Инструкция для пользователей

Nested Chat

Департамент лингвистики и ИИ

ООО Промобот

Создана: 30/11/2021

Обновлена: 21/10/2022

Оглавление

Вводная часть

Nested Chat – это диалоговая платформа компании Промобот, с микросервисной архитектурой, позволяющая создавать сценарии взаимодействия с роботом, выводя качество диалога на новый уровень за счет возможности поддержания контекста. Структура сценария позволяет избежать одноуровневой системы правил, которая мешает созданию грамотного диалога между человеком и роботом.

Функционал включает:

– Обработку запросов целиком на естественном языке с возможностью подключения распознавания по встроенным словарям, ключевым словам, и модулю NER;

– Сентимент анализ запроса – понимание эмоций по тексту;

– Умную Fallback-реакцию, если робот не знает ответа;

– Подключение любых типов экшенов – фото, видео, аудио, картинки, вывод текста, взаимодействие со сторонними API*;

– Персонализацию проекта – роли заполнения проекта;

– Полную автономность (выгрузка и загрузка проекта в формате JSON);

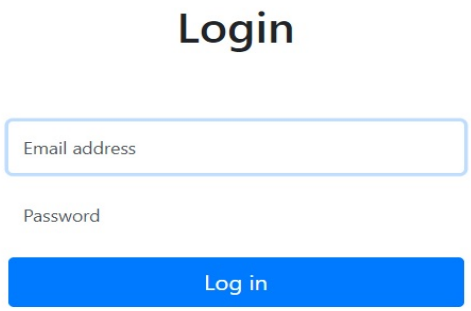
– Расширенную мультиязычность (106 языков);

– Расширенную мультимодальность;

– Возможность интеграции с мессенджерами и умными колонками и т. д.

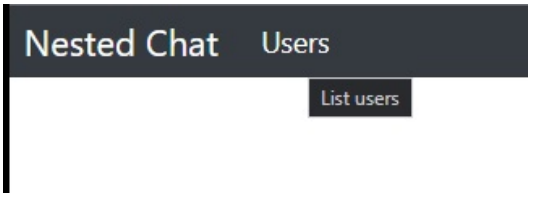
Авторизация

Чтобы начать работу в Nested Chat, необходимо зайти на сайт (https://ds.promo-bot.ru/login) и ввести адрес электронной почты и пароль.



Шаг 1

Создать учетную запись для нового пользователя может модератор. Для этого необходимо в левом верхнем углу найти вкладку **«Users»** и нажать на нее.



Шаг 2

Вы увидите список всех пользователей Nested Chat, для добавления нового пользователя нужно нажать на кнопку **«New»**, которая расположена в правом верхнем углу списка.

Шаг 3

Для регистрации нового пользователя необходимо заполнить основную информацию

1. **email** – адрес электронной почты
2. **name** – имя пользователя
3. **surname** – фамилия пользователя
4. **password** – пароль
5. **confirm password** – подтверждение пароля
6. **role** – пользовательская роль

Типы пользовательских ролей:

Guest

- имеет доступ только к назначенным ядрам, может запускать их TRAIN и видит только их. Не может создавать ядра сам.

Marker

- имеет доступ только к назначенным проектам, может видеть все существующие проекты

Moderator

- имеет доступ ко всем проектам
- может вносить правки во все проекты
- может приписывать/запрещать доступ к проектам

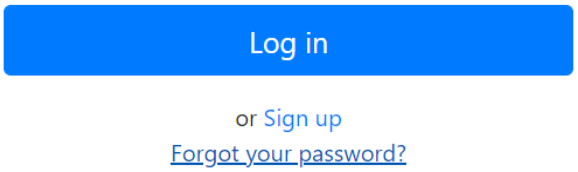
7. **available cores** – доступные проекты (ядра)

Примечания:

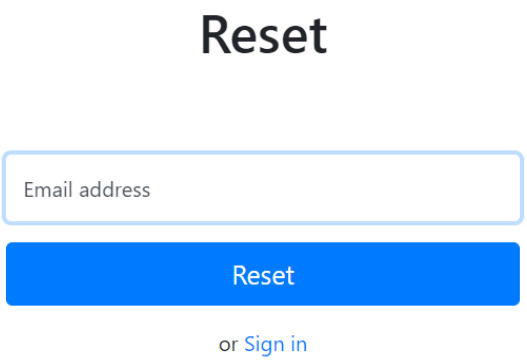
- Чтобы выбрать несколько ядер необходимо зажать клавишу **«ctrl»** и кликнуть на нужные ядра левой кнопкой мыши.

Чтобы сохранить изменения нажмите на кнопку **«Save»**

В случае если пользователь забыл пароль, у него есть возможность сбросить старый пароль, нажав на **«Forgot your password»**?



В следующем окне выйдет поле, в которое необходимо вписать e-mail, по которому пользователь был зарегистрирован.

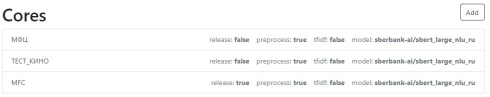


Далее на почту пользователю придет новый пароль. Осуществлять вход на сайт будет возможно, используя этот пароль.

Создание ядра

Шаг 1

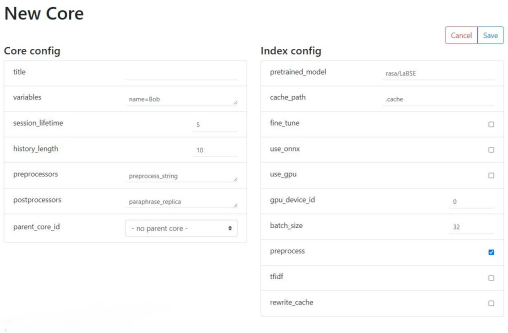
Для того, чтобы создать новое ядро на главной странице сервиса, необходимо зайти в список ядер и нажать на кнопку «**Add**».



Шаг 2

После нажатия на кнопку появится список параметров ядра, которые необходимо заполнить

Расшифровка параметров:



- **title** - название общего (большого) проекта.

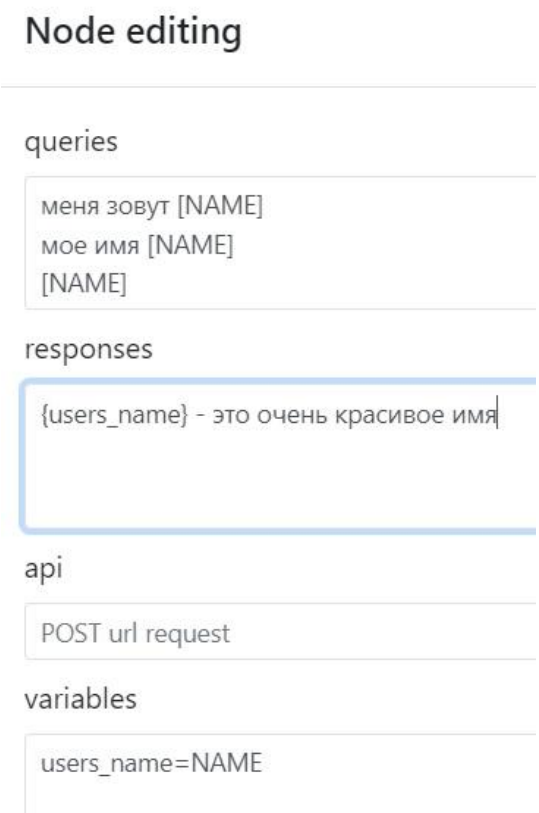
Система нейминга ядер:
Название ядра должно состоять из трех частей разделенных нижним подчеркиванием и содержать следующую информацию: “**название ядра_ЯЗЫК_род**”

Параметры названия:
- только латиница
- название ядра с маленькой буквы
- нижнее подчеркивание вместо пробелов
- обозначение языка стандартным сокращением в высоком регистре (DE, ENG, RUS)
- род (от чьего лица ведется диалог) в низком регистре. Допустимы только два обозначения: male (мужской) и female (женский)
- если ядро универсальное и может подключаться к другим ядрам через parent_id (является кастомным), то дополнительно четвертым параметром в конце добавлять к названию “_custom”
- если ядро является тестовым, то дополнительным четвертым параметров в начале названия добавлять “test_”
- если в ядре используются имена собственные, то они должны записываться с большой буквы (“Fedya”)

- **variables** – глобальный контекст. Слово/слова, которые в контексте всего ядра должны обозначать одно понятие. Если в **variables** создается глобальный контекст (***name*** = *Промобот*), тогда каждый раз, когда в тексте речи робота встречается {name}, в устной речи оно будет заменяться на «Промобот». Иначе это реализовано в создании локального контекста в рамках конкретной истории, где в локальный контекст также

можно записать все, что говорит человек на уровне данной пары вопрос-ответ. В этом случае ***variables*** = *queries*;

Если в **queries** есть вопрос с именованной сущностью, то можно записать эту сущность в переменную:



Также в variables можно записывать ссылки, эмоции. Чтобы не было ошибок в классификации variables, мы настаиваем на использовании постоянных тегов для variables:

тип variables	используемый тег
ссылка	URL_
эмоция	emotion_
движение	motion_

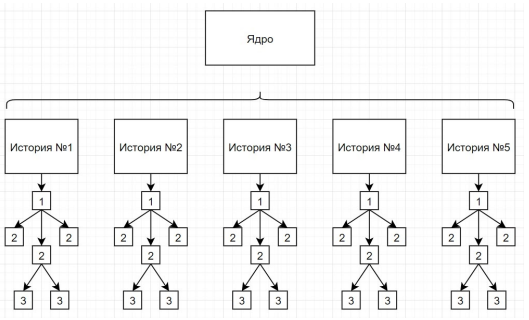
- **session_lifetime** - Время хранения сессии с контекстом с момента последнего взаимодействия. Рекомендуемое количество до 5.
- **history_length** - Максимальная длина контекста сессии. Хранит последние данные из диалога с пользователем, чтобы их использовать для поддержания диалога или для того, чтобы запоминать пройденные узлы сценария. Значение обозначает количество последних пройденных узлов. По умолчанию 10.

- **preprocessors** - список из action-скриптов, которые будут последовательно выполняться перед тем, как реплика попадает в движок диалоговой системы. Иначе говоря, перед тем, как входящая фраза будет обрабатываться.
- **postprocessors** - список из action-скриптов, которые будут последовательно выполняться после того, как движок диалоговой системы ответил.
- **parent_core_id** - название ядра, истории которого будут использоваться в созданном ядре. Доступно подключение только одного ядра. Если необходимо использовать несколько ядер, оформляйте их в виде водопадного наследования, при котором к основному ядру через parent_core_id привязывается наиболее крупное ядро, к которому в свою очередь привязывается ядро меньшего размера и значимости, и так далее.
- **pretrained_model** – название языковой модели, используемой в ядре (по умолчанию – sberbank-ai/sbert_large_nlu_ru);
- **cache_path** – путь к папке где хранится кэш (выставляем настройку по умолчанию)
- **fine_tune** – дообучение pretrained_model на queries узлов (по умолчанию false).
- **use_onnx** – оптимизация языковой модели на CPU (по умолчанию false).
- **use_gpu** – использование GPU(false – CPU, true – GPU (выполняется быстрее, но только если на сервере есть видеокарта));
- **gpu_device_id** – номер используемой видеокарты (по умолчанию 0);
- **batch_size** – скорость инференса языковой модели (увеличение значения повышает ресурсоемкость). Параметр по умолчанию 32;

Затем нажимаем **«Save»** чтобы сохранить.

Создание сценария

Политика работы сценария



Используя ядро, мы получаем доступ ко всем историям, которые в нем есть, поэтому обрабатывая первый запрос пользователя, мы выбираем среди начальных фраз истории наиболее похожую, и переходим на второй уровень вопросов сценария.

Если все проходит, так как запланировано и пользователь ведет диалог с роботом именно так, как это подразумевалось сценарием, то он движется вниз по выбранной ветке, а после завершения сценария возвращается обратно к выбору истории.

Если пользователь в середине диалога решает перевести тему и задает тот вопрос, которого в сценарии данной истории на том уровне нет, мы поднимаемся до самого верхнего уровня и ищем похожий запрос во всех начальных фразах доступных историй.

Создание сценария (истории)

Шаг 1

Создаем новую историю нажав на кнопку **«Add»**.

Шаг 2

Заполняем основную информацию

- **title** – наименование истории.
- **topic** – тема
- **domain** – направленность сценария («business» или «lets_talk**»**)

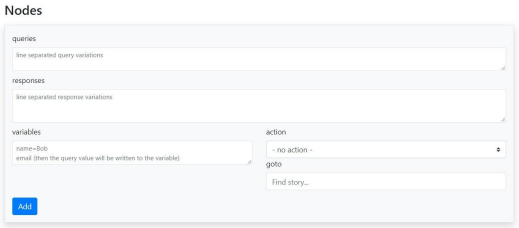
Затем нажимаем **«Save»** чтобы сохранить.

Шаг 3

Заполнение сценария

Сценарий (история) – это диалог пользователя с роботом, представляющий собой перечень вопросов и ответов. Каждая новая пара вопрос-ответ – это новый уровень сценария. Таких уровней может быть несколько, все зависит от целей, который ставит перед собой создатель сценария.

Заполнение истории начинается с первой (входящей фразы), именно она является триггером для начала взаимодействия с роботом и/или началом нового разговора внутри одного взаимодействия.

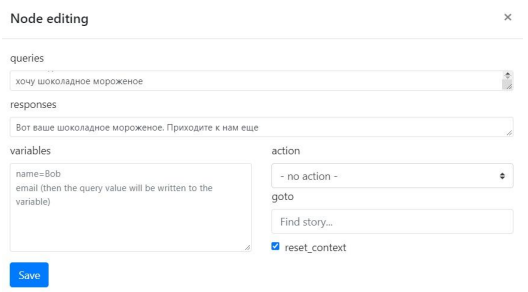


Поля доступные для заполнения одного уровня сценария:

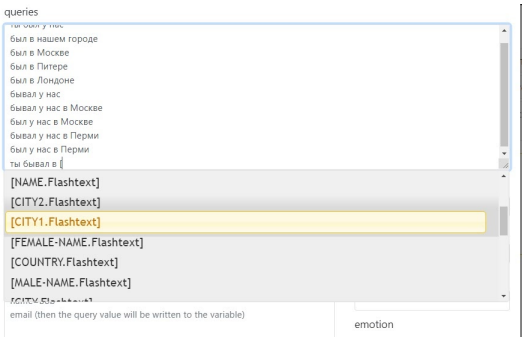
1. **«queries»** – фразы пользователя. Важно помнить, что сначала будут распознаваться обычные фразы, а уже потом правила. Правила оформляются по правилам регулярных выражений (Не забудьте перед правилом поставить знак **«@»**). Каждый новый вариант пишется с новой строки (Enter для перехода на новую строку).

   Чтобы реагировать на те реплики, которые не были заложены в структуру диалога при его создании, можно добавлять на нужный уровень распознавание любой фразы, кроме тех, которые уже добавлены, для этого используйте реплику с «@.*» в графе **«queries»**. В ответ робота вы можете записывать фразы, благодаря которым можно попросить перефразировать ответ или уточнить, что именно пользователь имел в виду. При использовании такого подхода необходимо помнить о том, что после данного регулярного выражения выход из истории будет не возможен. Для того чтобы избежать зацикливания, необходимо воспользоваться функцией **reset_context**. Добавляя ее на финальную

реплику истории, мы стираем контекст, и следующий заданный вопрос будет адресован всем корневым узлам историй. Чтобы установить **reset_context** на узел необходимо отметить этот параметр галочкой:



При добавлении вариантов на распознавание в истории могут быть использованы **газеттиры** (тематические списки + регулярные выражения). Они работают также как ключевые слова. Основные преимущества – покрывают всю заданную тематику и распознают изменяемые формы слова. Минусы большие трудозатраты при сборе данных. Принцип работы: выбираем из выпадающего списка, например, CITY, и робот распознает все строки, где было упоминание города. Для того чтобы вам выпал список, напишите открывающую квадратную скобку **[** . Если вы знаете название списка, то начните писать его, Nested Chat предложит вам списки, в названиях которых упоминается набранное вами слово.

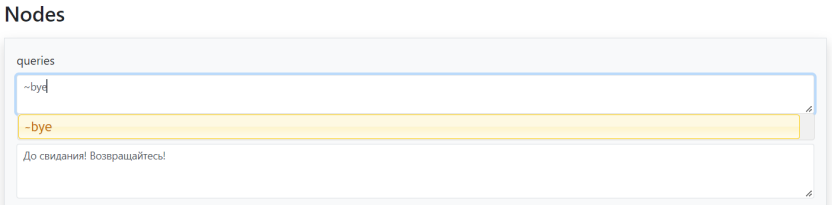


Список газеттиров:

- [TEMPERATURE] – температура
- [INTERNATIONAL-PASSPORT] – номер загранпаспорта
- [PERCENT] – проценты
- [PASSPORT-NUMBER] – номер паспорта РФ
- [AREA] – площадь
- [FOOD] – названия блюд
- [NAME] – имена
- [CITY] – все города
- [CITY1]– только Пермь
- [CITY2]– региональные центры России
- [FEMALE-NAME] – женские имена
- [COUNTRY] – названия стран
- [MALE-NAME] – мужские имена
- [TIME] – время
- [NUMBER] – числа
- [NUMERAL] – количественные числительные
- [AMOUNT-OF-MONEY] – денежные суммы
- [DISTANCE] – расстояние
- [QUANTITY] – вес
- [VOLUME] – объем
- [ORDINAL] – порядковые числительные
- [EMAIL] – адрес электронной почты
- [PHONE-NUMBER] – номер телефона

- [URL] – ссылка
- [CREDIT-CARD-NUMBER] – номер кредитной карты

При заполнении сценария Nested Chat могут быть использованы **интенты** из вкладки Intents. Преимущества: сокращение времени заполнения сценария в Nested Chat, оперативное внесение правок. Недостатки: большие трудозатраты при сборе данных. Принцип работы: выбираем интент из выпадающего списка, робот распознает все строки, где было упоминание этого слова. Для того чтобы вам выпал список, напишите тильду ~ . Если вы знаете название необходимого вам интента, то начните писать его, Nested Chat предложит вам списки, в названиях которых упоминается набранное вами слово.



Распознавание запроса пользователя:

4 метода:

1. Строка на естественном языке. *Например*: «Выдай паспорт».
2. Правило (ключевое слово в неизменяемой форме). *Например:* «@паспорт»
3. Правило (регулярное выражение). *Например:* «@^.*(паспорт/документ/загран).*$»
4. Газеттиры из вкладки Entities. *Например*: «[PASSPORT-NUMBER]»
5. Интенты из вкладки Intents. *Например*: ~bye
6. Сущности (контекстное распознавание на основе нейросетевой архитектуры) позволяют распознавать ключевые упоминания в контексте. Например: Москва может быть распознана в зависимости от контекста в качестве реки, города или прогулочного пароходика.*

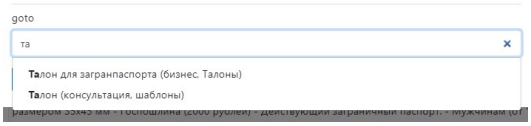
   Сущности, доступные для распознавания;

Название сущности	**Информация**
Location (локации)	• названия стран/городов/деревень • географические названия • здания и сооружения • внутренние помещения • адреса

Organization (организации)	• геополитические названия в роли организации • каналы, радиостанции, печатные издания • организации, предприятия, министерства
Person (имена)	• имена людей (реальных) • должности, профессии, род занятий • клички, имена вымышленных героев, прозвища
Product (продукты)	• произведения искусства • документы • бренды, торговые марки, названия музыкальных групп • услуги • события
Food (еда)	• названия блюд
Misc (общее)	• погода • флора и фауна

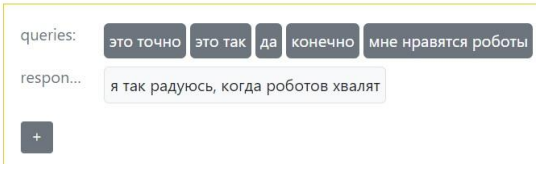
2. **«responses»** – ответ/ответы робота;
3. **«action»** – действие робота/ссылки/эмоции/лингвистические процессоры (название скрипта определяется названием основной функции скрипта) (данное поле заполняется при необходимости);
4. **«goto»** – ссылка на другую историю (данное поле заполняется при необходимости)

GOTO можно использовать для перехода в другой сценарий (в рамках одного ядра) или в начало текущего сценария (например, чтобы начать диалог по сценарию заново). Начинаем вводить title, а затем выбираем нужный вариант из предложенных:

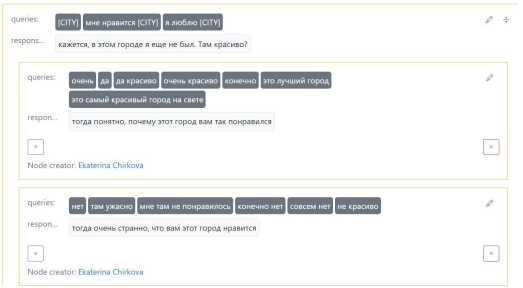


Шаг 4

История должна состоять минимум из одного уровня, но если есть необходимость создать несколько уровней в сценарии, можно воспользоваться переходом на второй уровень, для этого, необходимо нажать на знак «+» под той фразой, после которой последует переход на новый уровень диалога.



Несколько вариантов queries одного уровня должны располагаться с одним отступом (друг под другом). Варианты ответа пользователя на одном уровне не должны противоречить друг другу.



Шаг 5

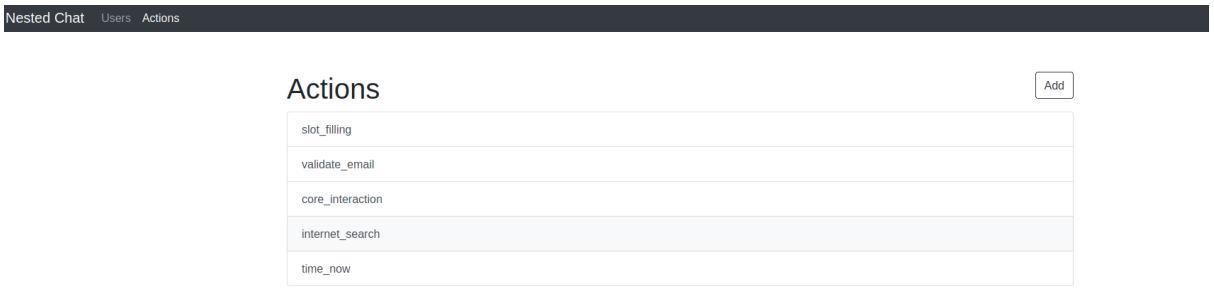
Заполнение вопросов и ответов последующих уровней идентично заполнению вопросов и ответов первого уровня. Вы можете делать переход на новый уровень во всех ветках диалога, или выборочно в тех, которые нужно развить дальше. Рекомендуемая глубина диалога 3-4 уровня.

Создание скриптов

Скрипт - это программа на языке Python, которая может быть создана в графическом редакторе кода Nested Chat. Затем сохраненная программа может быть использована в сценариях. Скрипты предназначены для расширения возможностей Nested Chat.

Шаг 1

Откройте вкладку Actions. Здесь будут все ваши скрипты. Чтобы создать скрипт, нажмите кнопку **«Add»**.



Шаг 2

Введите **имя скрипта**.

Система нейминга скриптов:
Название скрипта должно соответствовать формату записи “Snake Case”, а именно:

- состоять минимум из двух частей разделенных нижним подчеркиванием
- содержать следующую информацию: название скрипта, направленность скрипта (именно в таком порядке)
- в названии могут использоваться только латинские символы нижнего регистра и нижнее подчеркивание
- если скрипт является тестовым, то дополнительным третьим параметров в начале названия добавлять “test_”

Обозначение направленности:

_quest	скрипт направлен на проведение теста/анкетирования
_game	скрипт представляет из себя игру
_generation	скрипт позволяет выводить сгенерированную информацию (стихи/предсказания/сообщения)
_validity	скрипт проверяющий правильность вводимых данных
_search	скрипт позволяющий выдавать информацию из источника (базы данных, словаря, файла, интернет-ресурса)
_extract	скрипт позволяющий извлекать необходимую информацию из диалога с пользователем
_postprocessor	скрипт обрабатывает стандартный ответ сервиса
_preprocessor	скрипт обрабатывающий полученную от

	пользователя фразу

Необходимо соблюдать правила именования скрипта согласно правилам именования функций в Python. Например, **my_new_action**, как показано ниже. Вам нужно предоставить хотя бы одну функцию, название которой совпадает с **именем скрипта**. Эта функция называется **главной**. Она запускается каждый раз, когда сценарий вызывает скрипт. Сигнатура главной функции должна следовать формату, который предоставлен в шаблоне. Вы можете менять название функции и название аргумента (request, по-умолчанию)

Шаг 3

Напишите код для вашего скрипта. Вы можете создавать дополнительные функции и использовать их в главной функции. Вы можете создавать переменные вне функций. Чтобы сохранять контекстные переменные скрипта, создайте новую пару ключ/значение в словаре variables, который доступен в теле запроса (request). Очень важно понимать ответственность за утечки памяти и требования к вычислительным ресурсам.

Вы можете использовать все стандартные библиотеки Python не ниже версии 3.8. Те библиотеки, которые требуют установки, достаточно один раз установить, написав (строго) в первой строке скрипта “**#deps:**”, а затем через запятую без пробелов названия тех библиотек (названия должны соответствовать названиям с https://pypi.org/), которые необходимо установить. Когда библиотеки установлены, не забудьте использовать import. Настоятельно рекомендуется не удалять строку с установкой библиотек во избежание потери доступа к скрипту в дальнейшем.

Editing Action "action_visit_counter"

Cancel Save

action_visit_counter

```python
from app.sdk import ActionRequest, ActionResponse

def my_nlg_template(times: int):
  postfix = "раза" if times % 10 in [2, 3, 4] else "раз"
  return f"Вы обратились ко мне {times} {postfix}"

def action_visit_counter(request: ActionRequest) -> ActionResponse:
  try:
    session_id = request.session_id
    variables = request.variables

    times = variables.get('visit_counter', 0)
    times += 1

    # we store visits in session variables
    variables['visit_counter'] = times

    return ActionResponse(
      message=my_nlg_template(times)
    )
  except Exception as e:
    # return empty ActionResponse on failure
    return ActionResponse()

```

Шаг 4

Настройте узел для использования скрипта. Для этого в поле action (например, **time_now**) выберете ваше действие по имени. Затем обучите ядро, чтобы было известно какой скрипт нужно запускать. Узел готов к работе. Вы можете протестировать работу в чате.

Node editing

queries

Сколько времени?
Который час?
Сколько сейчас времени?
время

responses

К сожалению, в данный момент у меня нет доступа к часам

time_now

Вы можете редактировать скрипт. После редактирования кода не нужно обучать ядро. Если вы меняете название скрипта, убедитесь, что узлы тоже отредактированы. Вы можете удалить скрипт, если он больше не нужен.

Тестирование скрипта

При создании экшена появляется поле TESTS, которое отвечает за тестирование скрипта.

При проверке написанного скрипта необходимо убрать часть кода TESTS из комментариев. В поле ActionRequest пишем query, то есть то, что должен сказать пользователь. В поле ActionResponse пишем ожидаемое сообщение, то есть то, что должен вывести код. Далее нажимаем кнопку **«Save».**

Если скрипт отрабатывает корректно, то код сохранится. Иначе появится сообщение об ошибке в следующем формате: в поле expected – тестируемые вами данные, в поле returned – результат, который выводит скрипт.

Создание именованных сущностей

Именованная сущность - это слово или словосочетание обозначающее предмет или явления определенной категории. Примерами именованных сущностей являются имена людей, названия организаций и локаций.

Шаг 1

Откройте вкладку Entities. Здесь будут все ваши сущности. Чтобы создать новую сущность, нажмите кнопку **«Add».**

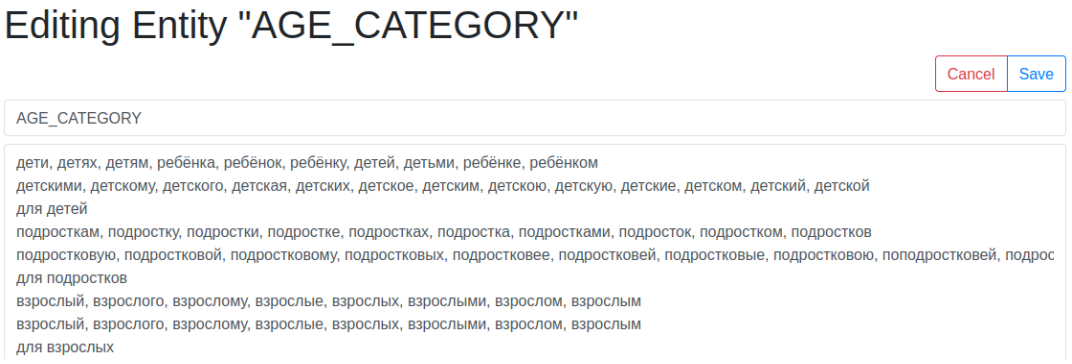
Шаг 2

Введите **имя скрипта**.

Система нейминга сущностей:
Название сущности должно соответствовать формату записи “Snake Case”, а именно:

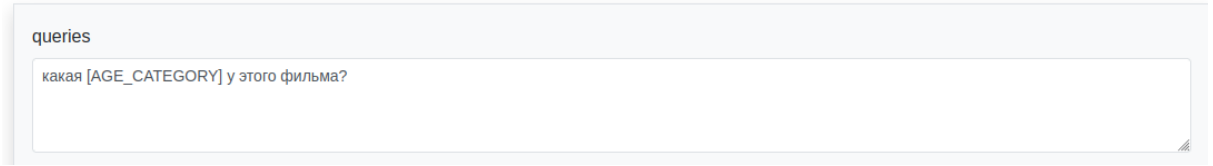
- состоять минимум из двух частей разделенных нижним подчеркиванием
- содержать следующую информацию: название сущности (отражающее общий смысл элементов сущности), обозначение языка стандартным сокращением в высоком регистре (DE, ENG, RUS) (именно в таком порядке)
- в названии могут использоваться только латинские символы верхнего регистра и нижнее подчеркивание
- если сущность является тестовой, то дополнительным третьим параметров в начале названия добавлять “TEST_”

В поле ввода вы можете создать столько примеров сущностей сколько вам требуется. Каждая строка отдельный пример, где мы можете перечислить через запятую все словоформы. Сущность определяется только по строгому совпадению.



Шаг 3

Используйте ваши сущности в сценариях. В поле ввода **queries** вы можете использовать сущность, чтобы не перечислять все возможные варианты. Для этого введите [, чтобы открылся открывающийся список со всеми сущностями.

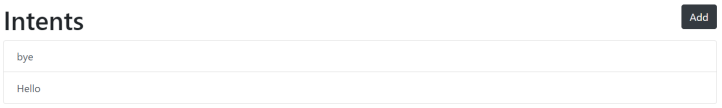


Создание интентов

Интент - пользовательское намерение (приказ, просьба, вопрос, запрос). Как правило, интент уже содержит сущность, но не обязательно. Примеры: "сделай кофе", "скопируй, пожалуйста", (скажи) как дела? и др.

Шаг 1

Откройте вкладку Intents. Здесь будут все ваши интенты. Чтобы создать новый интент, нажмите кнопку **«Add».**



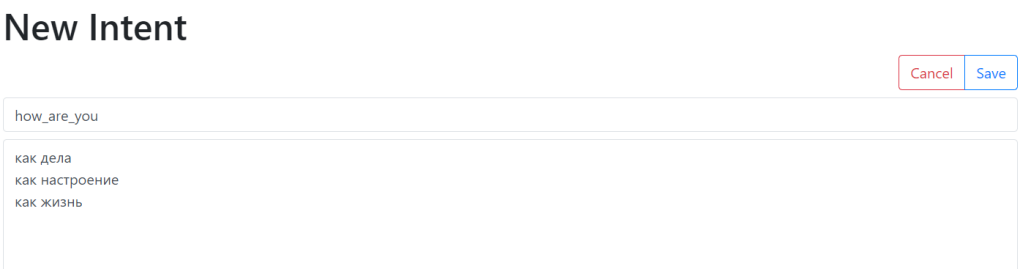
Шаг 2

Введите **название интента**.

Система нейминга интентов:
Название интента должно соответствовать формату записи “Snake Case”, а именно:

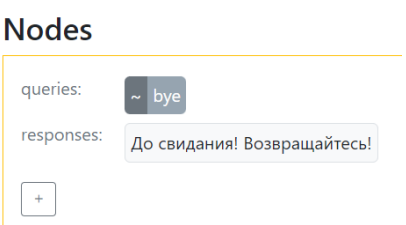
- состоять минимум из двух частей разделенных нижним подчеркиванием
- содержать следующую информацию: название интента (отражающее общий смысл элементов интента), обозначение языка стандартным сокращением в высоком регистре (DE, ENG, RUS) (именно в таком порядке)
- в названии могут использоваться только латинские символы нижнего регистра и нижнее подчеркивание
- если интент является тестовым, то дополнительным третьим параметров в начале названия добавлять “test_”

Название должно быть написано латиницей, вместо пробелов используйте нижнее подчеркивание. В поле ввода вы можете создать столько интентов, сколько вам требуется. Каждая строка – отдельный пример.



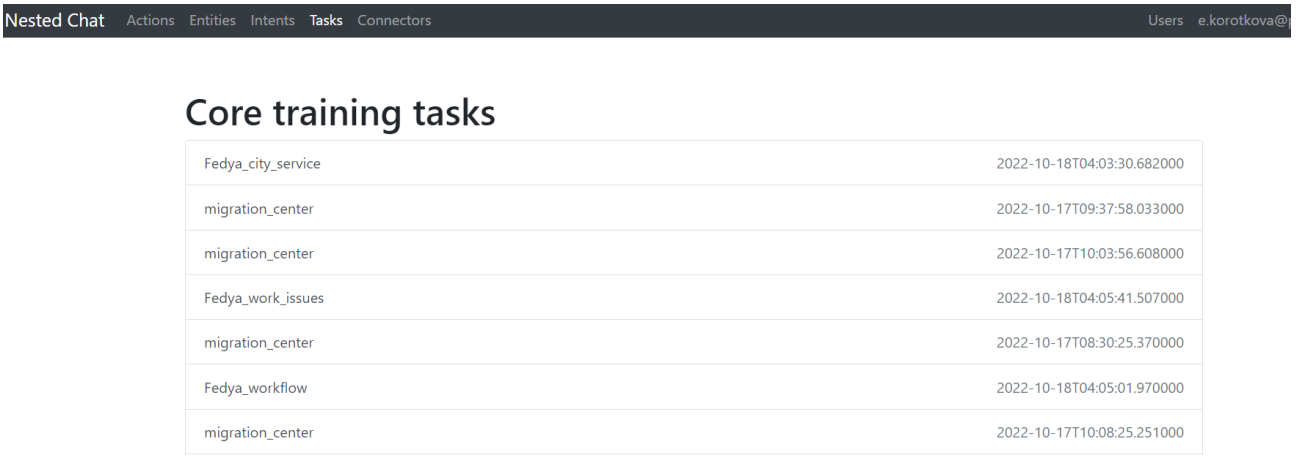
Шаг 3

Используйте ваши интенты в сценариях. В поле ввода **queries** вы можете использовать интент, чтобы не перечислять все возможные варианты. Для этого введите ~, чтобы открылся список со всеми интентами.



Tasks

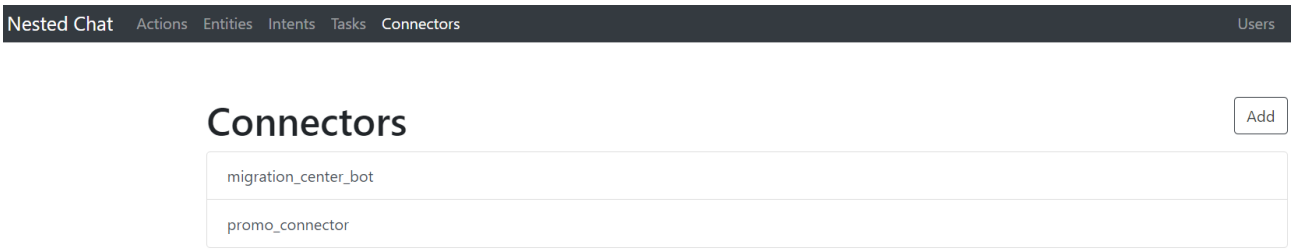
Core training tasks – вкладка, где отражены запущенные процессы на платформе Promobot Nested Chat.



Connectors

Коннекторы – связь вашего сервиса с внешними интерфейсами.

Коннекторы позволяют выводить ваших чат-ботов в неограниченное количество каналов. С этих каналов получаются сообщения и на ваши сервисы отправляется обработанный запрос в виде строки, ссылки или переменной.

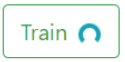


Пример коннектора для телеграмма. Обратите внимание, что в код необходимо вставить токен бота, который можно получить при создании бота в телеграме и идентификационный номер ядра, которое необходимо подключить.

```
#deps:pyTelegramBotAPI

import telebot

from app.sdk import Connections

TOKEN = "token"
bot = telebot.TeleBot(TOKEN, parse_mode=None)
core_id = "core-id"
connections = Connections(core_id)

@bot.message_handler(func=lambda message: True)
def proxy(message):
    chat_id = message.chat.id
    response = connections.response(chat_id, message.text, tol=0.7)
    bot.send_message(chat_id, response.replica)

bot.infinity_polling()
```

Запуск обновления сценариев ядра (релиз)

Чтобы получить возможность тестирования историй ядер и выведения данных в чат-бот или на робота, необходимо перевести бегунок “on” в активное положение. Для этого нажмите кнопку **«Train».**



В кнопке **«Train»** появится спиннер. Он будет крутиться, пока ядро не обучится на данных ядра, при этом вы можете закрывать и перезагружать страницу браузера. После тренировки данные (стори) ядра “выйдут в релиз”, то есть вы сможете их потестировать.

Тестирование ядра в сервисе Nested Chat

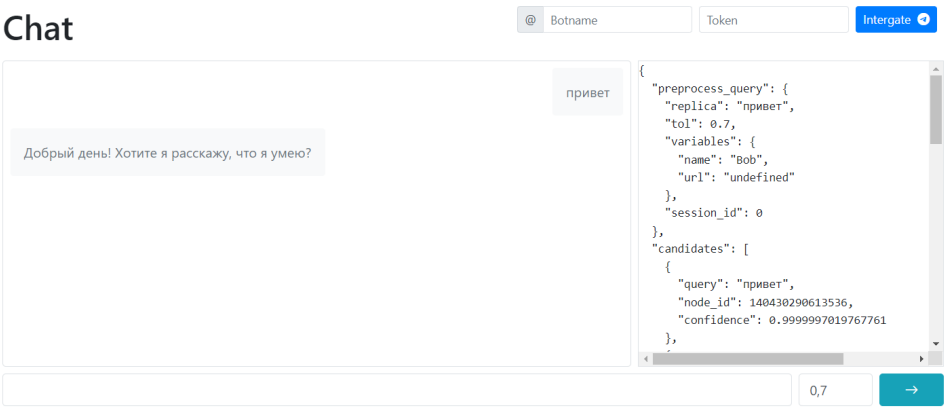
После вывода ядра в релиз можно проверить корректность работы сценариев путем тестового разговора с ботом. Для этого необходимо зайти в ядро и нажать кнопку **«Chat»**

После нажатия откроется окно чат-бота, в поле можно записывать запрос, и отправлять его чат-боту при помощи клавиши со стрелкой. Также здесь можно указать значение «confidence» (степень совпадения распознанной фразы с фразой истории) Чем меньше значение, тем меньше соответствий может быть у фразы сценария с входящей фразой (по умолчанию 0,7)

Во время общения в чате справа появляется окно с информацией из логов.

Перечень информации о входящем запросе

- replica – запрос-реплика, которую вы отправили
- tol (tolerance) – коэффициент, который выставлен в чате
- variables – поле переменных и их значения
- session_id – идентификационный номер сессии пользователя



Перечень информации о кандидатах – список queries в узлах стори, которые подходят под наш запрос.

- query – реплика узла, которую модель рассматривает в качестве кандидата
- node_id – идентификационный номер узла, в котором лежит query-кандидат

- confidence – вероятность соответствия запроса-реплики (replica) реплике узла (query), высвечиваются только те, которые соответствуют параметру tolerance (больше или равны этому значению)


```
"candidates": [
  {
    "query": "привет",
    "node_id": 140430290613536,
    "confidence": 0.9999997019767761
  },
  {
    "query": "здравствуйте",
    "node_id": 140430290613536,
    "confidence": 0.9693987369537354
  },
  {
    "query": "добрый день",
    "node_id": 140430290613536,
    "confidence": 0.8372535705566406
  },
```

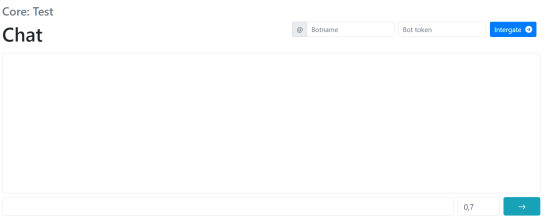

Перечень информации о response – об ответе:

- replica – выведенная реплика
- confidence – вероятность, с которой запрос соответствует одной из query
- session_id – идентификационный номер сессии пользователя
- variables – поле переменных и их значения


```
"response": {
  "replica": "Добрый день! Хотите я расскаж
  "confidence": 0.9999997019767761,
  "session_id": 140430115167920,
  "variables": {
    "name": "Bob",
    "url": "undefined"
  }
"repeat": false,
"postprocessed_response": {
  "replica": "Добрый день! Хотите я расска»
  "confidence": 0.9999997019767761,
  "session_id": 140430115167920,
  "variables": {
    "name": "Bob",
    "url": "undefined"
  }
}
```


Вывод ядра на телеграм-бота

Вывести ядро в чат-бот телеграма можно в разделе **«Chat».** Необходимо ввести в поле **Botname** (первое поле) имя бота без «@», а в поле **Bot token** (второе поле) - токен бота. Чтобы зарегистрировать бота для телеграма воспользуйтесь ссылкой https://t.me/BotFather.



После добавления данных, ссылка на подключенный бот будет отображаться на экране. Чтобы отключить бота, нажмите на красную кнопку с изображением корзины.

Формат поддерживаемых данных и доступ к API

Выгружать/загружать информацию, которая находится в ядре можно в формате .json

Перечень доступной для выгрузки информации

Название	Тип данных	Информация
response	str (строка)	ответ робота
confidence	num (числовое значение)	значение от 0 до 1, степень совпадения распознанной фразы с фразой в истории (сейчас этот показатель равен 0,7)
session_id	num (числовое значение	пользовательский id (опционально)
action	str (строка)	скрипт действия (опционально)
url	str (строка)	ссылка на сайт (опционально)
emotion	num (числовое значение)	номер эмоции робота
variables	dict (словарь – «ключ»: значение)	глобальный контекст (опционально)
delay	num (числовое значение)	задержка выдачи/принятия ответа

Доступ к API

Внутренний адрес API для использования в компании: http://192.168.88.24:3000/ (доступен проброс туннеля)

Внешний адрес API для удаленного доступа за пределами офиса Promotot: https://vm-chat.promo-bot.ru/login (доступен только с использованием VPN).

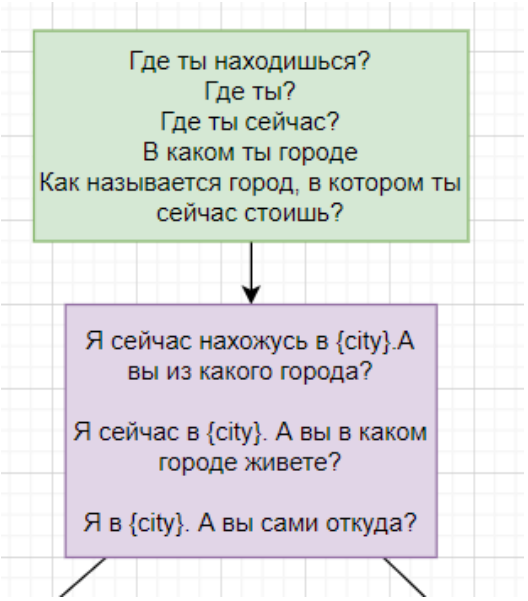
Выгрузка/загрузка готового проекта и диалогов пользователей

Для создания грамотного сценария можно использовать сервис draw io (онлайн или в расширении VSC), это позволит создавать наглядную и удобную схему сценария, кроме того в Nested Chat доступна загрузка файла со сценарием, что существенно сокращает время переноса сценария со схемы в сам сервис.

Для успешной загрузки файла со сценарием в чат необходимо придерживаться правил создания и оформления.

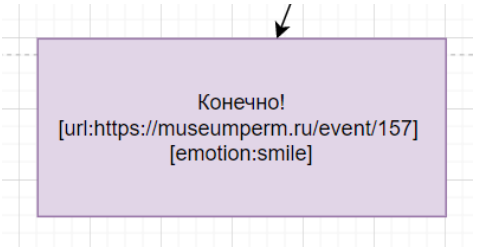
Правила оформления файла .drawio:

1. **Один документ – одно поле – один сценарий – одна верхушка диалога.** Не следует использовать несколько листов в одном документе, а также создавать несколько сценариев в одном поле. Для отрисовки второго сценария необходимо создать второй документ.
2. **Необходимо использовать четкую структуру диалога.** Порядок блоков: пользователь – робот – пользователь – робот и тд.
3. **Сценарий отрисовываем со стрелками.** Стрелки должны четко идти от одного блока ко второму.
4. **Все фразы пишем в одном блоке ответа**, запроса или вопроса (это касается как фраз робота, так и пользователя).



5. **Goto прописывается в отдельном блоке** после ответа робота в формате: goto, пробел, название стори. Пример: goto emotions

6. **Прочие параметры прописываем в этом же блоке** в квадратных скобках с новой строки, через двоеточие, без пробелов. Если не указывать какой-либо из параметров, он не выставится. В эмоциях – стандарт. Список параметров: action, api, emotion, url, strict_mode. *Пример*, [emotion:speak] [strict:true] – при необходимости



Когда сценарий создан и соответствует всем требованиям, мы можем перейти к следующему шагу и переработать его в файл формата .json для загрузки в сервис.

Инструкция по использованию кода:

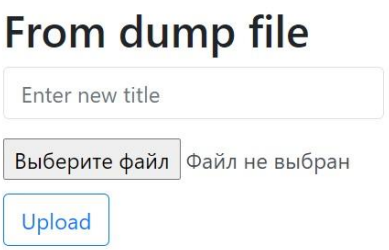
1. Проверить наличие файла с кодом и файла со сценарием в одной папке;
2. Открыть файл конвертера ConverterDrawIOtoJSON.py в VSCode;
3. В массиве «MY_ARGS» написать названия файла со сценарием первым элементом в формате [name].drawio;
4. При необходимости запросить подробную отладочную информацию, оставив в массиве «MY_ARGS» второй элемент «debug». Изменить название второго элемента на «NOdebug», если данная информация не нужна;
5. Открыть папку с обоими файлами в VSCode и запустить код;
6. При успешном результате работы скрипта в папке будет создан файл out, если в схеме сценария были ошибки оформления, то они будут описаны в документе log (при условии запроса отладочной информации).

Выгрузка сценариев ядра

Выгрузить сценарии ядра можно зайдя в само ядро Nested Chat и нажав на кнопку с изображением стрелки в правом верхнем углу:

Загрузка сценариев ядра

Загрузить json файл со сценариями/сценарием ядра можно выбрав файл при создании нового ядра.



Выгрузка логов с пользователями

Для того, чтобы загрузить csv файл с данными о взаимодействии пользователя с сервисом, необходимо нажать на кнопку **«Logs»** в правом верхнем углу ядра:

Также вы можете выгружать полные логи, используя этот код. При запуске не забудьте вписать в код порт Nested Chat и id ядер.

```
# pip install pandas requests loguru openpyxl

import os
import requests
import pandas as pd
from loguru import logger

if not os.path.exists("logs"):
    os.makedirs("logs")


def import_logs_to_excel(url: str, core_ids: list):
    for core_id in core_ids:
        response = requests.get(f'{url}/cores/{core_id}')

        if response.status_code == 200:
            core = response.json()
            logs = get_logs(core_id, url)

            if logs is not None:
                logs_frame = pd.DataFrame(logs)
                logs_frame.to_excel(f"logs/{core['title']}-{core_id}.xlsx")
        else:
            logger.error(f'core_id: {core_id}, detail: {response.text}')


def get_logs(core_id: str, url: str) -> list:
    response = requests.get(
        f"{url}/logs-flatten/{core_id}",
        params={"skip": 0, "limit": 0}
    )

    if response.status_code == 200:
        logs = response.json()
```

```
        if len(logs):
            return logs

    else:
        logger.error(f'core_id: {core_id}, detail: {response.text}')


# Это адрес API Nested-chat. В зависимости от сервера меняется ip и порт
NC_API_URL = "http://localhost:0000"


# Это список id ядер, чьи логи необходимо выгрузить, список можно и нужно редактировать
CORE_IDS = [
    "core-id",
    "core-id"
]

import_logs_to_excel(NC_API_URL, CORE_IDS)
```

Права на продукт и порядок использования лицензии

ООО «ПРОМОБОТ» является автором и разработчиком диалоговой системы «Nested Chat», обладает исключительным правом на программу для ЭВМ «Nested Chat» (далее также – Программа).

Порядок и условия использования программы другими лицами определяются Лицензионным соглашением (договором), заключаемым между ООО «ПРОМОБОТ» (Лицензиар) и Лицензиатом.

В лицензионном соглашении указываются допустимые способы использования программы, территория, на которой допускается использование Программы, срок действия договора, размер и порядок уплаты вознаграждения за пользование программой.

В случае нарушения своих прав, Лицензиар вправе осуществлять защиту своих прав в порядке и способами, предусмотренными законом, в том числе Лицензиар вправе требовать от нарушителя выплаты компенсации за нарушение указанного права без определения размера убытков.

Использование программы без заключения Лицензионного соглашения является нарушением прав Лицензиара.

Список литературы и ссылок

1. Сервис Nested Chat (внешний API: https://vm-chat.promo-bot.ru/; внутренний API: http://192.168.88.24:3000/)
2. Параметры API, необходимые для интеграция сервиса (http://192.168.88.24:8010/docs#/)
3. Список моделей, доступных для использования в Nested Chat (обязательно наличие языкового энкодера) (https://huggingface.co/models)
4. Информация о сервисе draw io (https://drawio-app.com/)

Лингвистические процессоры

Список лингвистических процессоров и игровых приложений, планируемых к интеграции в Nested Chat:

1. **Обработка гонорати**ва. Возможность смены неформального обращения робота с «ты», на «Вы» во всех ответах*
2. **Обработка рода.** Возможность смены пола робота (в речи) с мужского на женский*
3. **Обработка эллипсиса/анафоры**. Возможность восполнения недостающей (подразумеваемой) информации*
4. **Спеллчекер**. Исправление орфографических ошибок и опечаток.*
5. **Модуль восстановления запроса.** Для случаев когда ASR работает некорректно.*
6. **Синонимайзер**. Возможность подключения парафразера к полю запрос или полю ответ.*
7. **Обработка эмоционального состояния (Sentiment Analysis)**. Возможность подключения обработки эмоционального состояния человека по распознанному тексту.
8. **Генератор персонажей**. Возможность подключения модуля автоматической генерации персонажа.*
9. **Конвертер сценариев.** Возможность подключения модуля конвертации сценариев DraIO в JSON файлы загрузки NC*
10. **Модуль обработки паттернов.** Возможность подключения паттернов синонимов для извлечения запроса пользователя*
11. **Модуль обработки газеттиров.** Возможность подключения пользовательских списков для извлечения запроса пользователя
12. **Модуль обработки именованных сущностей.** Возможность подключения NER на основе ML и REGEXP
13. **Модуль обработки мультимодальности.** Для многоканального распознавания (жесты, текст, изображения, действия…)*
14. **Модуль обратного перевода.** Для конвертации перевода ядра в иностранный язык.*
15. **Модуль онтологий.** Для формирования ответа на основе логического вывода.*
16. Игровые приложения:
    - «**Порекомендуй фильм**» – опираясь на предпочтения говоряшего робот рекомендует к просмотру фильм
    - «**Собери робота**» – мозаика с изображением робота, которую нужно собрать
    - «**Узнай свою судьбу**» – генератор предсказаний
    - «**Загадай животное**» – робот предлагает загадать животное, а затем задает наводящие вопросы, пытаясь угадать животное
    - «**Виселица**» – робот загадывает слово, у пользователя есть ограниченное число попыток, чтобы угадать слово по одной букве
    - «**Гороскоп**» – робот дает астрологический прогноз
    - «**Минутка релакса**» – робот рисует на экране фракталы
    - «**Камень-ножницы-бумага**» – игрок выбирает один из трех вариантов, чтобы сразиться с роботом
    - «**Города**» – игроки по очереди называют города мира на последнюю букву предыдущего города, побеждает тот, кто последним назовет город
    - «**Поздравление**» – робот читает поздравление по запросу пользователя
    - «**Подсчет индекса массы тела**» – робот запрашивает у пользователя рост, вес. Подсчитывает индекс массы тела и дает рекомендации по питанию.
    - «**Загадки**» – робот загадывает загадки по запросу пользователя
    - «**Миф или правда**» – робот читает утверждение и предлагает пользователю угадать, миф это или правда
    - «**Угадай столицу**» – робот называет страну, а пользователю назвать столицу из двух предложенных вариантов.
    - «**Калькулятор**» – по запросу пользователя робот может производить арифметические действия с числами
    - «**Точная дата и время**» – по запросу пользователя робот называет точную дату и время
    - «**Жизнь**» – робот запускает математическую компиляцию развития жизни

- «**Быки и коровы**» – робот загадывает четырехзначное число и предлагает пользователю угадать, какое именно. В качестве подсказок он называет количество коров (цифр которые совпадают с теми, которые есть в числе робота, но которые находятся не на своих местах) и быков (цифр, которые находятся на своих местах).
- «**Генератор стихов**» – робот запрашивает у пользователя существительное, затем подбирает ему рифму и добавляет в известный стих, создавая забавные четверостишья.
- «**Сделать из мухи слона**» – у пользователя есть возможность сделать из слова «муха» слово «слон», меняя за ход только одну букву.
- «**Тест на стресс**» – задавая пользователю вопросы о его рабочем дне, робот делает замер уровня стресса и дает советы.
- «**Офис**» – квест офисного сотрудника
- «**Кошелек или жизнь**» – квест посвященный Хэллоуину
- «**Спасение свиней**» – философский квест про свиней
- «**Музыкальный квест**» – квест по сбору музыкальной группы и организации концерта
- «**Каникулы**» – квест по выбору места для отдыха

Приложение 1

Схема интеграции на робота

Параметры:

- nested_chat/confidence = 0.8
- nested_chat/core_id = id ядра с которым нужно работать
- nested_chat/url = http://localhost:8009

Подготовка:

- Установить docker https://docs.docker.com/engine/install/ubuntu/
- Создать директорию .backends в /home/promobot/
- Скачать и распаковать архив в ~/.backends wget -r ftp://master:Q5vCR1fC8ucu@192.168.88...ested_Chat
- Положить дамп ядра в ~/.backends/nested-chat/data

Шаги:

1) Создать подсеть для работы контейнеров (nested chat, sentiment, actions, duckling):

```
docker network create --subnet=172.19.0.0/16 nested-chat
```

OP: Отобразился id

2) Перейти в директорию ~/.backends/sentiment-master/app и собрать sentiment:

```
docker build -t sentiment
```

OP: Successfully built ******

Successfully tagged sentiment:latest

3) Запустить sentiment с указанием адреса в подсети созданной выше:

```
docker run --net nested-chat --ip 172.19.0.2 -d --restart always -p
5004:5000 -v ~/.backends/sentiment-master/model:/app/model sentiment
```

OP: Отобразился id

4) Перейти в директорию ~/.backends/nested-chat и загрузить образ nested-chat-engine.tar.gz:

```
docker load -i nested-chat-engine.tar.gz
```

OP: Loaded image: nested-chat-engine:latest

5) Запустить nested-chat-engine с указанием адреса в подсети созданной выше:

```
docker run --net nested-chat --ip 172.19.0.3 -d --restart always -p
8009:8000 -v ~/.backends/nested-chat/data:/ds/data -v
~/.backends/nested-chat/.cache:/ds/.cache -e LOCAL_MODE=1 -e
APP_PORT=8000 -e ACTION_SERVER=http://172.19.0.4:8000 -e
DUCKLING_URL=http://172.19.0.5:8000 -e
SENTIMENT_URL=http://172.19.0.2:5000 nested-chat-engine
```

OP: Отобразился id

6) Запустить nested-chat-actions с указанием адреса в подсети созданной выше:

```
docker run --net nested-chat --ip 172.19.0.4 -d --restart always -p
8104:8000 -v
~/.backends/nested-chat-actions/app/actions:/ds/app/actions -e
PORT=8000 -e INTERNET_SEARCH_URL=http://192.168.88.24:8011/search
nested-chat-actions
```

OP: Отобразился id

7) перейти в ~/.backends/nested-chat-actions и распаковать actions.rar в:

```
~/.backends/nested-chat-actions/app/actions
```

OP: Содержимое архива распаковано

8) Загрузить и запустить duckling

```
docker pull rasa/duckling
```

```
docker run --net nested-chat --ip 172.19.0.5 -d --restart always -p
8000:8000 rasa/duckling
```

OP: Отобразился id

9) Поверить что контейнеры запущены

```
docker ps
```

OP:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
*********	nested-chat-engine	"python run.py"	** ago	Up **	80/tcp, 0.0.0.0:8009->8009/tcp	*****
*********	sentiment	"python app.py"	** ago	Up **	0.0.0.0:5004->5000/tcp	*****
*********	nested-chat-actions	"python run.py"	** ago	Up **	80/tcp, 8012/tcp, 0.0.0.0:8104→8000/tcp	*****
*********	rasa/duckling	"duckling-example-ex…"	** ago	Up **	0.0.0.0:8000->8000/tcp	*****

Приложение 2

Описание полей API

root

GET/	получение адреса расположения сервиса

cores

Название параметра	Значение	Уточнения
GET/cores	получение адреса расположения ядра	
POST/cores	получение информации о текущих ядрах	
DELETE/cores/{core_id}	получение информации об удаленных ядрах	
GET/cores/{core_id}/relevance	получение информации о ядрах в релизе	на данный момент параметр отключен ввиду высокого потребления памяти
POST/cores/{core_id}/reload	получение запроса на перезагрузку ядра	**tol (tolerance)** - контролирует баланс между точностью (precision) и полнотой (recall) Речь идет про ложные срабатывания **variables** - глобальный контекст (принимает набор только тех ключей, которые объявлены в настройках ядра) **delay** - параметр ожидания ответа пользователя (на текущий момент устанавливается только по умолчанию с параметром 0)
POST/cores/{core_id}/response	получение запроса на выдачу ответа ядра	

entities

GET/entities	получение информации о текущих сущностях ядра
POST/entities	получение запроса на загрузку сущностей в ядро
POST/entities/reload	получение запроса на перетренировку ядра с загруженными сущностями

Основные эндпойнты перечислены по ссылке http://192.168.88.24:8102/docs#/, все управляются через фронт, кроме параметра **POST/cores/{core_id}/response**. Порядок установлен при описании параметров ядра по ссылке, прикрепленной выше.

Приложение 3

Схемы организации кода экшена для использования в сценариях Nested Chat

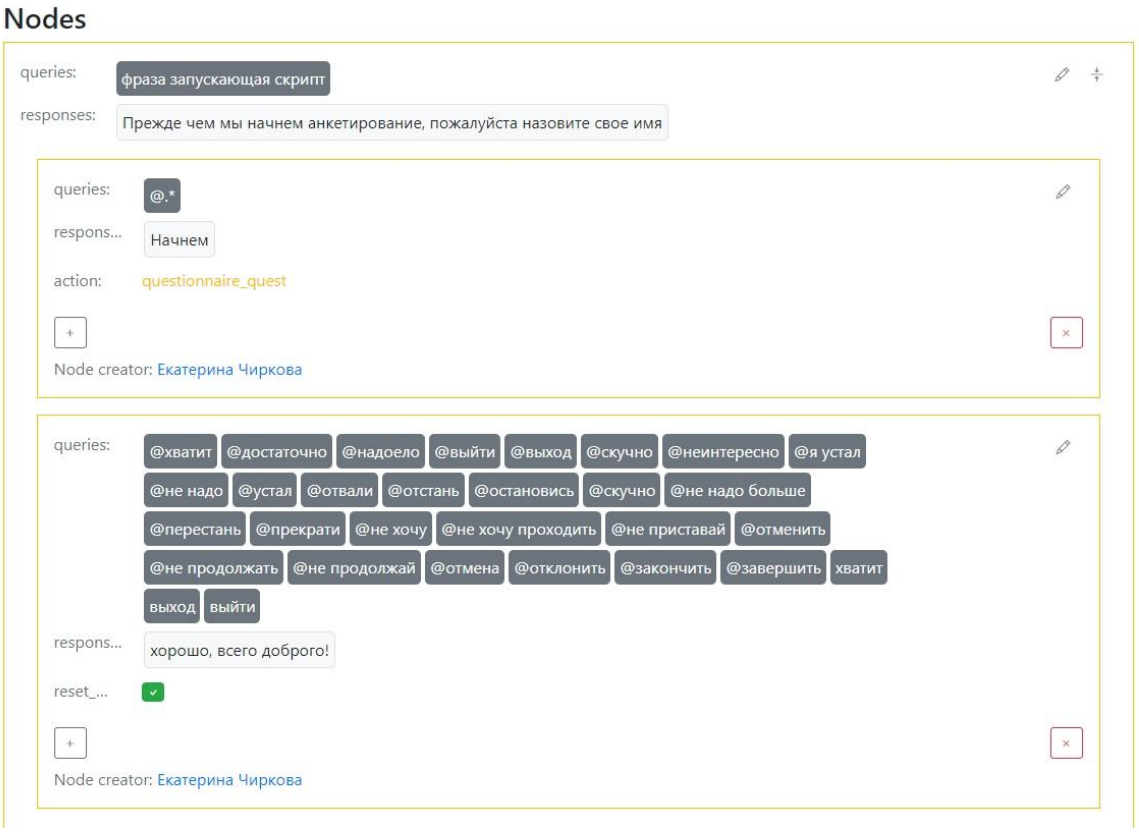
1. **_quest**

```
from app.sdk import ActionRequest, ActionResponse

__scores = {} #словарь для записи баллов набранных в ходе анкетирования
__number = {} #запись порядкового номера вопроса

#массив вариаций ответа для первого варианта
answer_1 = ["yes", "sure"]
#массив вариаций ответа для второго варианта
answer_2 = ["no", "nope"]

#список вопросов теста
QUESTIONS = [
    'вопрос №1',
    'вопрос №2'
]

#информация о начислении баллов
SCORES = [
    {INTENT_YES: 0, INTENT_NO: 1},
    {INTENT_YES: 1, INTENT_NO: 0}
]

#получение порядкового номера вопроса
def get_question(q_number: int):
    return QUESTIONS[q_number - 1]

#получение балла за ответ. intent = INTENT_YES/INTENT_NO
def get_score(q_number: int, intent: str):
    return SCORES[q_number - 1][intent]

#вывод результата
def get_result(scores):
    if 0 <= scores <= 3:
        return 'Разъяснение результата 1'
    elif 4 <= scores <= 7:
        return 'Разъяснение результата 2'
    elif scores >= 8:
        return 'Разъяснение результата 3'

#определение, есть ли еще вопросы в списке или можно выводить результат
def get_response(intent, number, scores):
    if number > 0 and number <= len(QUESTIONS):
```

```
            scores += get_score(number, intent)
        number += 1
        if number > len(QUESTIONS):
            response = get_result(scores)
        else:
            response = ''
            if number == 1:
                response = 'итак начнем, я задам вам ' + str(len(QUESTIONS)) + '
вопросов, в любой момент можете сказать "хватит".'
            response += get_question(number)
        return response, number, scores


def function_name_quest(request: ActionRequest) -> ActionResponse:
    try:
        #записываем id пользователя
        session_id = request.session_id
        try:
            #если в течение одной сессии мы обращаемся не первый раз, то
запрашиваем данные о номере вопроса и баллах за ответы, которые были записаны
шагом ранее
            number = __number[session_id]
            scores = __scores[session_id]
        except:
            #если данных нет, то создаем переменные с начальными значениями
            number = 0
            scores = 0
        #если вариант ответа не совпадает с допустимыми, то сервис просит
уточнить ответ
        if number != 0 and str(request.query) not in answer_1 and
str(request.query) not in answer_2:
            return ActionResponse(message="Не совсем понял. вариант 1 или
вариант 2?")

        else:
            if str(request.query) in answer_1:
                intent = INTENT_YES
            elif str(request.query) in answer_2:
                intent = INTENT_NO
            else:
                intent = "trash"

        response, number, scores = get_response(
                    intent, number, scores
                )
        #записываем новые данные о номере вопроса и баллах
        __number[session_id] = number
```

```
        __scores[session_id] = scores

        #отправляем сообщение в чат
        return ActionResponse(message= str(response))
    except Exception as error:
        return ActionResponse(message = f'Error: {error}, {request.variables}')
```

Для того чтобы иметь возможность запускать скрипт столько раз, сколько нужно, пока пользователь не ответит на все вопросы, или пока не откажется от викторины/анкетирования, оформлять историю с использованием данного типа экшена нужно следующим образом:



2. **_game**

```
from app.sdk import ActionRequest, ActionResponse

#при необходимости можно создавать словари для хранения информации по прохождению скрипта (см. экшен типа "_quest")

def function_name_game(request: ActionRequest) -> ActionResponse:
    try:
        #код
       answer = "текст ответа"
        return ActionResponse(message= str(answer))
    except Exception as error:
        return ActionResponse(message = f'Error: {error}, {request.variables}')
```

Если игровой экшен подразумевает многократный запуск, то оформлять в истории его необходимо также как

скрипт типа “_quest”. Если экшен запускается только один раз, его нужно прикрепить к тому уровню истории, на котором он запускается.

3. **_generation**

```
from app.sdk import ActionRequest, ActionResponse

def function_name_generation(request: ActionRequest) -> ActionResponse:
    try:
        #если необходимо работать с фразой, которую отправил пользователь, то ее можно записать в переменную
        users_phrase = request.query
        #код
       answer = "текст ответа"
        return ActionResponse(message= str(answer))
    except Exception as error:
        return ActionResponse(message = f'Error: {error}, {request.variables}')
```

Экшен нужно прикрепить к тому уровню истории, на котором он запускается.

4. **_validity**

```
from app.sdk import ActionRequest, ActionResponse

def function_name_validity(request: ActionRequest) -> ActionResponse:
    try:
       #в переменную phrase_for_check записываем фразу пользователя
        phrase_for_check = request.query
        #код

        #если необходимо вывести результат проверки, сохраняйте его в переменную response
        #если экшен не подразумевает отправку результата, то message необходимо оставить пустым
        return ActionResponse(message= str(response))
    except Exception as error:
        return ActionResponse(message = f'Error: {error}, {request.variables}')
```

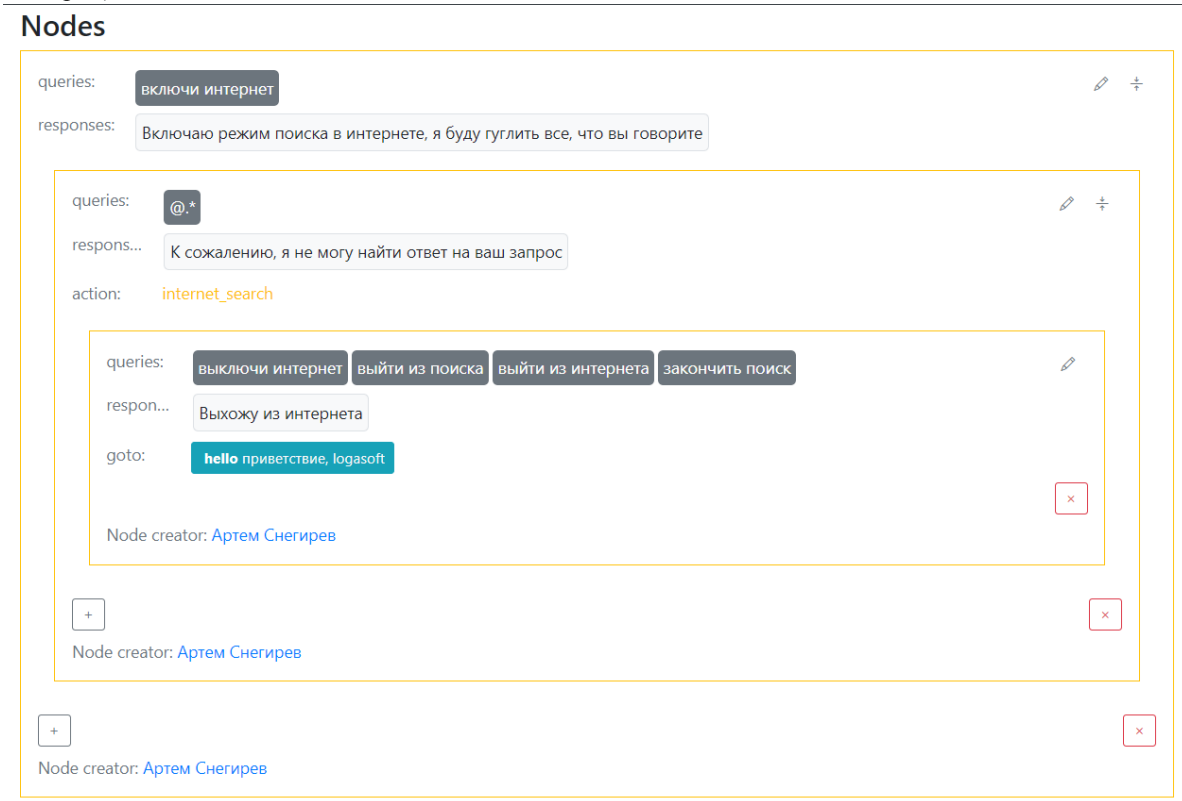
Экшен нужно прикрепить к тому уровню истории, на котором он запускается.

5. **_search**

```
from app.sdk import ActionRequest, ActionResponse

def function_name_search(request: ActionRequest) -> ActionResponse:
    try:
        #в переменную users_phrase записываем фразу пользователя, по которой будет осуществляться поиск
        users_phrase = request.query
        #код
```

```
        #результат проверки сохраняем в переменную response
        return ActionResponse(message= str(response))
    except Exception as error:
        return ActionResponse(message = f'Error: {error}, {request.variables}')
```

Сценарий с использованием экшена “_search” также как и экшен типа “_quest” должен быть организован таким образом, чтобы пользователь мог отправлять столько запросов на поиск, сколько ему требуется. Например скрипт с поиском информации в интернете выглядит таким образом (в графе goto может быть ссылка на любой из сценариев ядра, также существует возможность стирания контекста (reset_context) на фразе выхода из сценария):



6. **_extract**

```
from app.sdk import ActionRequest, ActionResponse

__info = {} #словарь для записи необходимой информации

def function_name_extract(request: ActionRequest) -> ActionResponse:
    try:
        #записываем id пользователя
        session_id = request.session_id
       #phrase - фраза пользователя, из которой можно извлечь информацию
        phrase = request.query
        #код
       #извлеченную информацию записываем сначала в переменную new_information,
а затем в словарь _info
        __info[session_id] = str(new_information)
        return ActionResponse(message="")
    except Exception as error:
```

```
        return ActionResponse(message = f'Error: {error}, {request.variables}')
```

Экшен нужно прикрепить к тому уровню истории, на котором он запускается. Обратите внимание на то, что экшен типа “_extract” подразумевает сбор информации, но фраза для ответа пользователю должна записываться не в message экшена, а в самом сценарии ядра.

7. **_postprocessor**

```
from app.sdk import ActionRequest, ActionResponse

def function_name_postprocessor(request: ActionRequest) -> ActionResponse:
    try:
       #ответ по умолчанию записываем в переменную default_answer
        default_answer = str(request.query)
        #код обрабатывающий стандартный ответа сервиса
       #обработанный ответ сервиса сохраняем в переменную final_answer

        return ActionResponse(
        message= str(final_answer),
        variables=request.variables
        )

    except Exception:
        #custom code (exception handling)
        pass
    else:
        return ActionResponse()
```

Ссылка на экшен типа “_postprocessor” прописывается в графе “postprocessors” в настройках ядра и будет обрабатывать все ответы сервиса.

8. **_preprocessor**

```
from app.sdk import ActionRequest, ActionResponse

def function_name_preprocessor(request: ActionRequest) -> ActionResponse:

    try:
       #ответ пользователя записываем в переменную default_answer
        default_answer = str(request.query)
        #код обрабатывающий ответ пользователя
       #обработанный ответ сервиса сохраняем в переменную final_answer

        return ActionResponse(
            message= str(final_answer)
            )

    except Exception:
        #custom code (exception handling)
        pass
```

```
    else:
        return ActionResponse()
```

Ссылка на экшен типа “_preprocessor” прописывается в графе “preprocessors” в настройках ядра и будет обрабатывать все ответы пользователя.