Стандарт на кодирование ПО

Используемый стек технологий

Для реализации требований, предъявляемых к ПО, будут использоваться следующие технологии:

1. Go - язык программирования, используемый для разработки бэкенд сервисов.
2. Python - язык программирования, используемый для написания алгоритмов машинного обучения.
3. Swift - язык программирования, используемый для написания клиентского приложения.

Стандарт кодирования Go

Для реализации требований, предъявляемых к серверной части разрабатываемой системы, выбран язык программирования Golang с использованием официального компилятора (https://go.dev/doc/install). Синтаксис, поведение конструкций языка, поведение данных, а также побочные эффекты языка отражены в официальной документации (https://go.dev/doc/).

Стандарт на форму представления исходного кода

Стандарт на форму представления исходного кода информационной системы соответствует стандартам встроенного в `golang` форматировщика `gofmt` (https://pkg.go.dev/cmd/gofmt) и стандартным правилам синтаксического анализатора `golangci-lint` (https://golangci-lint.run/).

В основе форматировщика `gofmt` лежат правила, созданные разработчиками языка Go (https://go.dev/doc/effective_go).

Данные утилиты совместно с фреймворком `pre-commit` позволяет следить за соответствием исходного кода стандартам форматирования, выводить несоответствия, устранять их в полуавтоматическом или автоматическом режимах, а так же запрещать фиксировать изменения кода, несоответствующие стандарту.

Используемые линтеры

Приведем список используемых линтеров с их кратким описанием:

- asciicheck - проверяет исходный код на наличие non-ASCII символов;
- dogsled - проверяет операторы присвоения на излишне большое количество пустых присвоений;
- errcheck - проверяет исходный код на наличие необработанных ошибок;
- exhaustive - проверяет исходный код на полноту switch конструкций для enum;
- gocognit - проверяет функции исходного кода на когнитивную сложность;
- gocyclo - проверяет функции исходного кода на цикломатическую сложность;
- gofmt - проверяет был ли код форматирован согласно правилам gofmt;
- goimports - проверяет форматирование зависимостей;
- gosimple - производит упрощение кода;
- govet - производит анализ исходного кода на наличие возможных ошибок;
- ineffassign - проверяет исходный код на наличие неиспользуемых переменных;
- lll - проверяет исходных код на слишком длинные строки;
- misspell - проверяет код на наличие опечаток;
- nestif - проверяет код на излишнюю вложенность условных операторов;
- staticcheck - проводит статический анализ кода;
- typecheck - проводит типизированный анализ кода;
- unconvert - проводит анализ исходного кода на ненужные приведения типов;
- unused - проверяет исходный код на наличие неиспользуемых переменных, констант, типов и так далее;
- whitespace - проверяет строки исходного кода на открывающие и завершающие пробелы;
- durationcheck - проверяет исходный код на возможные ошибки при перемножение временных типов;
- forcetypeassert - проверяет исходный код на наличие принудительного приведения типов;
- importas - проверяет исходный код на правильные алиасы для зависимостей;
- predeclared - проверяет исходный код на переопределение одного из объявленных идентификаторов;
- tagliatelle - проверяет структурные теги в исходном коде;
- godot - проверяет комментарии в исходном коде;
- wsl - добавляет пустые строки в исходный код для улучшения читаемости;
- gocritic - предоставляет диагностику, которая проверяет наличие ошибок, проблем с производительностью и стилем;
  расширяемый без перекомпиляции с помощью динамических правил;
  динамические правила написаны декларативно с использованием шаблонов AST, фильтров, сообщения отчета и необязательного предложения;
- prealloc - проверяет исходный код на места, где можно использовать выделение памяти;
- stylecheck - проверяет исходный код на следование стилю;
- nlreturn - проверяет исходный код на отсутствие новой строки перед return;

- unparam - проверяет исходный код на наличие неиспользуемых параметров в функциях;

Стандарт на документирование

Стандарт на документирование разработанного кода задается при помощи пакета `godoc` (https://pkg.go.dev/golang.org/x/tools/cmd/godoc). Согласно стандарту написания документации (https://tip.golang.org/doc/comment), каждая экспортируемая часть кода (функция, класс, интерфейс, пакет и т.д.) должна иметь свой документирующий комментарий. Размещая такие комментарии в коде и обращаясь к `godoc` можно получить веб-приложение, предоставляющее всю информацию об экспортируемых методах в виде веб-страниц.

Средствами документирования так же обеспечивается трассируемость исходного кода на Go. В документирующем комментарии указывается версия требования, которое реализуюеся в текущей части кода, и ее номер. Таким образом обеспечивается связь исходного кода и требований, которые в нем реализуются.

Стандарт на модульные тесты

Стандарт написания модульных тестов описан в стандартном пакете `testing` (https://pkg.go.dev/testing). Тесты для каждой функции группируются согласно подходу "Table Driven Unit Test".

Стандарт на организацию проекта

В качестве макета организации проекта используется (https://github.com/golang-standards/project-layout), совмещенный с подходами Clean Architecture.

- директории Go:
  - /cmd - основные приложения проекта;
  - /internal - внутренний код прилождений и библиотек;
    - /entity - сущности бизнес-логики;
    - /interactor - интеракторы, реализующие бизнес-логику;
    - /repository - репозитории, осуществляющие работу с данными;
  - /pkg - код библиотек, пригодных для использования в сторонних приложениях;
  - /vendor - зависимости приложений;
- директории приложений-сервисов:
  - /api - спецификации OpenAPI/Swagger, файлы JSON schema, файлы определения протоколов;
- директории web-приложений

    - /web - специальные компоненты для веб-приложений: статические веб-ресурсы, серверные шаблоны и одностраничные приложения;
- распространенные директории:
    - /configs - шаблоны файлов конфигураций и файлы настроек по-умолчанию;
    - /init - файлы конфигураций для процессов инициализации системы (systemd, upstart, sysv) и менеджеров процессов (runit, supervisord);
    - /scripts - скрипты для сборки, установки, анализа и прочих операций над проектом;
    - /build - сборка и непрерывная интеграция (Continuous Integration, CI);
    - /deployments - шаблоны и файлы конфигураций систем оркестраций IaaS, PaaS, операционных систем и контейнеров (docker-compose, kubernetes/helm, mesos, terraform, bosh);
    - /test - дополнительные внешние приложения и данные для тестирования;
- другие директории
    - /docs - документы пользователей и дизайна (в дополнение к автоматической документации godoc);
    - /tools - инструменты поддержки проекта;
    - /examples - примеры приложений и/или библиотек;
    - /third_party - внешние вспомогательные инструменты, ответвления кода и другие сторонние утилиты (например, Swagger UI);
    - /githooks - git hooks;
    - /assets - другие ресурсы, необходимые для работы (картинки, логотипы и т.д.);

Стандарт кодирования Python

Для реализации требования, предъявляемых к серверной части разрабатываемой системы, отвечающей за нейронные сети, выбран язык программирования Python.

Стандарт на форму представления исходного кода

Стандарт на форму представления исходного кода, написанного на Python, соответствует стандарту написания кода, предложенному разработчиками языка Python, а именно `PEP 8` (https://pep8.org/).

В качестве синтаксических анализаторов используются линтеры `flake8` (https://flake8.pycqa.org/en/latest/) и `pylint` (https://pylint.pycqa.org/en/latest/).

Стандарт на документирование

Стандарт на документирования описан в `PEP 257`, который, так же как `PEP 8`, был разработан и предложен создателями языка Python. Данный стандарт описывает правила написания docstring - строковых переменных, которые идут сразу за объявлениями модулей, функций, классов, методов. С использованием генератора документации `pydoc` (https://docs.python.org/3/library/pydoc.html) можно получить документа в любом формате.

Средствами документирования так же обеспечивается трассируемость исходного кода на Python. В документирующем комментарии указывается версия требования, которое реализуюеся в текущей части кода, и ее номер. Таким образом обеспечивается связь исходного кода и требований, которые в нем реализуются.

Стандарт на модульные тесты

Стандарт на написание модульных тестов описан в стандартном пакете `unittest` (https://docs.python.org/3/library/unittest.html).

Стандарт на организацию проекта

В качестве макета будем использовать следующую структуру директорий и файлов:

- /data - директории с данными;
  - /external
  - /interim
  - /processed
  - /raw
- /src - исходный код проекта;
  - /data - скрипты для работы с данными;
  - /features - скрипты для преобразования данных в features;
  - /models - скрипты для тренировки и использования обученных моделей;
  - /visualization - скрипты для визуализации
- другие директории
  - /reports - отчеты по моделям;
  - /references - разъяснительные материалы к данным и моделям;

Стандарт кодирования Swift

Для реализации требования, предъявляемых к клиентской части разрабатываемой системы выбран язык программирования Swift, разработанный компанией Apple, для написания приложения для своих платформ.

Стандарт на форму представления исходного кода

Стандарт написания кода на Swift описан в официальном гайде по стилю (https://google.github.io/swift/).

В качестве синтаксического анализатора используется линтер `SwiftLint` (https://github.com/realm/SwiftLint)

Стандарт на документирование

В качестве основного инструмента для генерации документации используется решение от Apple - `DocC` (https://developer.apple.com/documentation/docc). Данный инструмент использует особый синтаксис - модифицированную версию языка разметки Markdown, для генерации документации.

Стандарт написания документации при помощи Markdown описан в официальном гайде от Apple (https://developer.apple.com/documentation/xcode/formatting-your-documentation-content).

Средствами документирования так же обеспечивается трассируемость исходного кода на Swift. В документирующем комментарии указывается версия требования, которое реализуюеся в текущей части кода, и ее номер. Таким образом обеспечивается связь исходного кода и требований, которые в нем реализуются.

Стандарт на организацию проекта

В качестве макета будем использовать макет, основанный на принципах MVVM и Clean Architecture.

- директории приложения:
  - /Application - файлы конфигурации;
  - /Data - исходный код для работы с данными (репозитории, gateways и т.д.);
  - /Domain - исходный код, реализующий бизнес-логику:
    - /Entity - сущности бизнес-логики;
    - /Interactor - интеракторы, реализующие бизнес-логику;

    - /Presentation - исходный код, реализующий представления;
- директория тестов;
- директория тестов интерфейса;