Санкт-Петербургский государственный университет

ТРОФИМОВ Всеволод Алексеевич

Выпускная квалификационная работа

Инструмент создания интерфейсов для краудсорсинга без навыков разработки

Бакалавриат:

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа *СВ.5003.2017*
«Программирование и информационные технологии

Профиль «Автоматизация научных исследований»

Научный руководитель:

доцент кафедры технологии программирования,

кандидат технических наук,

Блеканов Иван Станиславович

Рецензент:

доцент кафедры компьютерных технологий и систем,

кандидат физико-математических наук,

Коровкин Максим Васильевич

Санкт-Петербург

2021 г.

СОДЕРЖАНИЕ

Введение

Актуальность

Первое популярное определение краудсорсинга дал J. Howe в 2006 году. Термин «краудсорсинг» был введен как «the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call.» – акт переноса организацией или общественным институтом функции, ранее выполняемой специально назначенным агентом (обычно сотрудником), на заранее не определенную, как правило большую, группу исполнителей в формате открытого призыва.[1]

Краудсорсинг успешно применялся разными способами для решения множества практических задач, например:

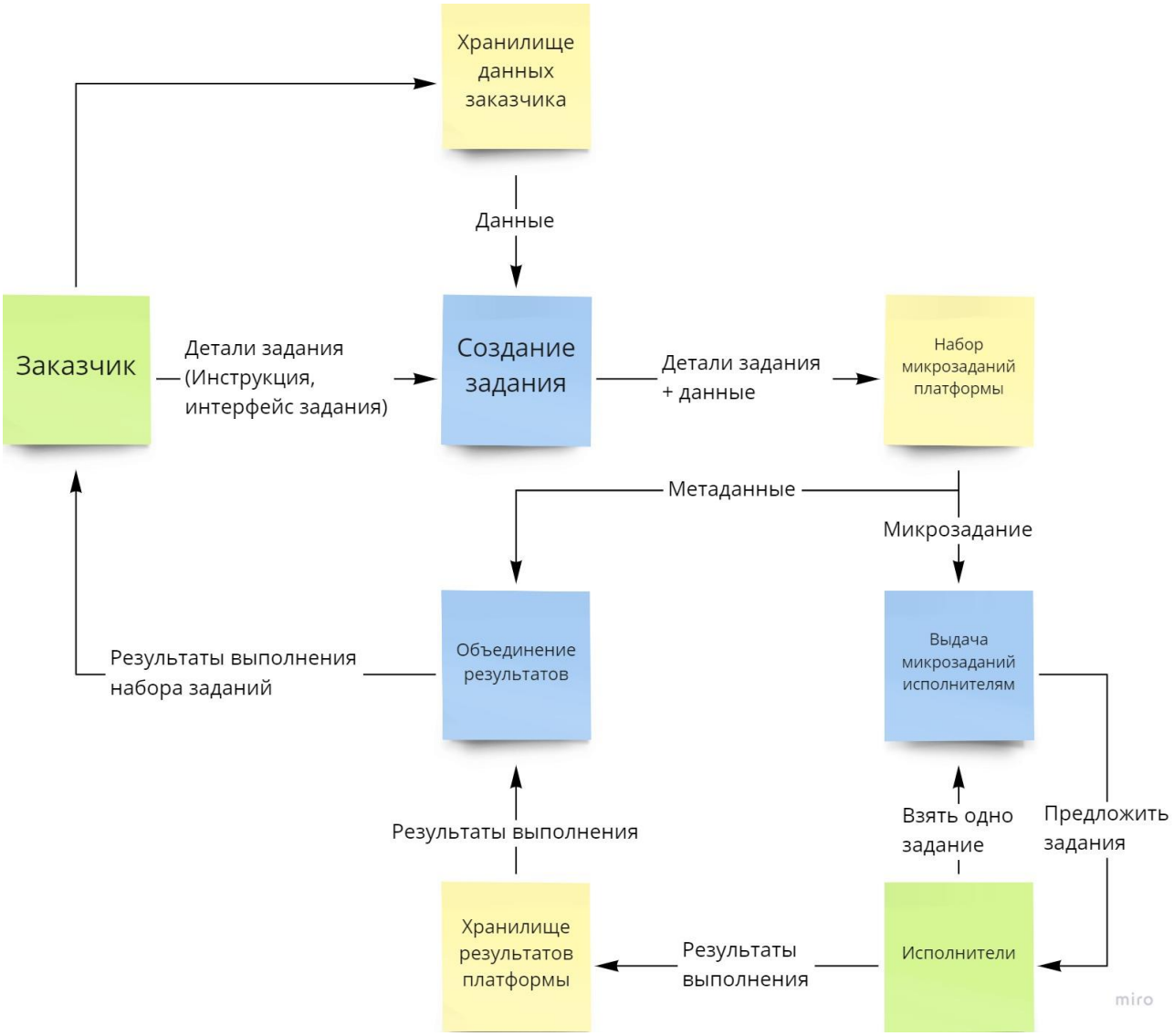
1. Оценка релевантности поисковой выдачи [2][3][4]
2. Сбор данных о содержимом изображений [5]
3. Оценка качества перевода [6][7]
4. Оценка пригодности изображений для просмотра детьми [8]
5. Аннотация текста для задач NLP (natural language processing – обработки естественного языка) [9]
6. Оценка качества видео [10]
7. Оценка удобства пользовательского интерфейса [11]
8. Систематизация научной литературы [12]
9. Семантическая иерархическая кластеризация [13]

Основные методы мотивации исполнителей выполнять краудсорсинг задания – социальная [5][14][18] и финансовая награды. Методы краудсорсинга,

привлекающие исполнителей с финансовой наградой, а точнее – микроплатежами за каждое выполненное задание, считаются подходящими для более широкого класса задач и более простыми в реализации [2].

На май 2021 года “Microtask crowdsourcing” – способ применения краудсорсинга, основанный на выдаче небольших заданий, оплачиваемых микроплатежами, уже выделился в индустрию и широкое направление для исследований [15][16].

Как правило microtask crowdsourcing используется посредством платформ краудсорсинга таких как Amazon Mechincal Turk, Яндекс.Толока или Appen. У этих платформ есть 2 основных типа пользователей: заказчики и исполнители. Заказчики создают оплачиваемые микрозадания для исполнителей. Исполнители выполняют эти микрозадания за финансовую награду. Процесс взаимодействия платформы, заказчика и исполнителя можно описать схемой, приведенной на рисунке ниже.



Обобщенная схема взаимодействия краудсорсинг платформы, заказчика и исполнителя [17]

В области microtask crowdsourcing существует множество задач, требующих дополнительного исследования, например:

1. Определение оптимальной денежной компенсации за выполнение задания [19]
2. Эффективное распределение задач по исполнителям [8]
3. Удержание и повышение вовлеченности исполнителей [20]

4. Агрегация результатов разметки [21][22][23]
5. Обнаружение мошенничества среди исполнителей [24][25]
6. Упрощение для заказчика создания заданий краудсорсинга [26][27]
7. Повышение качества интерфейсов и инструкций для заданий краудсорсинга [11] [28] [29][30]

Цель работы

Цель данной работы – упростить для заказчиков создание интерфейсов заданий краудсорсинга посредством конструктора интерфейсов. На момент начала работы, заказчики, использующие Amazon Mechanical Turk и Яндекс.Толоку вынуждены вручную создавать интерфейсы разметки на веб технологиях, хотя по внутренним данным Яндекс.Толоки подавляющее большинство заказчиков не имеет навыков дизайна и веб разработки.

Это позволит заказчикам быстрее и с меньшими усилиями создавать задания, и сэкономить благодаря тому, что они решат задачу используя краудсорсинг [31][32].

Платформе результат данной работы позволит централизованно контролировать и улучшать качество интерфейсов заданий.

Постановка задачи

Для достижения поставленной цели в работе выделены следующие задачи

- Обзор предметной области интерфейсов краудсорсинга и существующих способов их создания
- Разработка программного комплекса для создания интерфейсов краудсорсинга на основе веб технологий
- Тестирование разработанного решения на реальных заказчиках

Глава 1. Обзор

Первый шаг работы – выделить особенности интерфейсов краудсорсинга относительно классических интерфейсов приложений. Это позволит точнее сформулировать требование к поиску или разработке решения.

1.1 Обзор научных источников

Существует 2 заметно различающихся направления в интерфейсах краудсорсинга: классические и разговорные. Разговорные интерфейсы как правило представленные в формате чата, который дополнительно может быть обогащен медиа контентом и кнопками для ввода быстрого ответа.

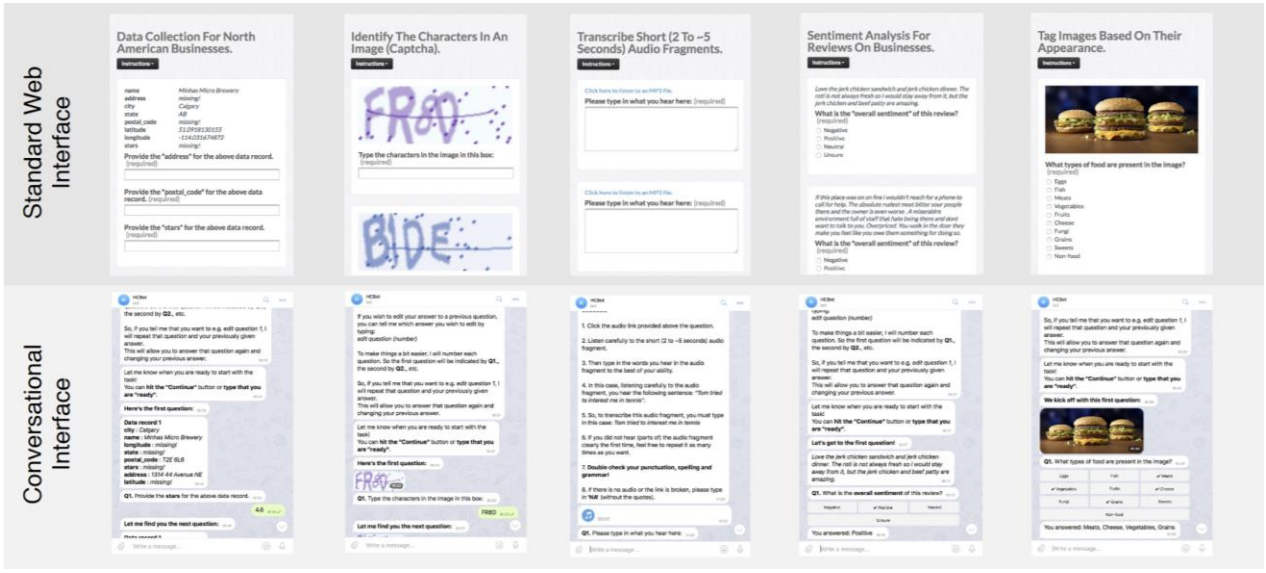


Рис. 1.1 – представление одних и тех же заданий задания в классической(сверху) и разговорной(снизу) форме [33]

В процессе обзора было принято решение не рассматривать разговорные интерфейсы как основной формат результата работы инструмента поскольку, по данным исследования проведенного в Delft University of Technology, разговорные интерфейсы показывают в среднем худшие результаты в сравнении с классическими интерфейсами [33]

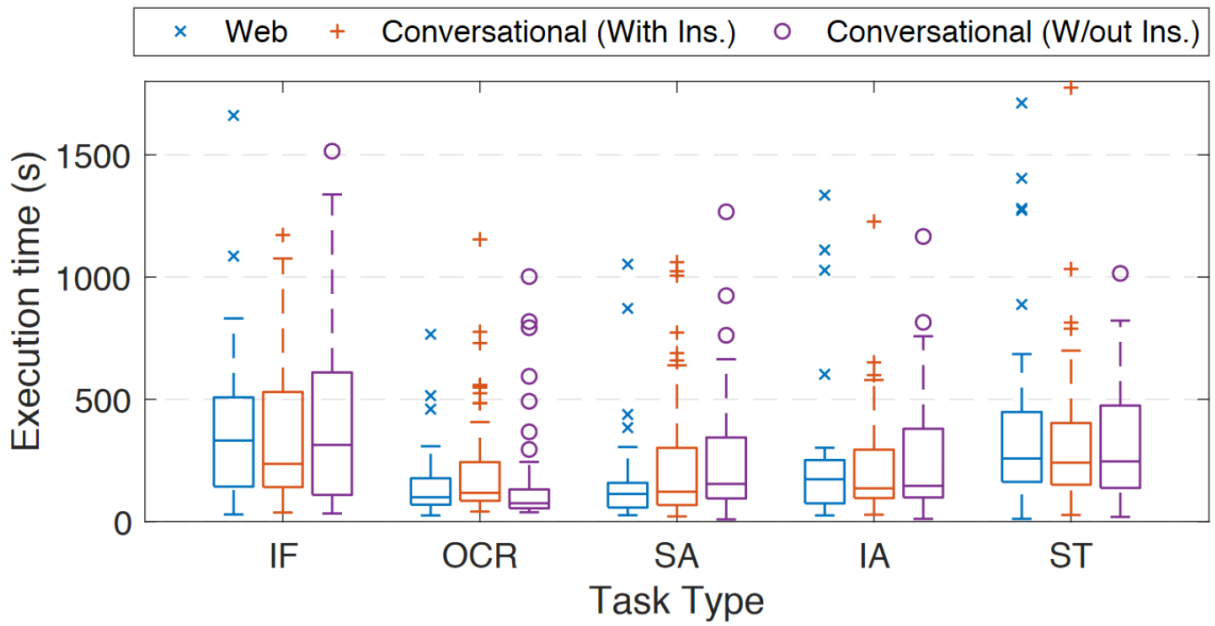


Рис. 1.2 – среднее время выполнения задания в зависимости от формата [33]

Table 3: Quality of crowdwork produced across different task and interface types.

Task type	Web	Conversational
Information Finding	0.95	0.92
Human OCR	0.75	0.82
Speech transcription	0.85	0.75
Sentiment analysis	0.93	0.88
Image annotation	0.90	0.81

Рис. 1.3 – среднее качество выполнения в зависимости от формата [33]

Table 4: Ratings of workers satisfaction. OV: Overall; IN: Instruction; EA: Ease of Job; PA: Pay

Task type	Platform	OV	IN	EA	PA
Information Finding	Web	4.5	4.3	4.5	4.5
	Conversational	3.0	3.4	2.9	3.3
Human OCR	Web	4.3	4.2	4.0	4.5
	Conversational	3.4	4.0	3.8	4.3
Speech Transcription	Web	4.7	4.7	3.9	4.1
	Conversational	4.1	4.5	4.0	4.3
Sentiment Analysis	Web	4.3	4.3	4.1	4.1
	Conversational	3.7	3.4	3.2	3.8
Image Annotation	Web	3.8	3.8	3.3	3.8
	Conversational	3.7	3.9	3.1	3.9

Рис 1.4 средняя удовлетворенность работников в зависимости от формата [33]

Существуют исследования, указывающие на то, что интерфейсы заданий – перспективное направление исследований в области краудсорсинга [28] и значимо влияет на результат разметки [29]. Также существует исследование посвященное непосредственно улучшению интерфейса, но в нем рассматривается выполнение лишь одного задания, а каждая версия включает в себя множество изменений, что не позволяет измерить эффект каждого отдельного изменения. Многие изменения описаны слишком абстрактно, чтобы извлечь из этого практическую пользу, например: «Simplify the instruction to make it easier to understand.» (упростили инструкцию, чтобы ее было проще понять) [34].

К несчастью, исследований посвящённых непосредственно упрощению создания интерфейсов заданий краудсорсинга или выделению особенностей таких интерфейсов найти не удалось.

1.2 Исследование особенностей интерфейсов краудсорсинга

В рамках данной работы было проведено исследование интерфейсов краудсорсинга на предмет нескольких аспектов:

1. Отсутствия решений, часто используемых в классических веб-интерфейсах приложений
2. Решений, встречающихся значительно чаще, относительно интерфейсов классических веб приложений
3. Существуют ли ярко выраженные классы задач, в которые попадает не менее 60% интерфейсов заданий. Т.е. рентабельно ли будет заранее создать интерфейсы заданий на стороне платформы, а не перекладывать эту задачу на заказчика.

В качестве данных для исследования были использованы интерфейсы заданий из сервиса Яндекс.Толока. В выборку попадали задания, которые за последний месяц были выполнены минимум один раз.

Для формирования первичных гипотез вручную были посмотрены и описаны 100 заданий. Для просмотра интерфейсов заданий был создан инструмент, который запускает их вне Яндекс.Толоки в достаточной мере, чтобы увидеть интерфейс, без полного покрытия функциональности, которая может быть нужна заданию (например API геолокации и сервис горячих клавиш были заменены на заглушки). Далее, на основе первичных гипотез была сформулирована система меток, в соответствии с которой были размечены остальные интерфейсы, на которые пришлось, суммарно, 90% всех выполненных заданий.

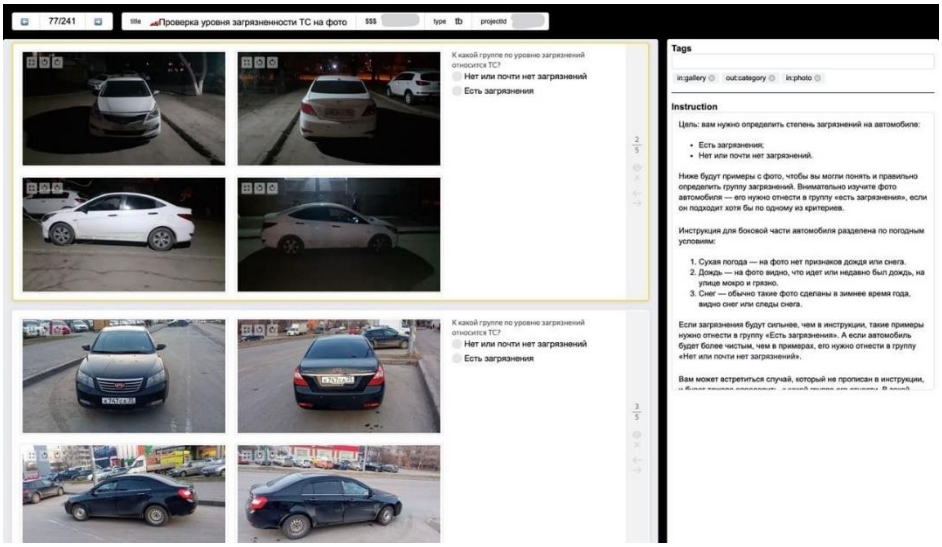


Рис. 1.5 – текущий интерфейс инструмента для просмотра заданий

В результате исследования были выделенные следующие особенности:

- В отличии от интерфейсов большинства программ в интерфейсах краудсорсинга почти всегда отсутствует навигация
- Почти в каждом задании присутствуют меры для борьбы с обманом и спешкой: проверка просмотренной видео, прослушивания аудио, перехода по ссылками из входных данных, проверка факта заполнения и формата заполнения каждого поля в интерфейсе
- Нередко встречаются сложные элементы ввода, такие как аннотация картинок, 3d выравнивание облака точек, интерактивные карты и т.д.
- Горячие клавиши встречаются в практически каждом задании, где нет обязательного ввода текста
- В отличии от форм – самого популярного вида высоко интерактивных интерфейсов, у многих заданий краудсорсинга расположений элементов отличается от вертикального списка.

Ярко выраженные классы задач были обнаружены, но не выключали в себя достаточного количества заданий, чтобы оправдать полный переход на парадигму заранее созданных интерфейсов заданий. Было принято решение вводить их в формате «пресетов» – заранее созданных интерфейсов, которые заказчик затем редактирует под свою задачу используя стандартные средства создания интерфейсов на платформе.

Также, в процессе исследования было обнаружено, что многие интерфейсы заданий, созданные заказчиками, содержат крайне непродуманные дизайн решения. Дизайн не входит в область интересов большинства заказчиков и тем более не является их целью при использовании краудсорсинг платформы. Примеры интерфейсов заданий, на которых негативно сказался дизайн, можно найти на рисунках 1.6 и 1.7.

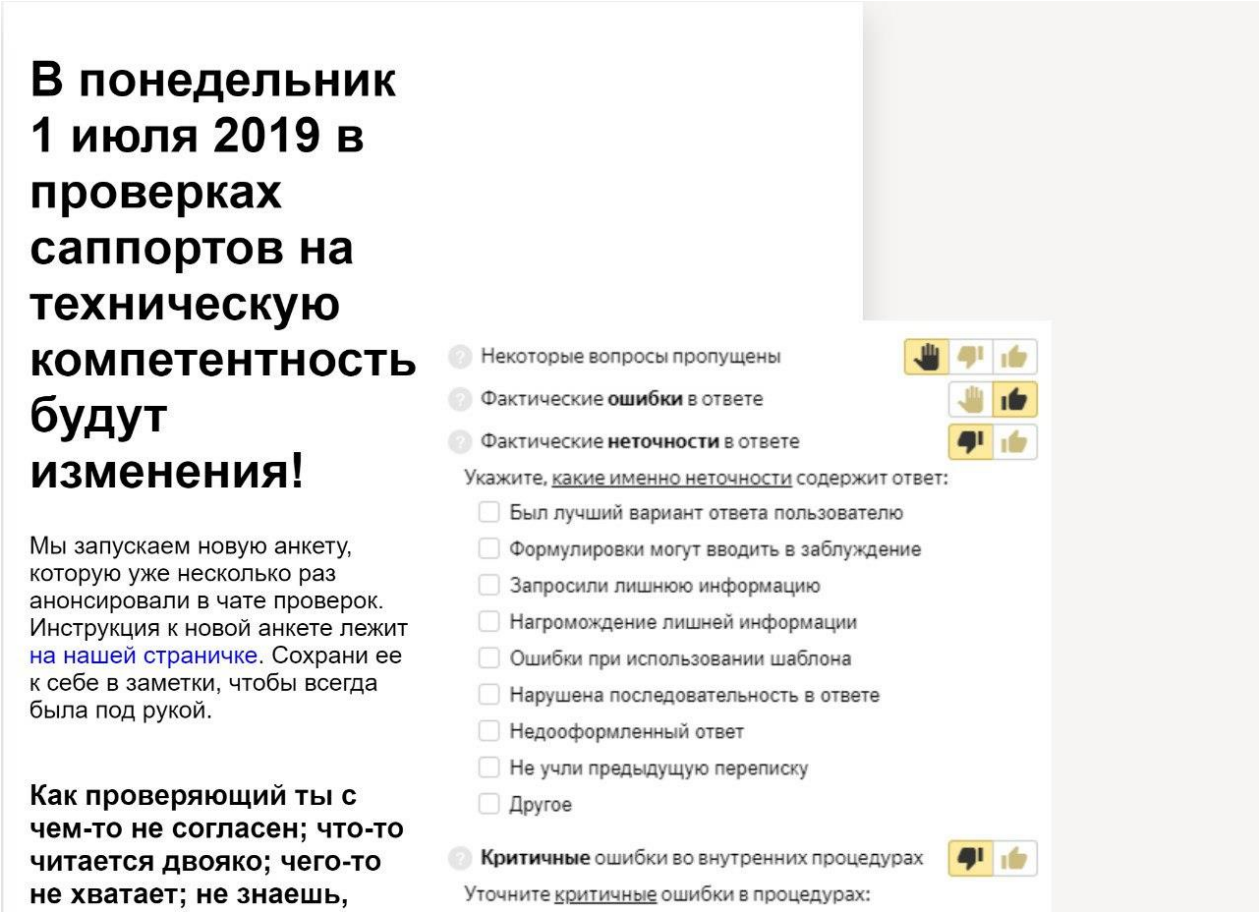


Рис. 1.6 – интерфейс задания с запутанной визуальной иерархией из-за типографики

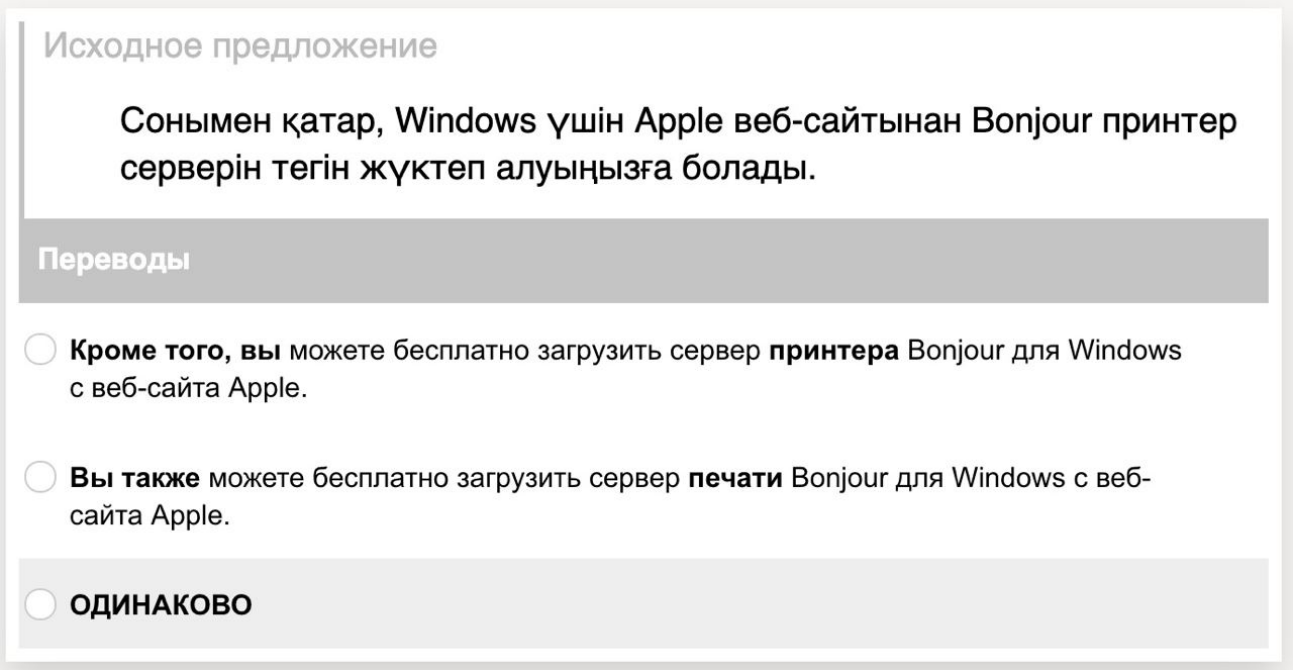


Рис. 1.7 – интерфейс задания с запутанной визуальной иерархией из-за некачественной верстки

1.3 Обзор методов и готовых решений для создания интерфейсов

Сразу стоит отметить, что на момент написания работы не удалось найти ни одно специализированного инструмента для создания интерфейсов заданий краудсорсинга. Поэтому в данном разделе будут рассматриваться инструменты создания интерфейсов общего назначения.

В результате исследования удалось сформулировать следующие требования к существующему решению:

1. Возможность создавать интерактивные веб интерфейсы (критерий **Web**)
2. Возможность встроить редактор и результат в Яндекс.Толоку. (критерий **Embed**)
3. Возможность запретить foreign code (вставки кода на другом языке, в контексте конструкторов обычно применяется по отношению к

языкам общего назначения). Это нужно чтобы избежать проблем подобных показанным рисунках 1.6 – 1.7. (критерий **No FC**)

4. Возможность использовать библиотеку компонентов и валидаций, отличную от встроенной в инструмент (критерий **Custom Lib**)
5. Возможность располагать инструменты не только вертикальным списком (критерий **Layouts**)
6. Возможность описывать базовую пользовательскую логику, вроде скрытия элементов по условию или изменения обязательности поля (критерий **Logic**)

Ниже приведена таблица соответствия готовых решений описанным выше критериям:

Решение	Web	Embed	No FC	Custom Lib	Layouts	Logic
blocks-ui.com/	✓	✓	✗	✓	✗	✗
openchakra.app/	✓	✓	✗	✗	✓	✗
laska.io/	✓	✓	✗	✓	✓	✗
craft.js	✓	✓	✓	✓	✓	✗
Яндекс.Формы	✓	✓	✓	✗	✗	✓
Внутренние решения из Яндекс.Толоки	✓	✓	✗	✓	✓	✓
Конструктор лэндингов Яндекса	✓	✗	✓	✗	✓	✗
mendix.com	✓	✗	✗	✓	✓	✓
appian.com	✓	✗	✗	✗	✓	✓

Таблица 1.1 – соответствие существующих решений критериям поставленной задачи

Кроме готовых решений были рассмотрены подходы к созданию интерактивных интерфейсов общего назначения.

Давно существует подход model-driven user interface generation, но за последние 15 лет никто так и не смог добиваться от него стабильно удовлетворительных результатов. Кроме того, внесение даже небольших правок в сгенерированные таким образом интерфейсы, как правило, является затруднительным. Вывод сделан на основе статей [35][36][37]. Неформальным примером этого подхода можно считать react-jsonschema-form [38]. Но тут все те же беды: сгенерированный интерфейс неочевиден для пользователя, а методы внесения правок неочевидны и основываются на написании ui schema, неявно связанной одновременно с подлежащей библиотекой компонентов и схемой данных.

Похожая, но значительно более популярная область исследований – grammar of graphics и все, что связано с интерактивной визуализацией данных. Однако там фокус направлен на создание визуализацией для исследовательского анализа данных. И главный сценарий для них – быстрое прототипирование умелым пользователем. Например, в случае VegaLight[39], сами авторы отмечают «our model favors conciseness over expressivity» (в нашей модели предпочтение отдано емкости, а не выразительности), что делает их подход плохо подходящим для создания интерфейсов краудсорсинга заказчиками т.к. заказчики могут не иметь специальных навыков разработки интерфейсов/интерактивных визуализаций данных, а выразительные примеры крайне полезны для достижения цели упрощения создания интерфейсов. Кроме того, VegaLight, активно полагается на foreign code для преобразования данных.

Глава 2. Проектирование и реализация

По результатам обзора было принято решение спроектировать и разработать новый инструмент для создания заданий краудсорсинга.

2.1 Конфигурация

Было принято решение разработать конструктор интерфейсов, первая версия которого будет конфигурироваться с помощью DSL на основе JSON синтаксиса. (DSL – Domain Specific Language – язык специфичный для описания узкого класса задач в одной предметной области [43], в данном случае, интерфейсов задай краудсорсинга). Использование в первой версии DSL, а не визуальной конфигурации позволяет дешевле создавать новые компоненты и экспериментировать с альтернативными способами описания. JSON был выбран как синтаксис для языка конфигурации по следующим причинам:

- По внутренним данным Яндекс.Толоки подавляющие большинство заказчиков – сотрудники IT компаний, как правило, это специалисты по Data Science, Data Engineering или Crowd Solutions Architects, т.е. они уже знакомы с синтаксисом JSON
- JSON имеет примитивный синтаксис, который можно, быстро объяснить заказчику. Он не основывается на невидимых символах как YAML, что упрощает его восприятие людьми не знакомыми с языками программирования
- Для JSON существуют библиотеки сериализации и десериализации для всех популярных языков программирования, что упрощает создание клиентов на других языках, например: для толоки существует python клиент, который, в том числе, позволяет создавать интерфейс на конструкторе используя Python классы [40]

Ниже приведен пример конфигурации интерфейса задания аннотации изображений

Конфигурация

```
{
  "view": {
    "type": "field.image-annotation",
    "label": "Выделите животных на фото",
    "image": {
      "type": "data.input",
      "path": "photo"
    },
    "data": {
      "type": "data.output",
      "path": "result"
    },
    "validation": {
      "type": "condition.required"
    }
  }
}
```

Рис. 2.1 – пример конфигурации

Рис. 2.2 – интерфейс задания, созданного на основе примера из Рис. 2.1

Рассмотрим пример из Рисунке 2.1. Задание состоит из единственного компонента *field.image-annotation* – компонента для аннотации областей на картинке. У него есть 2 свойства, значения которых должны быть строками *label* и *image*. Значение свойства *image* заменено на компонент типа *data.input*:

```
{
      "type": "data.input",
      "path": "photo"
}
```

Это указывает, что ссылку на изображение будет прочитана из входных данных задания, а именно из поля *photo*. Кроме того, в конфигурации всех *field.** компонентов есть обязательное свойство *data*, которое контролирует куда будет записан результат разметки этого компонента. В данном случае в поле *result*. Это значение обновляется в реальном времени и его можно использовать в других местах конфигурации, например, чтобы давать исполнителю подсказки по разметке или показывать дополнительные поля при определённых значениях. Кроме того, в конфигурации всех *field.** и *view.** компонентов есть необязательное свойство *validation,* позволяющее добавить проверку. В данном случае *condition.required* гарантирует, что задание можно будет отправить, только после заполнения этого поля.

Если описывать разработанный DSL обобщенно, конфигурация конструктора состоит из компонентов. Каждый компонент в ней записывается в формате

```
{
    "type": "<тип компонента>.<название компонента>"
    ...свойства компонента
}
```

А любая конфигурация имеет вид

```
{
    "view": { "type": "view.*" /* или "field.*" */   },
    "plugins": [ /*необязательный массив компонентов типа plugin.* */ ]
}
```

Компоненты бывают следующих типов:

- view – представление данных
- layout – представление, позволяющее расположить несколько view определенным способом. Технически от view ничем не отличается
- field – элемент управления для разметки
- data – указатель на входные или выходные данные задания
- helper – функция преобразование данных
- condition – условие валидации
- plugin – компонент для настройки среды, например горячих клавиш
- action – действие, которое изменяет состояние интерфейса или производит сайд эффект. Например, открывает ссылку в новой вкладке.

Также был введен набор простых правил, обеспечивающих более удобное написание конфигурации.

1. Компоненты можно вкладывать друг в друга сколь угодно глубоко. Т.е. значением свойства компонента может быть другой компонент.
2. Свойства, имеющие примитивный тип могут быть заменены на data или helper компонент. Все свойства, значением которых должен быть view компонент, также могут принимать field и layout компоненты.

3. Некоторые компоненты, например helper.if выступают как аналогично template / generic функциям в языках программирования общего назначения.

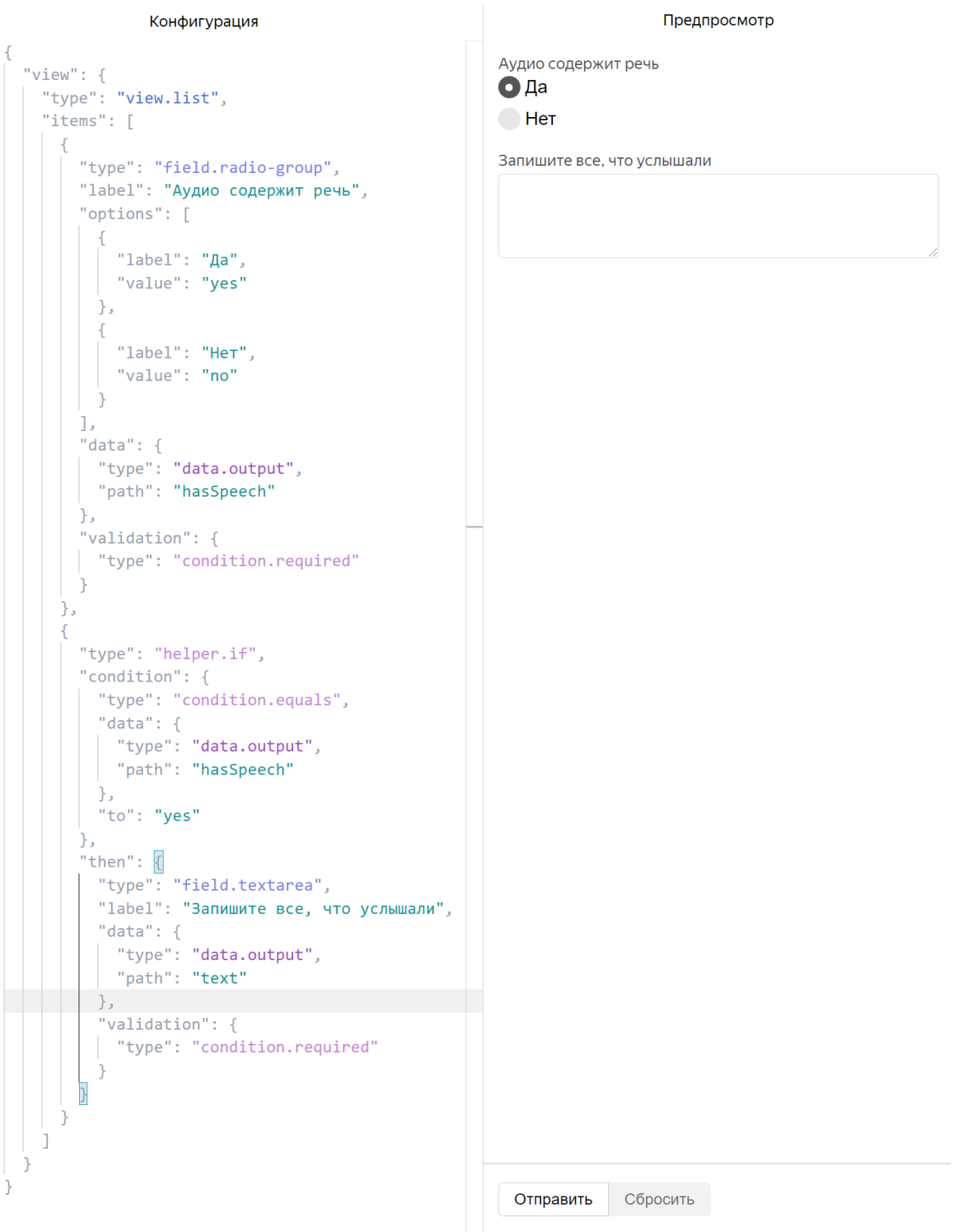


Рис. 2.3 – конфигурация и предпросмотр интерфейса задания с helper.if

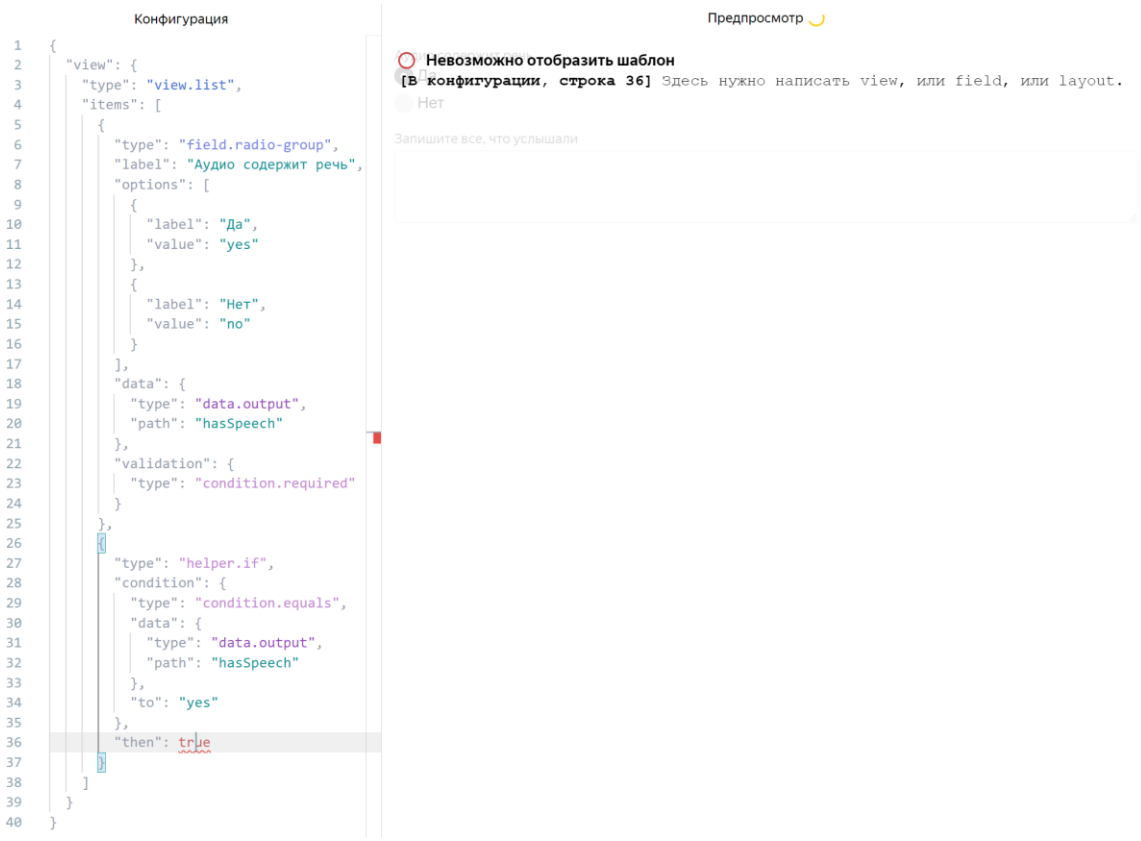


Рис. 2.4 – пример конфигурации с ошибкой типа в результате helper.if

4. У field компонентов в свойстве validation нет нужды дублировать значение свойства data, оно передает автоматически

5. Валидации, находящиеся в неактивной ветке helper.if и аналогичных компонентов не запускаются. По результатам внутренних тестов такое поведение интуитивно для заказчиков.

2.2 Программный комплекс

Для интеграции созданных на конструкторе интерфейсов заданий в Яндекс.Толоку был спроектирован и разработан специальный программный комплекс. Он был спроектирован таким образом, чтобы заказчикам не требовалось покидать Яндекс.Толоку, если они используют конструктор, а у платформы оставался контроль над всеми конфигурациями и компонентами. Сначала рассмотрим основные аспекты архитектуры в контексте интеграции с Яндекс.Толокой, которая является основным потребителем сервиса.

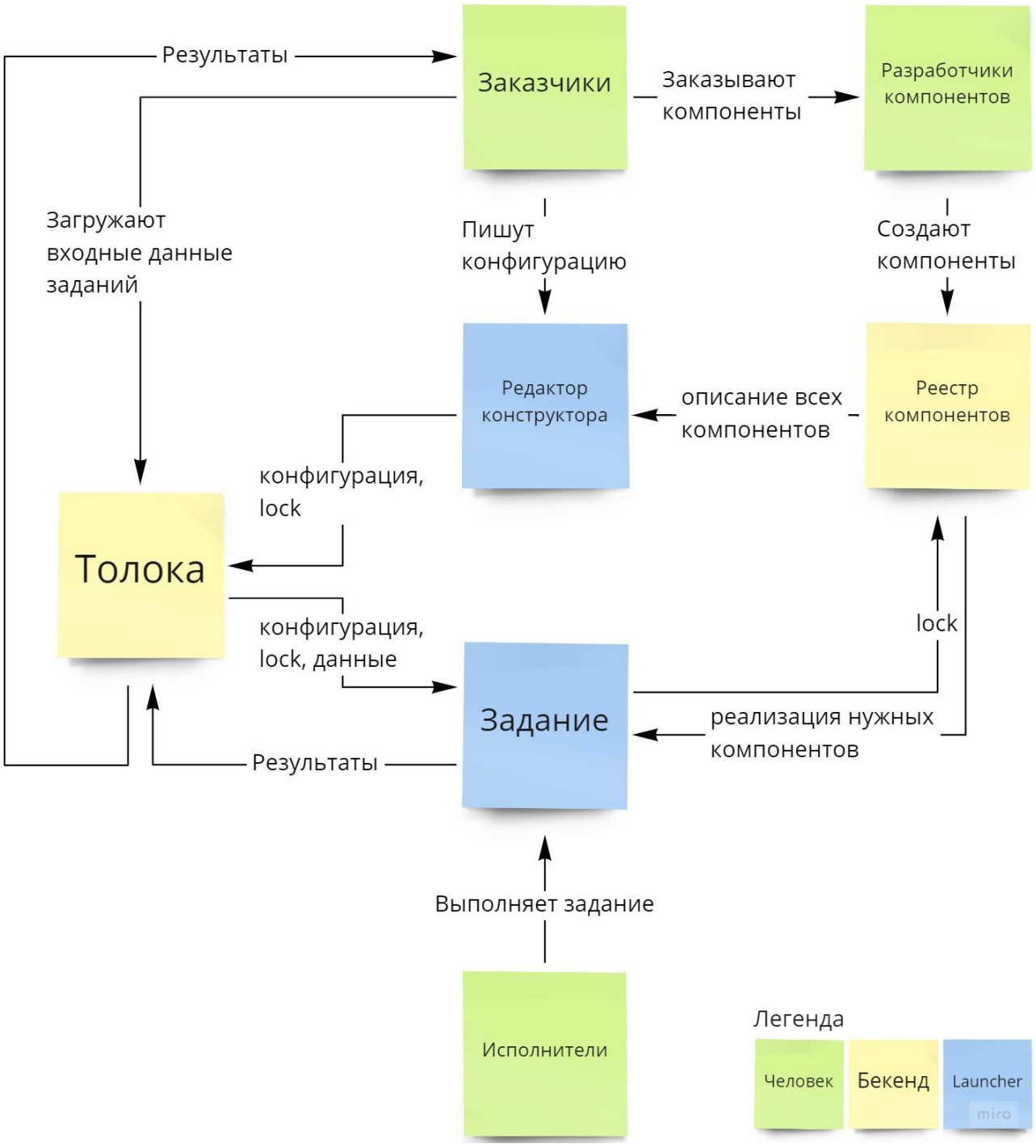


Рис. 2.5 – диаграмма архитектуры интеграции с Яндекс.Толокой

Основной сценарий работы для заказчиков в толоке выглядит так:

1. Создать проект (шаблон задания) один раз
2. Многократно загружать входные данные и получать результаты выполнения заданий

В рамках создания проекта заказчики также создают интерфейс задания. Редактор конструктора встроен в интерфейс толоки с помощью iframe.

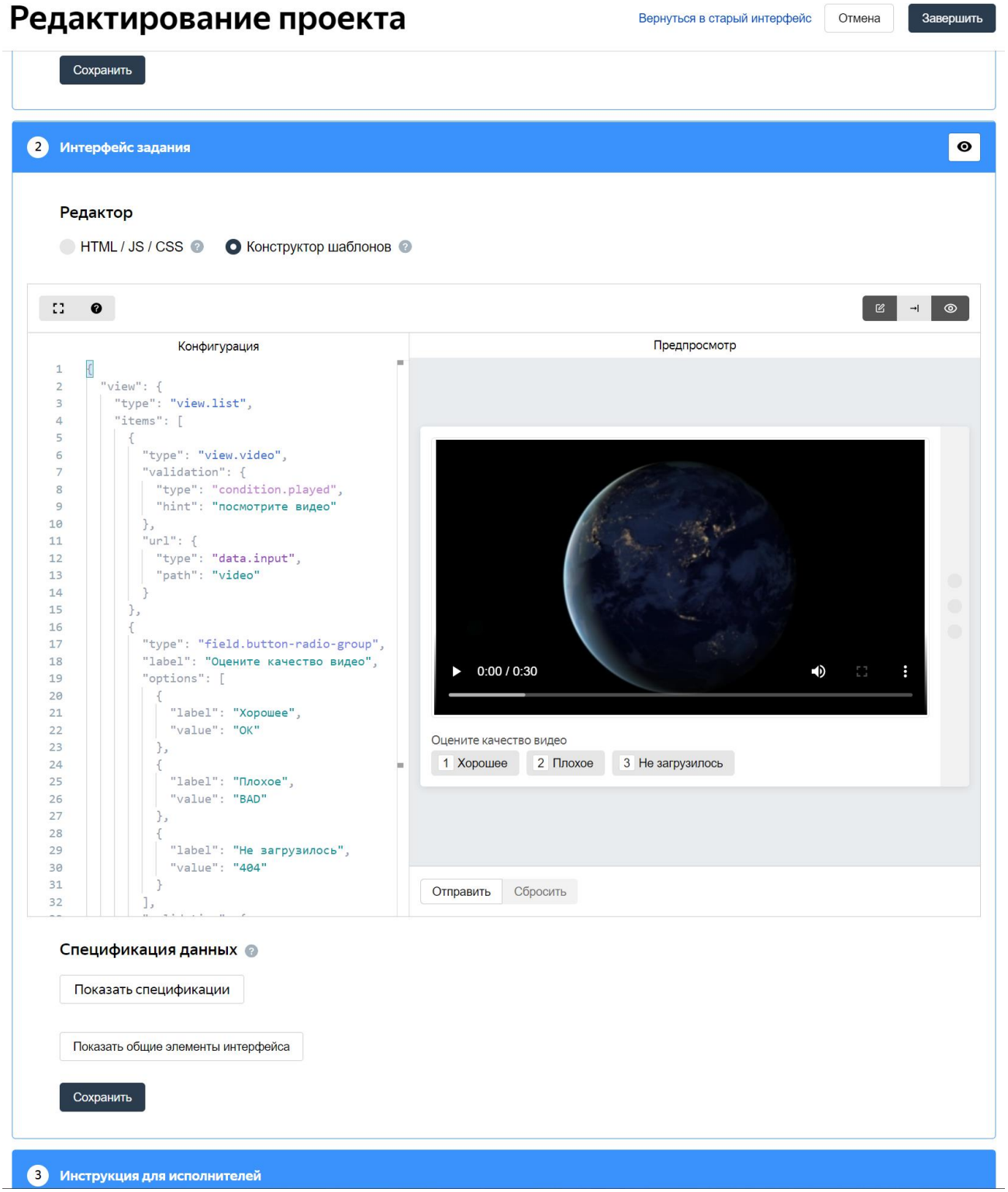


Рис. 2.6 – редактор конструктора, встроенный в интерфейс Яндекс.Толоки

Редактор реализован на аналогичном с толокой технологическом стеке: React + Typescript + Mobx и LESS + CSS-modules для стилизации. Этот выбор позволяет свободно переиспользовать стили и реализации компонентов с толокой, что помогает сохранить визуальную и UX консистентность. В качестве текстового редактора для конфигурации используется monaco-editor. Это тот же редактор, что используется в Visual Studio Code. Этот выбор был сделан т.к. Visual Studio Code является самой популярной IDE для python из имеющих расширяемый редактор [41] и самой популярной IDE для Typescript/JavaScript [42] одновременно. Таким образом комбинации клавиш и паттерны взаимодействия с редактором сразу известны и привычны пользователям, что дополнительно снижает порог входа. Также для monaco-editor был разработан модуль, добавляющий поддержку подсветки синтаксиса, автодополнения и валидации для DSL конфигурации конструктора. Метаданные всех компонентов и реализацию компонентов, участвующих в предпросмотре редактор запрашивает из реестра компонентов. Метаданные нужны, чтобы валидировать конфигурацию и совершать автодополнение. В толоку редактор сообщает конфигурацию, которую написал пользователь и lock – объект, который указывает какие версии компонентов видел заказчик при создании интерфейса задания. Запоминать эти версии необходимо, чтобы спустя годы после создания интерфейса задания и в случае выпуска несовместимых версий компонента у исполнителей он, по-прежнему, работал корректно. Lock вычисляется на основе конфигурации интерфейса задания. Также редактор позволяет создавать задания с мультиязычным интерфейсом. Для реализации форматирования дат, времени и т.д. используется библиотека React-Intl которая была выбрана т.к. реализует стандарт ICU, в то время как большинство других JavaScript библиотек для перевода на используют проприетарные форматы, что усложнит поддержку проекта в будущем. Непосредственно перевод осуществляется на стороне толоки.

После того, как заказчик создал проект и загрузил выходные данные для заданий, исполнители начинают эти задания выполнять. Интерфейс выполнения тоже можно считать частью конструктора. Скриншот интерфейса выполнения приведен на рисунке 2.7. Интерфейсы выполнения и редактор в рамках конструктора называются launcher. Все launcher ответственны за управление инициализацией интерфейсов заданий и коммуникацию с внешней средой. Например, в launcher.template (интерфейс выполнения задания) каждое задание представляется в виде одной карточки и является независимым экземпляром результата конструктора. Чтобы эти экземпляры отобразить и получать с них результаты выполнения launcher.template делает следующее:

1. Получает от толоки config и lock
2. С помощью bootstrap (общая для всех launcher библиотека для работы с реестром и ядром конструктора) запрашивает в реестре компоненты необходимые для отображения задания (на основе lock)
3. Встраивается в жизненный цикл WorkspaceSDK (библиотеки для создания интерфейсов заданий в толоке). Из него получает входные данных заданий, API горячих клавиш и геолокации, информацию о пользователе
4. После загрузки компонентов из реестра и инициализации WorkspaceSDK преобразует некоторые API Workspace SDK в стандартный формат конструктора, чтобы компоненты могли ими пользоваться.
5. Вызывает отображение заданий (результат работы конструктора – тоже React компонент)
6. При изменении выходных данных в заданиях сообщает об этом толоке через WorkspaceSDK

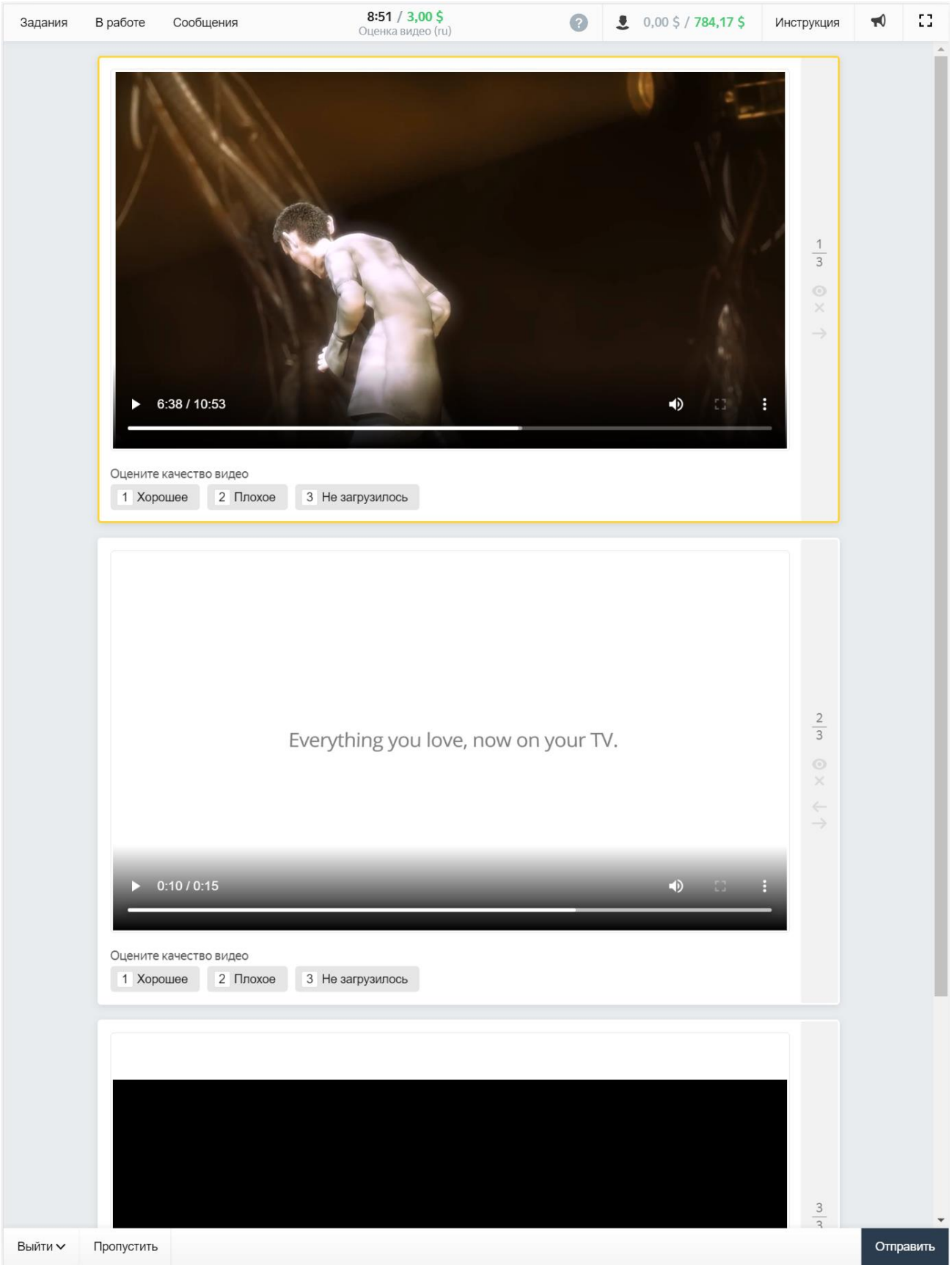


Рис. 2.7 – интерфейс выполнения задания

Рассмотрим подробнее момент «при изменении выходных данных в заданиях сообщает об этом Толоке». С помощью реактивности состояние интерфейса задания поддерживается консистентным как между компонентами, так и на уровне «интерфейс задания – launcher». Для реализации такой реактивности используется модель, когда чтение данных автоматически приводит к подписке на эти данные, а изменение данных – к оповещению всех подписчиков. Оно реализовано поверх Proxy – стандартного API Javascript которое позволяет выполнять произвольный код при обращении и изменении свойств объекта, в т.ч. еще не существующих. В конструкторе этот API используется не напрямую, а через библиотеку Mobx – она обеспечивает транзакции и оптимизации оповещения подписчиков поверх Proxy API. Эта система эффективно решает задачу поддержания консистентности ввиду особенностей изменений, которые происходят: это, практически всегда, пользовательский ввод. Т.е. за один раз изменяется очень небольшая часть данных, а сами изменения могут происходить много раз в секунду (например, при вводе текста или выделении областей на изображении)

Еще одним примечательным моментом является реестр компонентов. Его основная функция: получать lock и возвращать ссылки на скачивание реализации всех компонентов, которые нужны, чтобы отобразить интерфейс задания. Lock – объект формата

```
{
    "core": "<версия ядра в формате semver>",
    "<тип-компонента.название-компонента>": "<версия компонента в формате semver>",
    "<тип-компонента.название-компонента>": "<версия компонента в формате semver>",
    ...и так далее для каждого компонента в lock
}
```

Все компоненты, которые упоминаются в lock зависят от компонента *core*. Компоненты также могут зависеть друг от друга. В lock находятся только компоненты, непосредственно использованные в конфигурации.

Lock специально не содержит информации о зависимостях компонентов. Это позволяет реализовать следующую особенность поведения конструктора: реестр всегда возвращает новейшую версию компонента в рамках одного semver major (т.е. до версии, содержащей несовместимые изменения). Такая особенность, во-первых, значительно снижает затраты на поддержку конструктора т.к. почти никогда не приходится поддерживать несколько разных версий одного компонента. Во-вторых, это позволяет доставлять исправления ошибок и улучшения компонентов во все сервисы, использующие конструктор, без необходимости в миграции со стороны сервиса. Т.к. зависимости компонента могут меняться от версии к версии, информацию о них невозможно сохранить в lock и гарантировать актуальность этой информации. Поэтому реестр занимается сбором транзитивного замыкания зависимостей в момент запроса. Реестр реализован на java + spring и внутренних технологиях Яндекса.

С возможностью централизованной публикации компонента в несколько сервисов, несомненно, связан риск публикации компонента с ошибками. Чтобы этого избежать был предпринят ряд защитных мер:

- Исходный код всех компонентов сейчас находится в монорепозитории конструктора
- Публикация компонента в реестр на всех пользователей возможна только после вливания его в основную git ветку в монорепозитории
- Для вливания в основную ветку, код компонента должен пройти проверки
    - линтеров – ESLint + Stylelint
    - системы типов – Typescript
    - тестов – jest + внутренние технологии Яндекса

- дополнительные проверки специфичные для конструктора (DangerJS)
- Код ревью членом команды конструктора

- Изменении semver компонента вычисляется автоматически на основе коммитов, для валидации коммитов используется husky

Кроме того, весь код в монорепозитории автоматически форматируется линтерами и prettier. Сам монорепозиторий разбит на npm пакеты и управляется lerna.

Такие правила могут ощутимо замедлить создание новых компонентов, особенно на ранних стадиях. Поэтому в реестр была добавлена такая функциональность как prerelease теги. Их основная суть такова: можно опубликовать версию компонента, содержащую prerelease тег, эта версия будет использоваться только при ответе на запросы, в которых указан тот же prerelease тег. Эти версии компонентов автоматически создаются на каждый коммит в pull request. Это позволяет разработчикам быстро проверять и получать обратную связь о компонентах «в разработке». В pull request автоматически создается комментарий с ссылкой на редактор, который делает запросы с этим prerelease тегом. Но для публикации на всех пользователей уже надо выполнить ряд правил, которые помогают обеспечить стабильность инструмента.

Процесс публикации компонента заключается в том, что его исходный код минифицируется, бандлинтся и загружается на CDN. Метаданные компонента и ссылки на CDN загружаются в реестр компонентов. Это происходит автоматически. В качестве CI системы и CDN используются внутренние решения Яндекса.

Глава 3. Тестирование и апробация

Кроме требований, описанных в разделе 1.3, для оценки разработанного инструмента были введены 2 параметра:

1. Полнота покрытия области т.е. какую часть интерфейсов заданий в Яндекс.Толоке возможно создать на конструкторе
2. Порог входа – насколько тяжело заказчикам будет пользоваться конструктором

Для оценки полноты покрытия предметной области был проведен следующий эксперимент:

1. Были выбраны наиболее популярные задания, на которые на момент эксперимента приходилось 50% всего времени работы исполнителей на платформе за последний месяц.
2. Командой конструктора была проведена оценка, можно ли создать интерфейс задания используя конструктор без потери функциональности. У этой оценки было 2 градации: уже можно и «можно при условии разработки недостающего компонента»
3. Интерфейсы заданий, для которых была выставлена оценка «уже можно» были созданы на конструкторе, чтобы проверить корректность этого утверждения

На рисунке 3.1 можно ознакомиться с результатами эксперимента. В категорию «Бесполезно» были отнесены интерфейсы заданий, выражение которых на конструкторе имеет смысл только в виде единственного компонента, подходящего лишь для одного задания или одной предметной области, например, 3d выравнивание облака точек собранного лидарами.

Рис 3.1 – результаты эксперимента

Для оценки порога входа были проведены следующие мероприятия:

1. Проводились воркшопы по конструктору, в рамках которых команда кратко рассказывала про то, что такое конструктор, показывала полностью процесс создания интерфейса одного задания, затем заказчики самостоятельно создавали интерфейсы 2 других заданий
2. С помощью UX лаборатории Яндекса было проведено UX исследование, в котором участвовали как опытные заказчики, так и люди, просто выразившие интерес к платформе (далее новички). Участникам была доступна документация конструктора и редактор, но они не могли обращаться в саппорт и к команде конструктора.

На воркшопах все заказчики успешно справились с созданием интерфейсов заданий. Из-за соглашения о неразглашении детали UX исследования нельзя раскрывать. Обобщенно можно отметить, что более 70% заказчиков справились с созданием задания. Затруднения чаще происходили у новичков. По результатам UX тестирования были внесены соответствующие доработки.

Заключение

Результаты работы

В результате работы был проведен обзор особенностей интерфейсов краудсорсинга и методов их создания. Был спроектирован и разработан программный комплекс, инструмент, позволяющий заказчикам создавать интерфейсы заданий без навыков веб разработки. Разработанный инструмент был протестирован на полноту выражения интерфейсов заданий краудсорсинга, также было проведено UX тестирование, которое показало положительный опыт использования заказчиками.

Инструмент уже внедрен в сервис Яндекс.Толока. На май 2021 года более 35%-ов активных заданий в Яндекс.Толоке используют эту платформу для создания интерфейсов заданий (по внутренним данным Яндекс.Толоки). Ряд заказчиков, которые раньше просили разработчиков интерфейсов создать интерфейсы заданий, теперь создают задания полностью самостоятельно. Часть заказчиков все еще обращается к разработчикам для создания интерфейса задания, но, когда разработчики используют конструктор, создание интерфейса занимает в несколько раз меньше времени (по внутренним данным Яндекс.Толоки). Также, инструмент внедрен в несколько внутренних сервисов Яндекса и в них тоже показывает положительный опыт использования. Разработанный инструмент решает поставленную задачу: упрощает создание интерфейсов заданий краудсорсинга полностью забирая на себя ряд технических задач, необходимых для создания интерфейса задания. Кроме этого, он значительно снижает количество ошибок реализации в интерфейсах заданий краудсорсинга, в сравнении с тем, когда интерфейс каждого задания разрабатывается индивидуально. (по внутренним данным Яндекс.Толоки)

Перспективы развития

Стоит отметить, что у данного инструмента и подхода есть перспективы дальнейшего развития. Среди них стоит выделить следующие направления:

- Несмотря на явно положительный результат в сравнении с созданием интерфейсов на веб технологиях, использование DSL доставляет неудобство некоторым заказчикам. Поэтому в будущем планируется рассмотреть переход на визуальную конфигурацию, которая компилируется в текущую.
- На момент написания работы конструктор включал в себя избыточные сущности, такие как action и plugin, которые вызывали у некоторых заказчиков затруднения при использовании. Сейчас разрабатывается прототип, позволяющий скрыть их без потери выразительности и гибкости.
- На момент написания статьи конструктор предлагает использовать избыточно низкоуровневые компоненты, вроде *field.radio-group* и helper.if в качестве стандартного подхода. Дальнейшие исследования могут быть направлены на проектирование более высокоуровневых компонентов, что дополнительно упростит создание интерфейсов заданий. Также это упростит внедрение лучших практик интерфейсов краудсорсинга в интерфейсы заказчиков, которые о таких практиках не осведомлены.

Список использованной литературы

1. Hirth M., Hoßfeld T., Tran-Gia P. Anatomy of a Crowdsourcing Platform - Using the Example of Microworkers.com. // Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. 2011. С. 322–329.
2. Alonso O., Baeza-Yates, R. Design and Implementation of Relevance Assessments Using Crowdsourcing // Advances in Information Retrieval. 2011. С. 153–164.
3. Grady, C., Lease, M. Crowdsourcing Document Relevance Assessment with Mechanical Turk // Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon’s Mechanical Turk. 2010. С. 172-179
4. Kazai G., Milic-Frayling N., Costello J. Towards methods for the collective gathering and quality control of relevance assessments. // Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR ’09. 2009. С. 452–459.
5. Von Ahn L. Games with a Purpose. // Computer, 39(6). 2006. С. 92–94.
6. Callison-Burch C. Fast, cheap, and creative: evaluating translation quality using Amazon's Mechanical Turk // EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1. 2009. С. 286-295
7. Bloodgood M., Callison-Burch C. Using Mechanical Turk to build machine translation evaluation sets // CSLDAMT '10: Proceedings of the

NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk. 2010. С. 208-211.

8. Karger D. R., Oh S., Shah D. Budget-Optimal Task Allocation for Reliable Crowdsourcing Systems // Operations Research, 62(1). 2014. С. 1–24.
9. Cheap and fast --- but is it good?: evaluating non-expert annotations for natural language tasks / Snow R., O'Connor B., Jurafsky D., Y.Ng A. // EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing. 2008. С. 254-263
10. Assessing internet video quality using crowdsourcing / Figuerola Salas Ó., Adzic V., Shah A., Kalva H. // Proceedings of the 2nd ACM International Workshop on Crowdsourcing for Multimedia - CrowdMM '13. 2013. С. 23-28.
11. Komarov S., Reinecke K., Gajos K. Z. Crowdsourcing performance evaluations of user interfaces // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13. 2013. С 207-216.
12. Таран Е.А., Маланина В.А., Касати Ф. Алгоритм использования краудсорсинговых инструментов для сбора и анализа данных научных исследований (на примере подготовки систематизированного обзора литературы) // Экономика и управление инновациями. 2020. № 4. С. 39-46.
13. Cascade: Crowdsourcing Taxonomy Creation / Chilton L. B., Little G., Edge D., Weld D. S., Landay J. A. // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13. 2013. С 1999–2008.

14. Nov O., Naaman M., Ye C. What drives content tagging // Proceeding of the Twenty-Sixth Annual CHI Conference on Human Factors in Computing Systems - CHI ’08. 2008. С. 1097–1100.

15. Human Beyond the Machine: Challenges and Opportunities of Microtask Crowdsourcing / Gadiraju U., Demartini G., Kawase R., Dietze S. // IEEE Intelligent Systems, 30(4). 2015. С. 81–85.

16. The Dynamics of Micro-Task Crowdsourcing / Difallah D. E., Catasta M., Demartini G., Ipeirotis P. G., & Cudré-Mauroux P. // Proceedings of the 24th International Conference on World Wide Web - WWW ’15. 2015. С. 238-247

17. Naderi, B. Motivation of Workers on Microtask Crowdsourcing Platforms. Berlin, Technische Universität, 2017. 125с. С. 3.

18. Nakatsu R., Grossman E. Designing Effective User Interfaces for Crowdsourcing: An Exploratory Study // Lecture Notes in Computer Science. 2013. С. 221–229.

19. Horton J., Zeckhauser R., Algorithmic Wage Negotiations: Applications to Paid Crowdsourcing // CrowdConf. 2010.

20. Qui S., Gadiraju U., Bozzon A., Improving Worker Engagement Through Conversational Microtask Crowdsourcing // CHI '20: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. 2020. С. 1-12.

21. Hung N., Tam N., Tran L., Aberer K. An Evaluation of Aggregation Techniquesin Crowdsourcing // Web Information Systems Engineering – WISE. 2013. С. 1-15.

22. Community-based bayesian aggregation models for crowdsourcing. / Venanzi M., Guiver J., Kazai G., Kohli P., Shokouhi M. // Proceedings of the 23rd International Conference on World Wide Web - WWW ’14. 2014. C. 155-164.

23. Truth discovery and crowdsourcing aggregation / Gao J., Li Q., Zhao B., Fan W., Han J. // Proceedings of the VLDB Endowment, 8(12). 2015. C. 2048–2049

24. Controlling Quality and Handling Fraud in Large Scale Crowdsourcing Speech Data Collections / Rothwell S., Elshenawy A., Carter S., Braga D., Romani F., Kennewick Michael., Kennewick B. // INTERSPEECH. 2015. C. 2784-2788.

25. Understanding Malicious Behavior in Crowdsourcing Platforms / Gadiraju U., Kawase R., Dietze S., Demartini G. // Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI ’15. 2015. C. 1631-1640.

26. Kulkari A., Can M., Hatmann B. Collaboratively crowdsourcing workflows with turkomatic // CSCW '12: Proceedings of the ACM 2012 conference on Computer Supported Cooperative WorkFebruary. 2012. C. 1003–1012.

27. Deco: A System for Declarative Crowdsourcing / Park H., Pang R., Parameswaran A., Garcia-Molina H., Polyzotis N., Widom J. // 8th International Conference on Very Large Data Bases. 2012. C. 1-4.

28. Chen J., Menezes J., Bradley A. Opportunities for Crowdsourcing Research on Amazon Mechanical Turk. [Электронный ресурс] URL: https://www.researchgate.net/publication/228954187_Opportunities_for_

Crowdsourcing_Research_on_Amazon_Mechanical_Turk дата обращения (11.06.2019)

29. Davis J., Rahmanian B. User interface design for crowdsourcing systems // AVI '14: Proceedings of the 2014 International Working Conference on Advanced Visual InterfacesMay. 2014. С. 405–408.

30. Keep it simple: reward and task design in crowdsourcing / Finnerty A., Kucherbaev P., Tranquillini S., Convertino G. // Proceedings of the Biannual Conference of the Italian Chapter of SIGCHI. 2014. С. 1–4.

31. Kittur, A., Chi, E., Suh, B. Crowdsourcing User Studies With Mechanical Turk // CHI '08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2008. С. 453–456.

32. Sorokin, A., Forsyth, D. Utility data annotation with Amazon Mechanical Turk // 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. 2008. С. 1-8.

33. Chatterbox: Conversational Interfaces for Microtask Crowdsourcing / Mavridis P., Huang O., Qui S., Gadiraju U., Bozzon A. // UMAP '19: Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization. 2019. С. 243–251.

34. He X., Zhang H., Bian J., User-centered design of a web-based crowdsourcing-integrated semantic text annotation tool for building a mental health knowledge base // Journal of Biomedical Informatics. 2020. № 110. С. 1-13.

35. Faria J., Rosado da Cruz A., A Metamodel-based Approach For Automatic User Interface Generation // Models’10. 2010. С. 1-16.

36. Akiki, P. A., Bandara, A. K., Yu, Y. Adaptive Model-Driven User Interface Development Systems // ACM Computing Surveys. 2014. № 47(1), C. 1–33.

37. Aquino, N., Vanderdonckt, J., Pastor, O. Transformation templates. Proceedings of the 2010 ACM Symposium on Applied Computing - SAC ’10. 2014. C. 1195-1202.

38. react-jsonschema-form Github Readme [Электронный ресурс]. URL: https://github.com/rjsf-team/react-jsonschema-form (дата обращения 20.06.2019)

39. Vega-Lite: A Grammar of Interactive Graphics. / Heer J., Wongsuphasawat K., Moritz D., Satyanarayan A. // IEEE Transactions on Visualization and Computer Graphics 23, 1. 2017. C. 341-350.

40. Annotating ground truth for image segmentation. Toloka-kit. [Электронный ресурс]. URL: https://github.com/Toloka/toloka-kit/blob/main/examples/image_segmentation/image_segmentation.ipynb (дата обращения 16.05.2021)

41. Here are the Most Popular Python IDEs/Editors. [Электронный ресурс]. URL: https://www.kdnuggets.com/2020/10/most-popular-python-ides-editors.html (дата обращения 20.05.2021)

42. State of JS 2020. [Электронный ресурс]. URL: https://2020.stateofjs.com/en-US/other-tools/ (дата обращения 20.05.2021)

43. Fowler M., Parsons R. Domain Specific Languages / Addison-Wesley Professional. 2010. 640c.