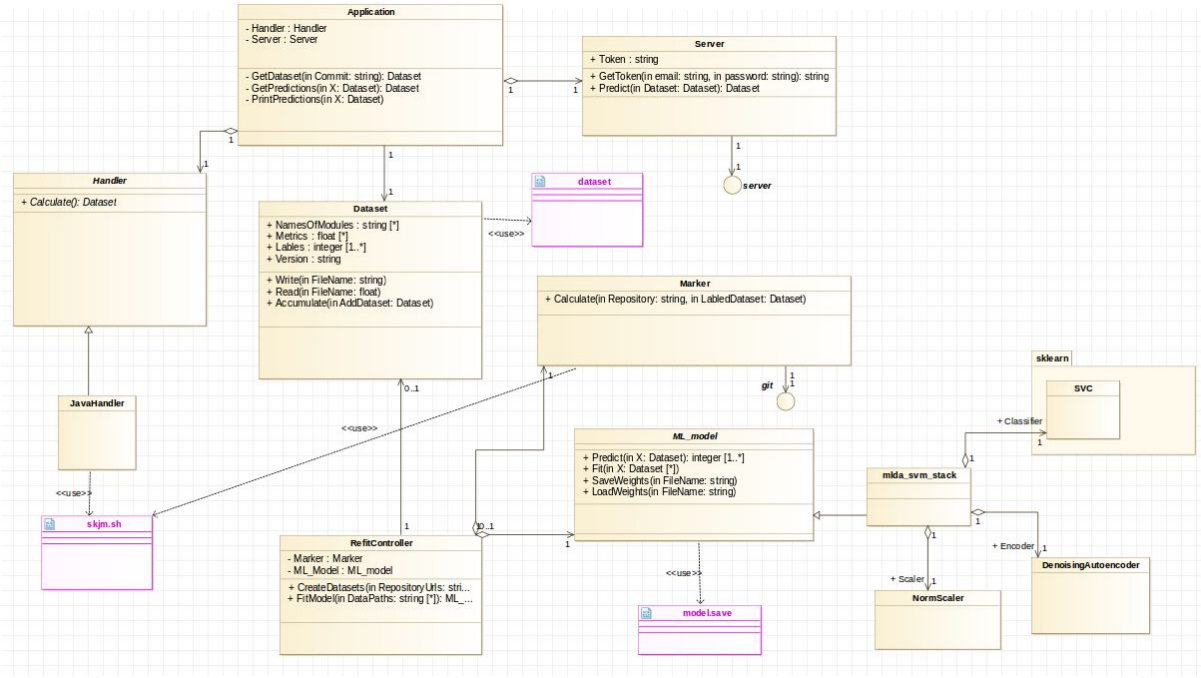


Рис. 1. Диаграмма классов

2.2. Описания и обязанности классов

Привести таблицу с перечнем классов и обязанностями, коллегами. Обязанностей и коллеги приводятся из первого задания, агрегированно по всем взаимодействиям.
Таблица 2. Обязанности и коллеги классов предметной области бронирования билетов.

Класс	**Контракт**	**Коллеги**
Application	Инициализирует и хранит Server, Handler Выполняет действиеgetDataset: предоставляет Dataset Выполняет действиеgetPredictions: изменяет Dataset Выполняет действие printPredictions: использует Dataset, выводит предсказания	Server Handler Dataset

Server	Хранит Token Предоставляет общение с внешним интерфейсом server: выполняет действия getToken, Predict	Application server
Handler	Абстрактный класс: имеет действие Calculate, предоставляет Dataset	Application JavaHandler (наследник)
JavaHandler	Класс-наследник:: выполняет действие Calculate, использует skjm.sh, предоставляет Dataset	Handler (родитель) skjm.sh (файл-скрипт)
Dataset	Хранит, предоставляет, изменяет массивы имен модулей, метрик, меток, версию, использует dataset	Application RefitController dataset (файл)
RefitController	Инициализирует и хранит Marker, ML_model Выполняет действие CreateDatasets: предоставляет Dataset Выполняет действие FitModel: вызывает действие Fit в ML_model	Dataset Marker ML_model
Marker	Выполняет действие Calculate, создает размеченный Dataset, использует skjm.sh	RefitController git (внешний интерфейс) skjm.sh (файл-скрипт)
ML_model	Выполняет действие Predict, использует Dataset Выполняет действие Fit, использует Dataset Выполняет действия SaveWeights, LoadWeights Использует и изменяет сохраненные веса model.save	RefitController model.save (файл) mlda_svn_stack (наследник)
mlda_svn_stack	Хранит детали реализации алгоритма предсказаний: NormScaler, DenoisingAutoencoder, SVC	ML_model NormScaler DenoisingAutoencoder SVC

NormScaler, DenoisingAutoencoder, SVC - детали реализации алгоритма предсказаний.

3. Описание функциональных требований к системе

3.1. Модель вариантов использования

Пользователь хочет узнать, есть ли ошибки в модулях его программы.
Разработчик хочет иметь возможность дообучать модель при поступлении новых данных.

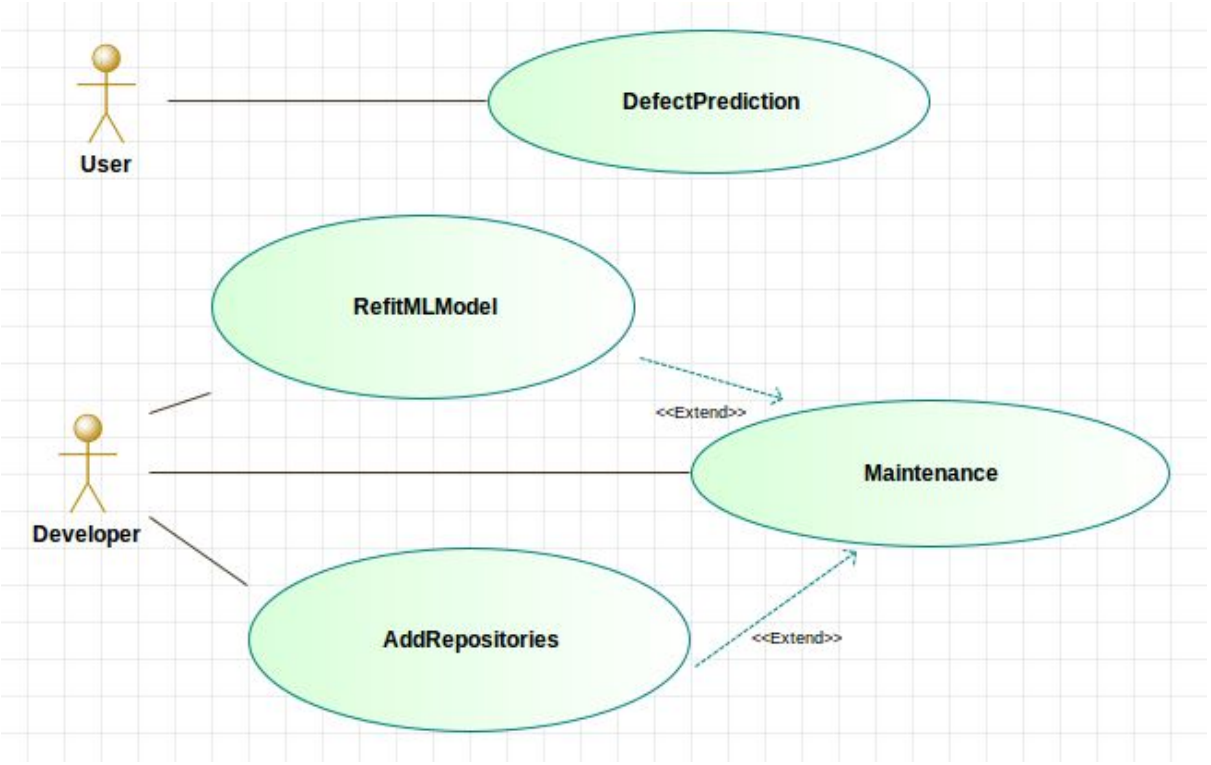


Рис. 2. Диаграмма вариантов использования.

3.2. Вариант использования DefectPrediction

Акторы: User (Пользователь)
Цель: Пользователь хочет получить метки - будет ли содержаться в данном модуле кода ошибка.
Предусловия: Доступность gitlab CI, строгое соответствие версии инструмента разметки той, что лежит на сервере, собранный проект на java.
Постусловия: Пользователь получил метки.
Основной сценарий:

1. Пользователь загружает проект на gitlab.
2. Запускает инструмент Defect Prediction.
3. Система соединяется с сервером при помощи email и пароля, получает токен.
4. Система создает объект класса Handler.
5. Система считает метрики по текущему состоянию проекта при помощи Handler, записывая результаты в Dataset.
6. Система отсылает dataset серверу, получая от него предсказания.
7. Система выводит предсказания в лог.

Альтернативные сценарии:

1. В командной строке не переданы email и пароль: система выводит строку с примером правильного вызова и останавливает приложение.
2. Аутентификация на шаге 3 не удалась: система выводит сообщение и останавливает приложение.
3. Посчитать метрики проекта на шаге 5 не удалось: система выводит сообщение и останавливает приложение.

3.3. Вариант использования Maintenance

Акторы: Developer (Разработчик)
Цель: Обучение ML модели на новых данных
Предусловия: Разработчик имеет новые репозитории для обучения.
Постусловия: Разработчик получает файл с весами для ML модели.
В данном варианте использования предусмотрены два сценария, которые могут быть выполнены последовательно.
Сценарий AddRepositories:

1. Разработчик создает файл с ссылками на репозитории, которые он хочет скачать разметить и добавить в базу данных.
2. Разработчик запускает скрипт, передавая ему в качестве параметра имя этого файла.
3. Скрипт выкачивает репозитории, формирует из них датасеты, размечает и сохраняет в нужном формате.

Альтернативные сценарии:

- обработать репозиторий в п.3 не удалось

При всех альтернативных вариантах отображаем в логе соответствующие ошибки и продолжаем работу.

Сценарий RefitMLModel:

1. Разработчик запускает скрипт.
2. Скрипт обучает модель на всех имеющихся в распоряжении данных.
3. Скрипт сохраняет веса полученной модели в файл.

Альтернативный сценарий:

1. Разработчик создает файл с именами датасетов, на которых нужно обучить модель.
2. Разработчик запускает скрипт, передавая ему файл с именами.
3. Скрипт обучает модель на указанных данных.
4. Скрипт сохраняет веса полученной модели в файл.

4. Описание архитектуры системы

4.1. Описание архитектуры

Каналы и фильтры
Организация обработки потоков данных в том случае, когда процесс обработки распадается на несколько шагов. Эти шаги могут выполняться отдельными обработчиками, возможно, реализуемыми разными разработчиками или даже организациями. При этом нужно принимать во внимание следующие факторы:

Каждая отдельная задача по обработке данных разбивается на несколько мелких шагов. Выходные данные одного шага являются входными для других. Каждый шаг реализуется специальным компонентом — **фильтром (filter)**. Фильтр потребляет и выдает данные инкрементально, небольшими порциями. Передача данных между фильтрами осуществляется по **каналам (pipes)**.

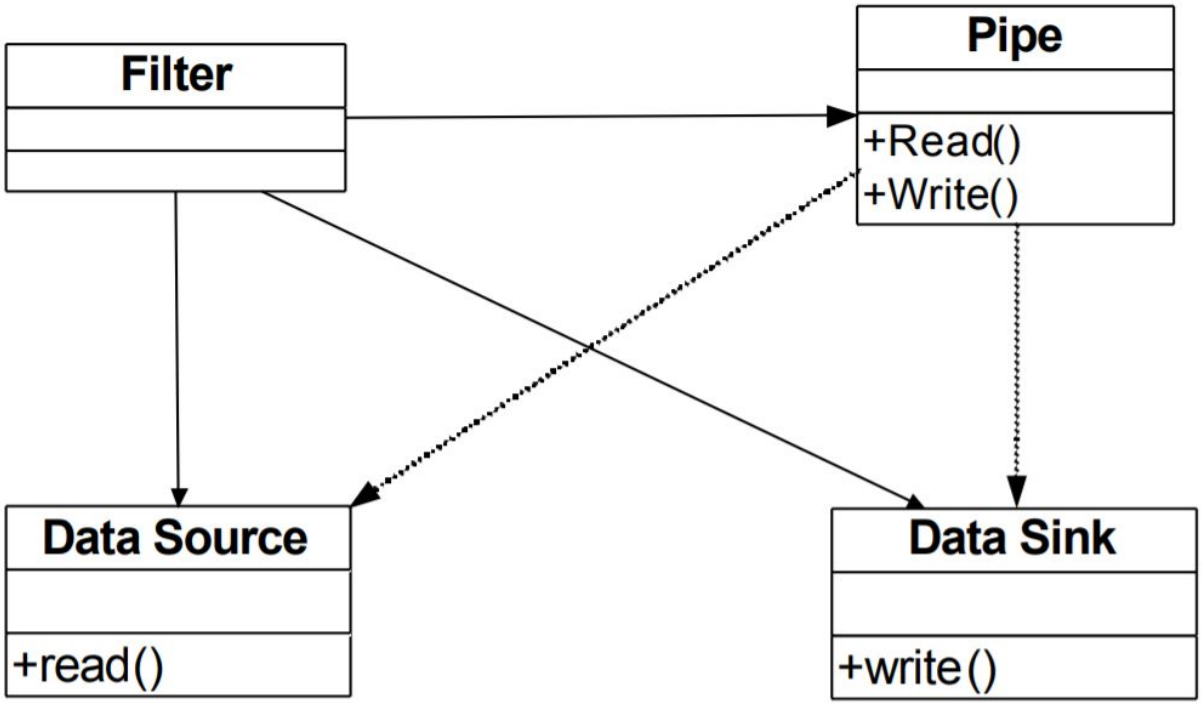
4.2. Обоснование выбора архитектурного стиля

Среди рассмотренных архитектурных стилей:
1. трехуровневая архитектура
2. монолитное приложение
3. двухуровневая архитектура
4. каналы и фильтры

Для разрабатываемого клиента системы предсказания дефектов лучше подходит стиль каналов и фильтров, потому что на стороне клиента есть две независимых компоненты, в каждой из которых нужно совершать последовательные действия
Стиль многоуровневая архитектура не подходит, потому что на стороне клиента нет необходимости выделять несколько слоёв.
Стиль монолитное приложение не подходит потому, что на стороне клиента есть две независимых части системы. Разумно не соединять их в монолитное приложение.

4.3. Описание вспомогательных интерфейсов и классов

Типичная диаграмма классов в случае архитектуры pipes and filters выглядит вот так:



В нашей системе она выглядит вот так:

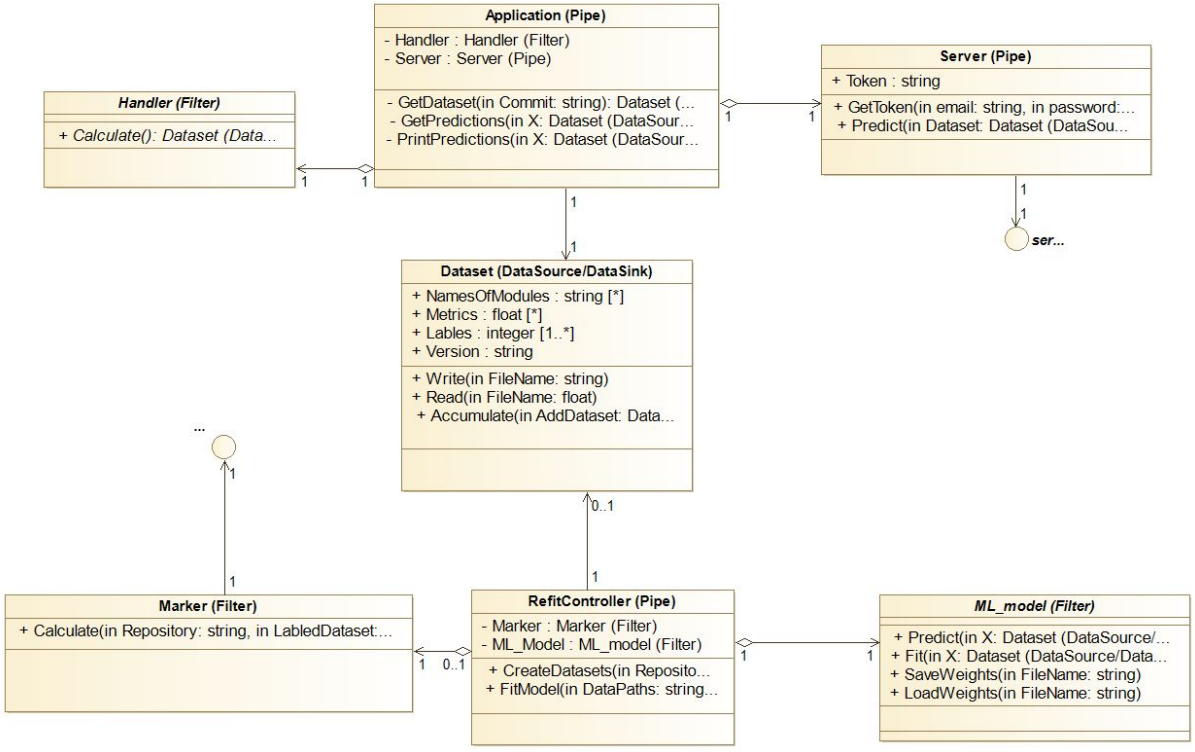


Рис. 3. Описание классов и интерфейсов платформы реализации

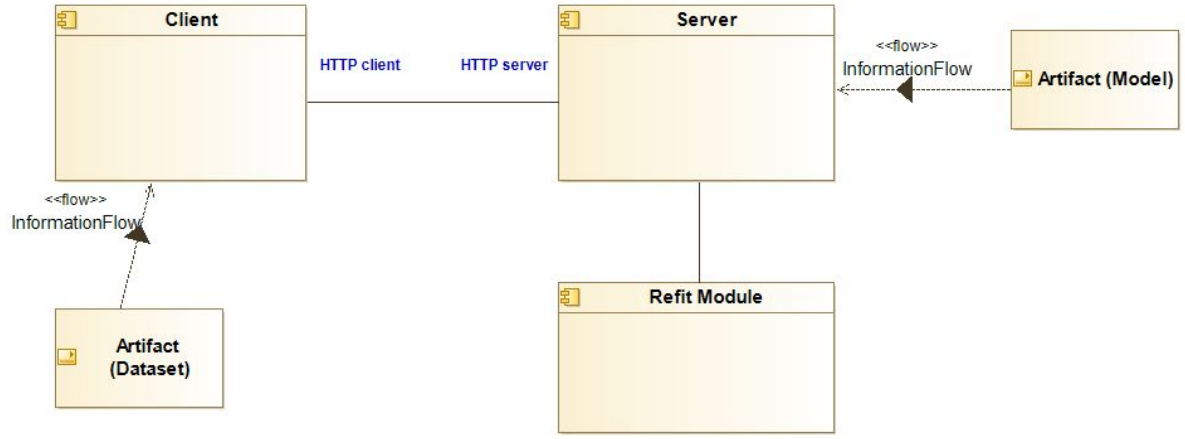
Таблица 2. Обязанности и коллеги классов предметной области бронирования билетов.

Класс	**Контракт**	**Коллеги**
Dataset	Исполняет функции как Data Source, так и Data Sink.	Application (Pipe) Refit Controller (Pipe) Handler (Filter) ML_model (Filter)
ML_model	Filter, получает на вход Dataset и возвращает predictions	Dataset (DataSource/DataSink) Refit_controller (Pipe)
Refit_controller	Pipe Соединяет DataSource/DataSink класс Dataset с фильтрами Marker и ML_model	Dataset (DataSource/DataSink) Marker (Filter) ML_model (Filter)
Marker	Filter. Создаёт датасет по ссылке на репозиторий	Refit_controller (Pipe)
Handler	Filter Считает метрики	Application (Pipe) Dataset (DataSource/DataSink)
Server	Pipe Соединяет Application и настоящий сервер	Application
Application	Pipe Соединяет Handler и Server	Handler Server Dataset

4.4. Структура компонентов системы и решения по реализации архитектуры

Привести диаграмму компонентов согласно архитектурному стилю.

Рис. 3. Структура системы



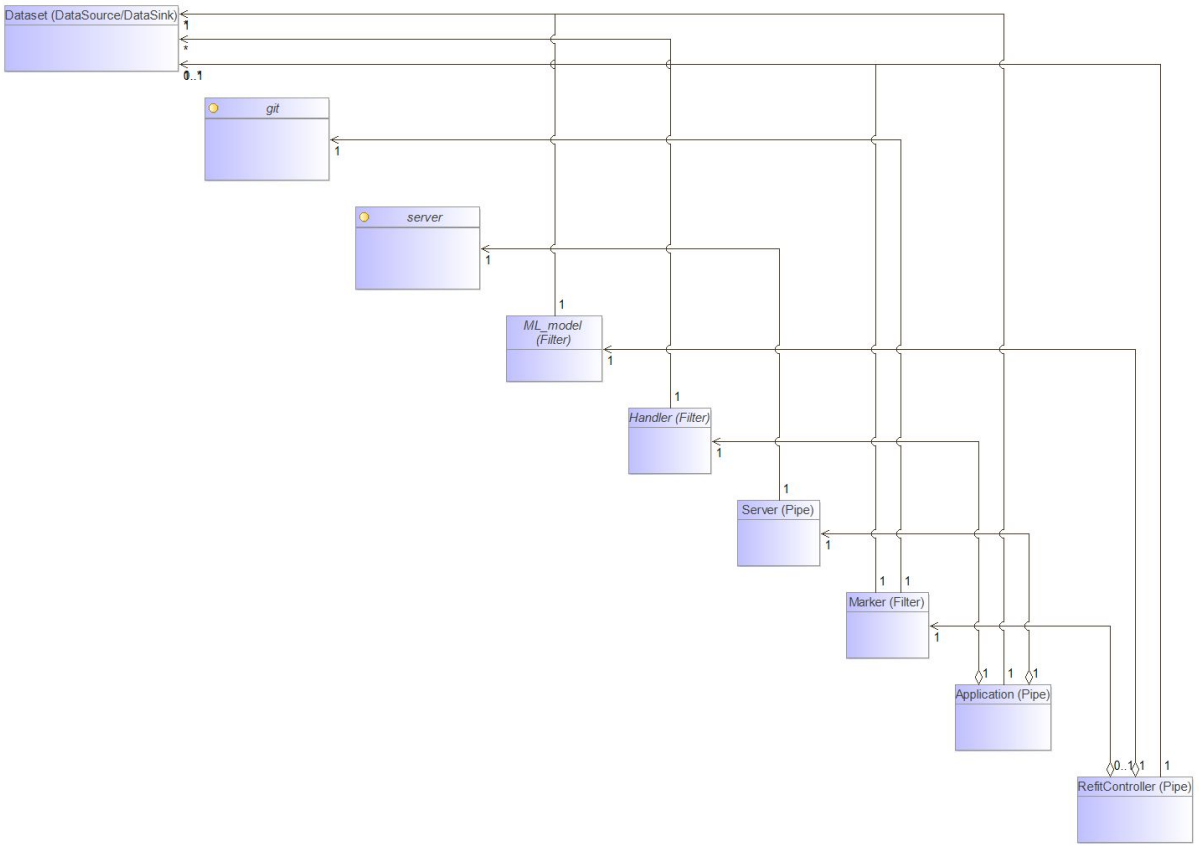
В таблице указать назначение каждого компонента. Заполняется по описанию архитектуры.

Таблица 2. Обязанности и коллеги классов предметной области предсказания дефектов

Компонент	**Назначение**	**Интерфейсы и порты**
Client	Хранится у пользователя, считает метрики и отправляет их серверу	HTTPS
Server	Принимает значения метрик и применяет к ним обученную модель	HTTPS
Refit module	Нужен для дообучения модели.	

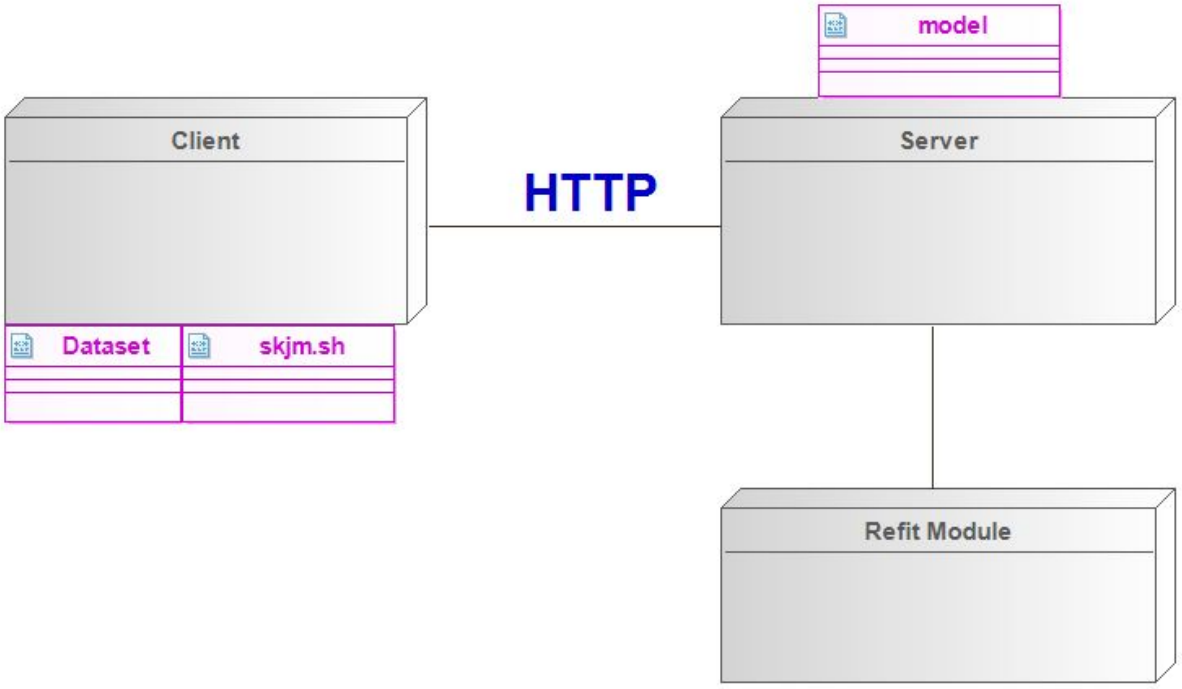
4.5. Структура пакетов

Рис. 4. Структура пакетов модели системы



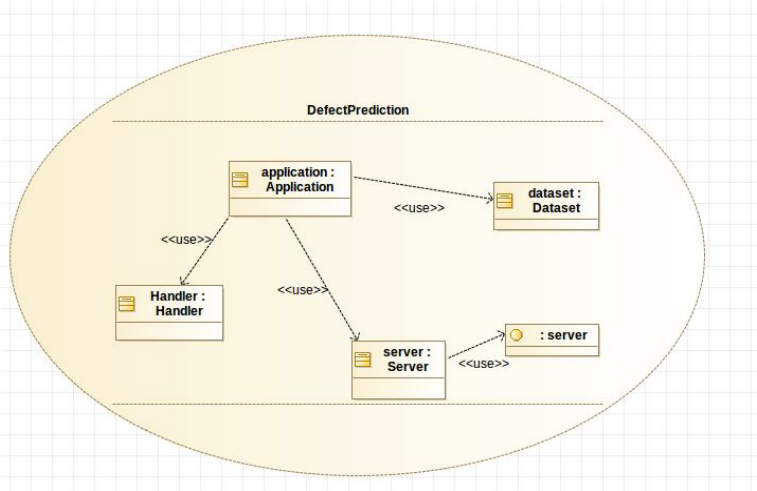
4.6. Размещение компонентов системы

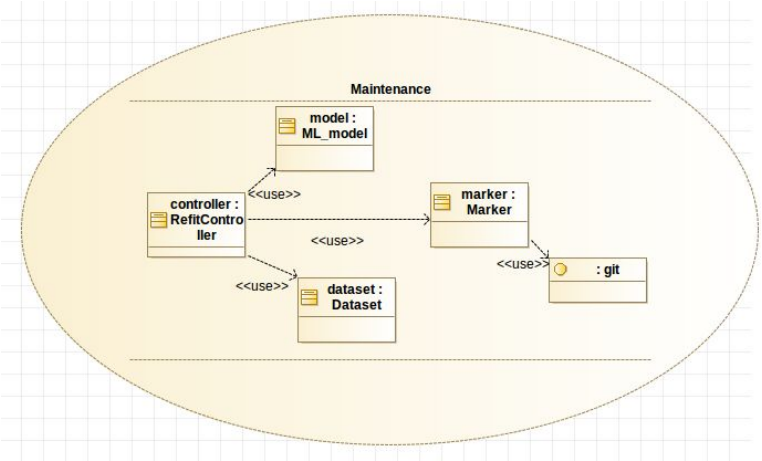
Общение между клиентом и сервером происходит с помощью HTTP, при необходимости на стороне сервера используется дообучение модели.



5. Описание механизмов реализации

Кооперации для двух основных вариантов использования:



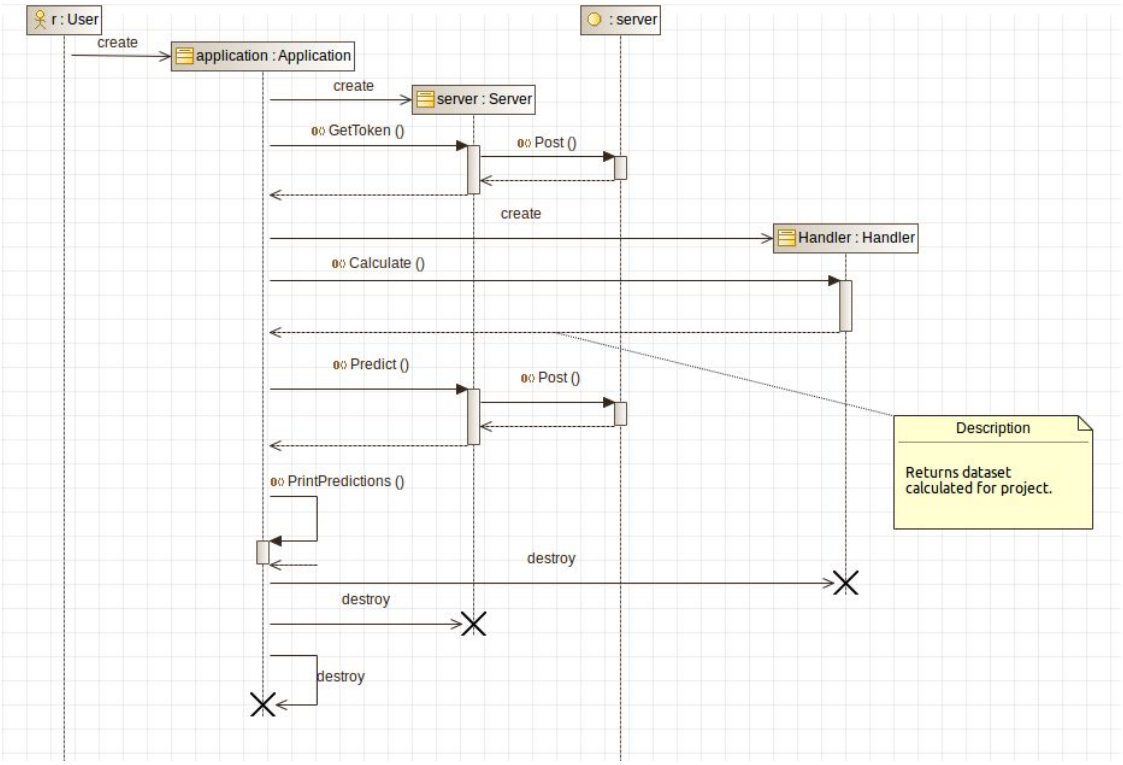


5.1 Реализация варианта использования DefectPrediction

5.1.1 Механизм инициализации

Пользователь запускает скрипт application в процессе CI, передавая ему в качестве параметра токен, полученный при регистрации на сервере. Создается экземпляр класса Application, который создает экземпляры класса Server и Handler. При создании экземпляр класса Server пытается установить соединение с удаленным сервером по известной ссылке через интерфейс, предоставляемый библиотекой requests. Если соединение не удалось, то скрипт завершается с ошибкой. Если идентификация прошла успешно, то создается объект Handler.

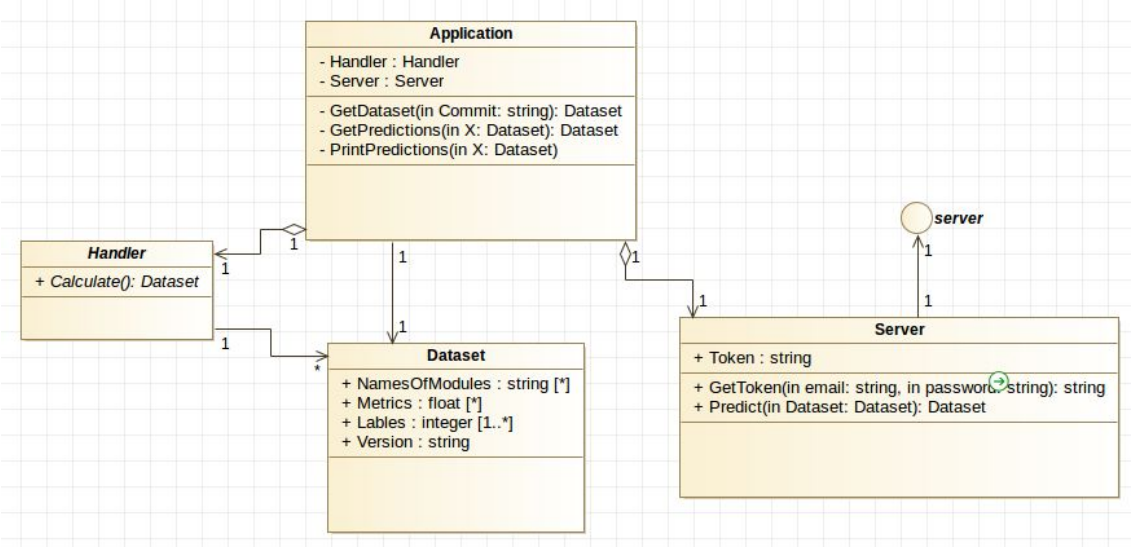
5.1.2 Процесс



5.1.3 Обязанности классов

Класс	**Обязанности**
Application	Инициализирует экземпляры классов Handler, Server, реализует последовательность взаимодействий
Handler	Считает метрики по проекту, возвращает датасет
Dataset	Хранит в себе метрики проекта
Server	Обеспечивает связь с сервером, отсылает метрики, получает предсказания

5.1.4 Диаграмма классов



5.2 Реализация варианта использования Maintenance

5.2.1 Механизм инициализации

Разработчик запускает скрипт, передавая ему две группы параметров, отвечающих каждая своему юзкейсу.
Для AddRepositories:

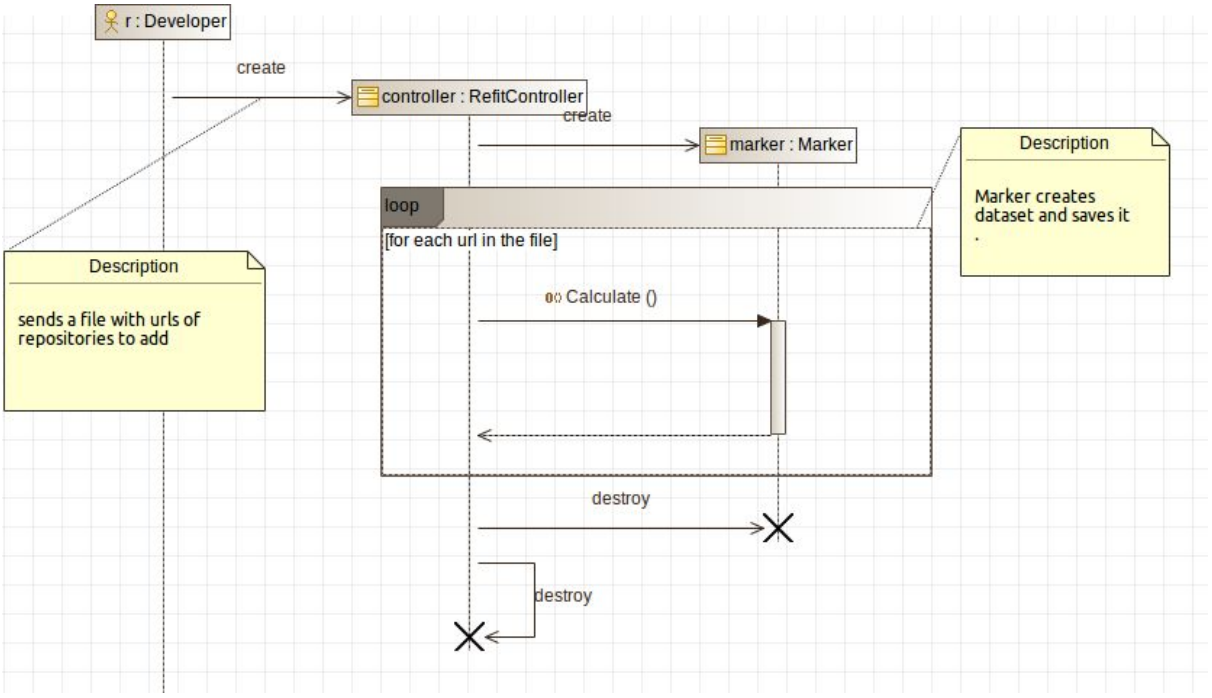
- repo - файл со ссылками на репозитории, которые требуется скачать, превратить в датасеты и разметить
- sdir - место, куда положить новые размеченные датасеты

Для RefitMLModel

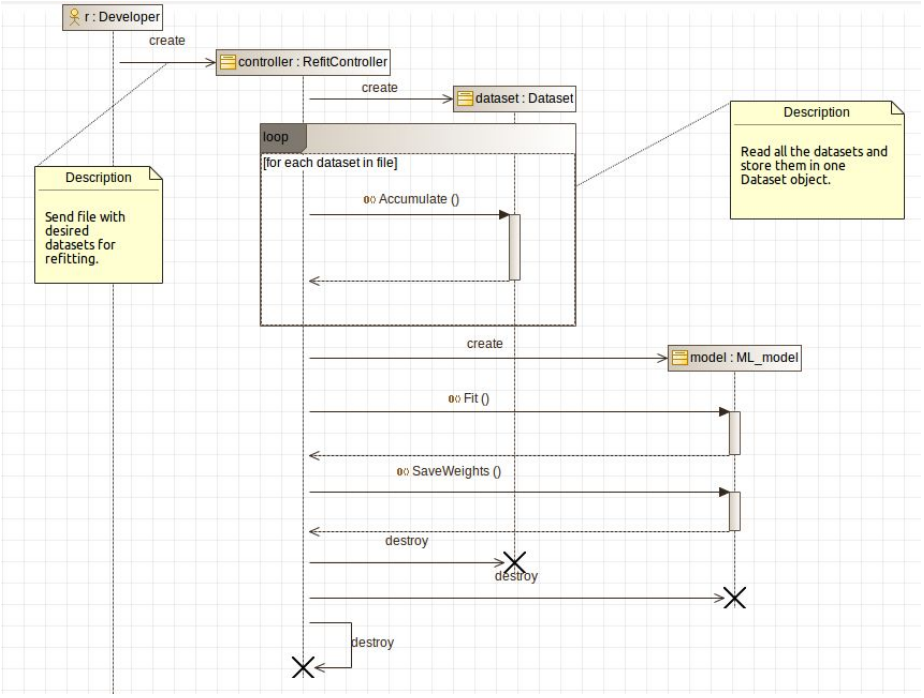
- data - файл с именами датасетов, на которых нужно обучить модель или all
- dir - папка с датасетами, на которых следует обучить модель, если в параметр data было передано значение all

Если переданы обе группы параметров, оба юзкейса выполняются последовательно.

5.2.2 Описание процесса AddRepositories



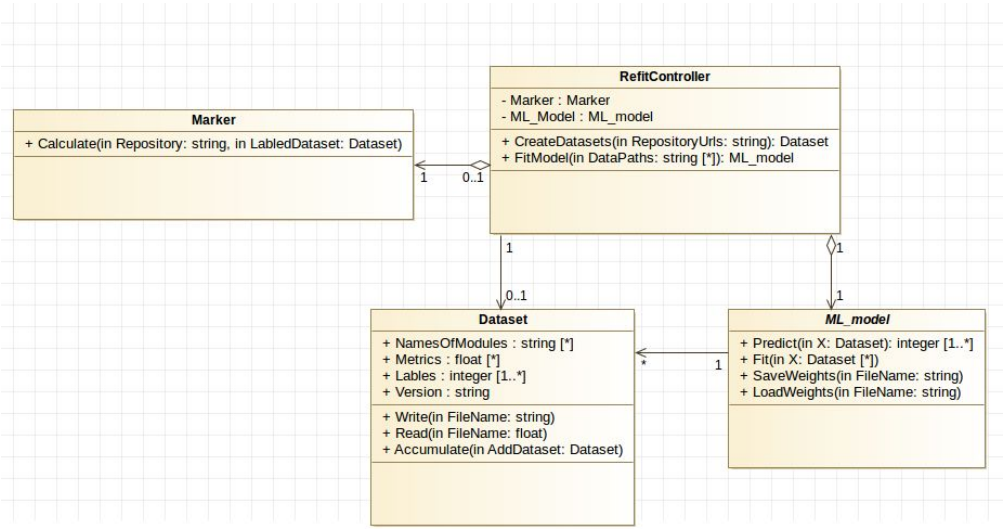
5.2.3 Описание процесса RefitMLModel



5.2.4 Обязанности классов

Класс	**Обязанности**
RefitController	Инициализирует и хранит ML_Model и Marker, исполняет процессы AddRepositories и RefitMLModel
ML_Model	Обучается на размеченном датасете, сохраняет веса в файл
Dataset	Хранит в себе данные (метрики, метки, названия модулей, версия датасета), читает и пишет их в файл, аккумулирует данные нескольких датасетов
Marker	Создает размеченный датасет по ссылке на репозиторий

5.2.5 Диаграмма классов



6. Обеспечение нефункциональных требований

Перечислить в подразделах нефункциональные требования, упоминаемые в постановке задачи. Пояснить, каким образом они удовлетворены в архитектуре и дизайне системы.

- Доступность интегрирования приложения в CI/CD

- Доступность сервера, где лежит ML-модель через Интернет
- Отображение результата работы CI/CD в интерфейсе CI/CD

6.1. Доступность интегрирования приложения в CI/CD

Продукт является встраиваемым в CI/CD gitlab. Пользователь взаимодействует с продуктом с помощью интерфейса CI/CD. Должна быть доступность CI/CD (подключение через Интернет), и приложение должно быть адаптировано к интегрированию в CI/CD, внешний интерфейс должен иметь доступ к использованию приложения.

6.2. Доступность сервера, где лежит ML-модель через Интернет

Класс Application через класс Server, который является каналом в выбранной архитектуре каналов и фильтров, обращается к внешнему интерфейсу server. server обеспечивает выполнение методов GetToken (получение уникального токена с помощью которого реализуется доступ к приложению) и Predict (расчет предсказаний на основе полученных метрик). После расчета предсказаний через канал Server рассчитанные предсказания передаются в Application, и могут быть выведены в лог методом printPredictions. Для осуществления вышеописанной цепочки необходимо иметь доступ к server посредством интернет соединения.

6.3. Отображение результата предсказаний в интерфейсе CI/CD

Пользователь взаимодействует с продуктом с помощью интерфейса CI/CD. Пользователь должен иметь возможность посмотреть полученные предсказания. Полученные предсказания должны быть отображены в логе пользователя. Это осуществляется за счёт встраивания приложения в CI/CD и вызова функции printPredictions.

7. Описание логической структуры и реализация системы

7.1. Логическая структура системы

Диаграмма классов клиента для gitlab:

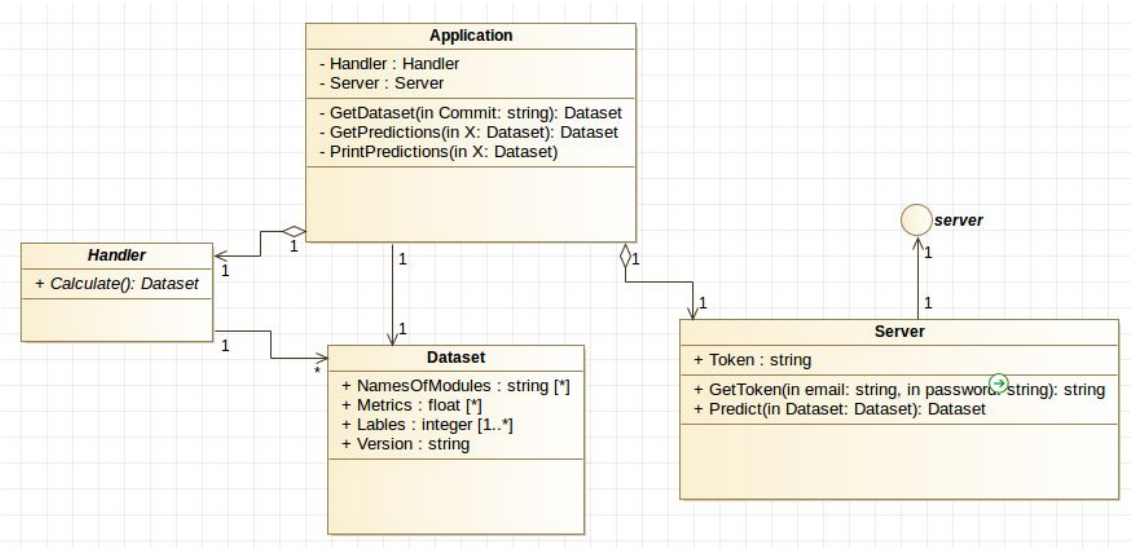
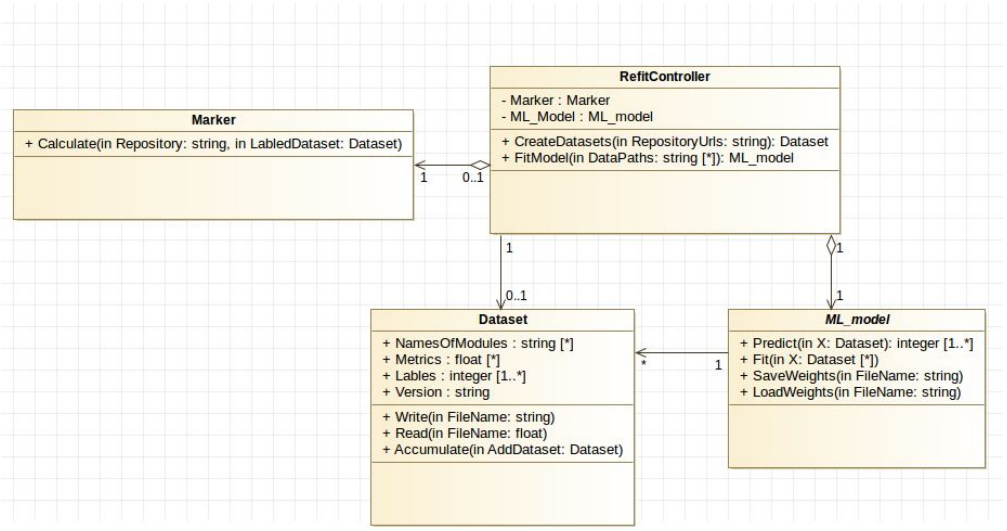


Диаграмма классов клиента поддержки:



Класс	**Контракт**	**Коллеги**
Application	Инициализирует и хранит Server, Handler Выполняет действиеgetDataset: предоставляет Dataset Выполняет действиеgetPredictions: изменяет Dataset Выполняет действие printPredictions: использует Dataset, выводит предсказания	Server Handler Dataset
Server	Хранит Token Предоставляет общение с внешним интерфейсом server: выполняет действия getToken, Predict	Application server

Handler	Абстрактный класс: имеет действие Calculate, предоставляет Dataset	Application JavaHandler (наследник)
JavaHandler	Класс-наследник:: выполняет действие Calculate, использует skjm.sh, предоставляет Dataset	Handler (родитель) skjm.sh (файл-скрипт)
Dataset	Хранит, предоставляет, изменяет массивы имен модулей, метрик, меток, версию, использует dataset	Application RefitController dataset (файл)
RefitController	Инициализирует и хранит Marker, ML_model Выполняет действие CreateDatasets: предоставляет Dataset Выполняет действие FitModel: вызывает действие Fit в ML_model	Dataset Marker ML_model
Marker	Выполняет действие Calculate, создает размеченный Dataset, использует skjm.sh	RefitController git (внешний интерфейс) skjm.sh (файл-скрипт)
ML_model	Выполняет действие Predict, использует Dataset Выполняет действие Fit, использует Dataset Выполняет действия SaveWeights, LoadWeights Использует и изменяет сохраненные веса model.save	RefitController model.save (файл) mlda_svn_stack (наследник)
mlda_svn_stack	Хранит детали реализации алгоритма предсказаний: NormScaler, DenoisingAutoencoder, SVC	ML_model NormScaler DenoisingAutoencoder SVC

NormScaler, DenoisingAutoencoder, SVC - детали реализации алгоритма предсказаний.

7.2. Реализация

Реализации всех классов системы можно увидеть в проекте на python, поставляемом вместе с заданием. Запуск:

- для получения предсказаний по проекту нужно в процессе CI после сборки проекта запустить скрипт application.py
- для выкачивания и разметки новых репозиториев нужно запустить скрипт main.py, передав ему на вход файл с адресами репозиториев и папку для их сохранения в качестве параметров repo и sdir

- для обучения модели на новых репозиториях нужно запустить main.py, передав ему на вход all или файл с путями к датасетам для обучения в качестве параметра data, в случае передачи all можно передать в dir папку с датасетами.

8. План тестирования

№	действие актора	ожидаемый результат	соответствие
1 - get predictions			
1.1	Пользователь запускает скрипт application в процессе CI без параметров	система просит передать параметры при запуске и падает	альтернативный сценарий 1
1.2	Пользователь запускает скрипт application, передает незарегистрированные на сервере email и пароль	система пытается подключиться к серверу, получает ошибку, пишет, что подключение не удалось и падает	альтернативный сценарий 2
1.3	Пользователь запускает скрипт application, передает верные email и пароль, но предусловия не выполнены (например, проект еще не собран)	система подключается к серверу, пытается посчитать метрики, пишет, что подсчитать их не удалось и падает	альтернативный сценарий 3
1.4	Пользователь запускает скрипт application, передает верные email и пароль, все предусловия выполнены.	система подключается к серверу, считает метрики, передает их, получает предсказания, пишет предсказания в лог	основной сценарий
2 - AddRepositories			
2.1	Разработчик запускает скрипт main, передает в него файл со ссылками на репозитории через параметр repo	Система выкачивает репозитории, составляет по ним датасеты и сохраняет их в	основной + альтернативный сценарии

		стандартную папку. Если какую-то ссылку обработать не удалось - система пишет об этом в лог	
2.2	Разработчик запускает скрипт main, передает в него файл со ссылками на репозитории и папку для сохранения через параметры repo и sdir	Система выкачивает репозитории, составляет по ним датасеты и сохраняет их в переданную папку. Если какую-то ссылку обработать не удалось - система пишет об этом в лог	основной + альтернативный сценарии
3 - RefitMLModel			
3.1	Разработчик запускает скрипт main, передавая ему в качестве параметра data файл со ссылками на датасеты	система обучает модель на указанных датасетах, сохраняет веса модели в файл. Если какой-либо из указанных датасетов обработать не удалось, система пишет об этом в лог.	альтернативный сценарий
3.2	Разработчик запускает скрипт main, передавая ему в качестве параметра data значение all	система обучает модель на датасетах из стандартной папки, сохраняет веса модели в файл. Если какой-либо из указанных датасетов обработать не удалось, система пишет об этом в лог.	основной сценарий

9. Вывод

В процессе работы над проектом была спроектирована система для предсказания ошибок в модулях кода в процессе CI и реализованы прототип клиента для gitlab CI и удобный скрипт для дообучения ML модели.