ХАКАТОН

Binary Brains

2026

Кейсы

Старт: **16 января 2026**, 10:00 (МСК)
Дедлайн: **25 января 2026**, 23:59 (МСК)

Максимум баллов за кейс: 129 (80 базовых + 49 бонусных)

Содержание

Обзор кейсов

1 **UI Designer**

Конструктор интерфейсов + рендерер

2 **Workflow Designer**

Редактор процессов + state machine

3 **Data Model Designer**

Редактор моделей данных + binding engine

4 **Dashboard Builder**

Конструктор дашбордов + query builder

Максимум баллов: **129** за каждый кейс (80 базовых + 49 бонусных)

Подход к данным

> **Важно:** Во всех кейсах **не используются SQL-базы данных**. Вместо этого:

- **Схема** — JSON-описание структуры (сущности, поля, типы, связи)
- **Данные** — массивы JSON-объектов, соответствующие схеме

```
// Схема (metadata)
{ "entity": "Order", "fields": [
    { "name": "id", "type": "number" },
    { "name": "customer_id", "type": "reference", "target": "Customer" }
]}

// Данные (records)
[
  { "id": 1, "customer_id": 101 },
  { "id": 2, "customer_id": 102 }
]
```

Такой подход позволяет:

- Работать полностью в браузере без серверной БД
- Легко экспортировать/импортировать через JSON-файлы
- Фокусироваться на логике, а не на инфраструктуре

> Примеры JSON-схем в кейсах — иллюстративные. Ожидаем от участников собственные продуманные структуры.

Кейс 1: Low-code UI Designer

Суть задачи

Разработайте визуальный конструктор интерфейсов для Low-code платформы.

Решение состоит из двух модулей:

- **Designer** — редактор для сборки интерфейса из компонентов (drag & drop)
- **Interpreter** — движок отображения и обработки действий пользователя

> **Целевой пользователь:** бизнес-аналитик, который хочет быстро собрать форму или панель управления без написания кода.

Обязательные требования

Designer (Конструктор)

Рабочая область (Canvas)

- Зона для размещения компонентов
- Сетка для выравнивания элементов
- Выделение компонента при клике

Палитра компонентов

Компонент	Описание	Свойства
Header	Заголовок	`text`, `level` (h1/h2/h3)
Button	Кнопка	`text`, `variant`, `disabled`
Form Input	Поле ввода	`label`, `placeholder`, `required`, `type`
Checkbox	Флажок	`label`, `checked`
Table	Таблица данных	`columns`, `rows`
Group	Контейнер	`direction`, `padding`

Drag & Drop

- Перетаскивание компонентов из палитры на canvas
- Перемещение уже размещённых компонентов
- Изменение размеров компонентов

Вложенность

- Group может содержать другие компоненты, включая вложенные Group
- Минимальная глубина — 3 уровня
- Визуальная индикация при перетаскивании в Group

Панель свойств

- Редактируемые свойства выбранного компонента
- Изменения применяются сразу

Сохранение и загрузка

- Экспорт интерфейса в JSON-файл
- Импорт из JSON-файла

Interpreter (Движок отображения)

Рендеринг

- Загрузка JSON-схемы
- Динамическая сборка интерфейса
- Корректное отображение вложенности и стилей

Интерактивность

- Form Input позволяет вводить текст
- Checkbox переключается
- Table отображает данные
- Button реагирует на клик

Обработка событий

- При клике на Button — вывод alert или сбор данных формы
- Данные собираются в объект `{ "id-поля": "значение", ... }`
- Результат выводится в консоль или alert

Режим предпросмотра

- Кнопка «Предпросмотр» открывает Interpreter с текущей схемой
- Возможность вернуться к редактированию

JSON-схема

Схема должна включать: типы компонентов, ID, стили и свойства, иерархию вложенности.

```
{
  "version": "1.0",
  "name": "Форма обратной связи",
  "components": [
    {
      "id": "header-1",
      "type": "Header",
      "props": { "text": "Свяжитесь с нами", "level": "h1" },
      "styles": { "color": "#333", "marginBottom": 20 },
      "position": { "x": 0, "y": 0, "width": 400, "height": 48 }
    },
    {
      "id": "form-group",
      "type": "Group",
      "props": { "direction": "vertical", "padding": 20 },
      "children": [
        {
          "id": "name-input",
          "type": "FormInput",
          "props": { "label": "Ваше имя", "required": true, "type": "text" }
```

```
        }
      ]
    }
  ]
}
```

> **Примечание:** приведённая схема — лишь пример. Мы ждём от участников собственные улучшенные варианты структуры JSON.

Дополнительные требования (бонусы)

Требование	Описание	Баллы
Entity Binding	Интерпретатор принимает схему entity + record и заполняет форму для редактирования	до 12
Bindings	Поддержка выражений `{{ }}` в свойствах	до 10
Undo/Redo	Отмена/повтор действий, минимум 10 шагов	до 7
Копирование	Ctrl+C/V для дублирования компонентов	до 5
Валидация форм	Подсветка required-полей при отправке	до 5
Доп. компоненты	Select, Text Area, Image и др.	до 10

Entity Binding: Интерпретатор получает на вход: схему формы, схему сущности (entity) и запись (record). Поля формы автоматически заполняются данными из record. При сохранении — возвращается обновлённый record.

Критерии оценки

Критерий	Баллы	Что оценивается
Работа с компонентами	12	6 компонентов, редактирование свойств
Drag & Drop	10	Перетаскивание, перемещение, ресайз
Вложенность	8	Group принимает элементы на 3+ уровня
Рендеринг	10	Интерфейс собирается из JSON
Интерактивность	10	Компоненты реагируют на действия
Сбор данных	5	Данные формы выводятся в консоль
Полнота схемы	5	Типы, ID, стили, иерархия
Документация	5	Формат описан в README
Внешний вид	7	Современный дизайн
Удобство	5	Интуитивно понятно
Стабильность	3	Нет критических багов

Максимум: **80** базовых + **49** бонусных = **129**

Что НЕ требуется

— База данных — всё работает в браузере
— Авторизация пользователей
— Сохранение на сервер — достаточно экспорта/импорта файлов

Пример сценария

1. Открываю Designer, вижу пустой canvas и палитру компонентов
2. Перетаскиваю Header, ввожу текст «Заявка на отпуск»
3. Перетаскиваю Group, внутрь добавляю Form Input, Checkbox, Button
4. В панели свойств ставлю required для нужных полей
5. Нажимаю «Предпросмотр» — вижу готовую форму
6. Заполняю поля, нажимаю «Отправить» — в консоли JSON с данными
7. Возвращаюсь в Designer, экспортирую JSON-файл

Кейс 2: Workflow Designer

Суть задачи

Разработайте визуальный редактор для проектирования логики на основе конечных автоматов (State Machines).

Решение состоит из двух модулей:

- **Workflow Designer** — визуальный редактор диаграмм состояний
- **State Machine Core** — движок исполнения переходов

Термины

Термин	Значение	Пример
State	Состояние сущности	«Черновик», «Завершено»
Transition	Переход между состояниями	Черновик → На согласовании
Trigger	Событие перехода	`submit`, `approve`
Condition	Условие перехода	`amount > 1000`
Initial State	Начальное состояние	Всегда одно
Final State	Конечное состояние	Может быть несколько

Обязательные требования

Workflow Designer

Рабочая область

- Графический интерфейс для создания диаграмм
- Перемещение по холсту (pan)
- Масштабирование (zoom)

Узлы (States)

Тип	Вид	Описание
Start	Зелёный кружок	Начальное состояние, одно на диаграмму
State	Прямоугольник	Обычное состояние
End	Красный кружок	Конечное состояние

Связи (Transitions)

- Направленные связи между узлами
- Линии со стрелками
- Подпись триггера на линии

Панель настройки переходов

- **Trigger** — событие перехода
- **Condition** — условие перехода

Экспорт/Импорт

- Сохранение схемы в JSON
- Загрузка из JSON

State Machine Core

Входные данные

- Текущее состояние
- Входное событие
- JSON-схема процесса
- Контекст для проверки условий

Валидация перехода

Возвращает:
- `{ valid: true, targetState: "..." }` — переход возможен
- `{ valid: false, reason: "..." }` — переход невозможен

Выполнение перехода

- Смена состояния на целевое
- Запись перехода в лог

Логирование

История: timestamp, from, to, event, контекст.

Демонстрационный интерфейс

- Подсветка текущего состояния
- Кнопки для отправки триггеров
- Поле для ввода контекста
- История переходов

JSON-схема

```
{
  "id": "order-approval",
  "name": "Согласование заказа",
  "initial": "draft",
  "states": {
    "draft": {
      "name": "Черновик",
      "transitions": [{ "target": "pending", "trigger": "submit" }]
    },
    "pending": {
      "name": "На рассмотрении",
      "transitions": [
        { "target": "approved", "trigger": "approve", "condition": "amount <= 10000" },
        { "target": "rejected", "trigger": "reject" }
      ]
    },
```

```
    "approved": { "name": "Согласовано", "type": "final" },
    "rejected": { "name": "Отклонено", "type": "final" }
  }
}
```

> **Примечание:** приведённая схема — лишь пример. Мы ждём от участников собственные улучшенные варианты структуры JSON.

Дополнительные требования (бонусы)

Требование	Описание	Баллы
Параллельные состояния	Fork/Join — разделение и слияние потоков выполнения	до 12
Валидация диаграммы	Проверка: есть Start/End, все состояния достижимы	до 8
Actions	Действия при переходе: log, alert, setContext	до 7
Авто-выравнивание	Кнопка автоматического расположения узлов	до 5
Таймеры	Автоматический переход по истечении времени	до 7
Вложенные состояния	State с внутренней state machine	до 10

Критерии оценки

Критерий	Баллы	Что оценивается
Работа с узлами	10	Создание, перемещение, удаление
Работа с переходами	12	Связи, trigger, condition
Экспорт/Импорт	8	JSON сохраняется и загружается
Валидация перехода	10	Проверка возможности A → B
Выполнение перехода	8	Смена состояния, проверка условий
Логирование	4	История переходов
Демо-интерфейс	3	Тестирование процесса
Полнота схемы	5	Граф процесса
Документация	5	Формат в README
Внешний вид	7	Читаемость диаграммы
Удобство	5	Интуитивное редактирование
Стабильность	3	Нет багов

Максимум: **80** базовых + **49** бонусных = **129**

Кейс 3: Data Model Designer

Суть задачи

Разработайте визуальный редактор моделей данных и движок связывания данных с UI.

Решение состоит из двух модулей:

- **Entity Designer** — редактор сущностей и связей
- **Binding Engine** — связывание данных через выражения `{{ }}`

Термины

Термин	Значение	Пример
Entity	Сущность предметной области	Заказ, Клиент
Field	Поле с типом данных	`name: String`
Relation	Связь между сущностями	Заказ → Клиент
Constraint	Ограничение на поле	`required`, `unique`
Binding	Связывание данных с UI	`{{ order.total }}`

Обязательные требования

Entity Designer

Рабочая область

- Диаграмма сущностей (карточки с полями)
- Перемещение карточек (drag)
- Масштабирование (zoom)

Типы полей

Тип	Описание	Пример
String	Строка	`"Иван"`
Number	Число	`1500.50`
Boolean	Логическое	`true`
Date	Дата	`"2026-01-20"`
Enum	Перечисление	`["active", "inactive"]`
Reference	Ссылка на сущность	`→ Customer`

Ограничения

- `required` — обязательное поле

- `unique` — уникальное значение

Связи (Relations)

- One-to-One (1:1)
- One-to-Many (1:N)
- Many-to-Many (N:M)

Экспорт/Импорт

- Сохранение схемы сущностей в JSON-файл
- Загрузка схемы из JSON-файла

JSON-схема сущностей

```
{
  "version": "1.0",
  "entities": [
    {
      "id": "customer",
      "name": "Клиент",
      "fields": [
        { "name": "id", "type": "Number", "constraints": ["required", "unique"] },
        { "name": "name", "type": "String", "constraints": ["required"] },
        { "name": "email", "type": "String", "constraints": ["unique"] },
        { "name": "status", "type": "Enum", "values": ["active", "inactive"] }
      ]
    },
    {
      "id": "order",
      "name": "Заказ",
      "fields": [
        { "name": "id", "type": "Number", "constraints": ["required", "unique"] },
        { "name": "customer", "type": "Reference", "target": "customer" },
        { "name": "total", "type": "Number" },
        { "name": "created_at", "type": "Date" }
      ]
    }
  ],
  "relations": [
    { "from": "order", "to": "customer", "type": "many-to-one", "field": "customer" }
  ]
}
```

> **Примечание:** приведённая схема — лишь пример. Мы ждём от участников собственные улучшенные варианты структуры JSON.

Binding Engine

Парсинг выражений

```
"Привет, {{ user.name }}!" → "Привет, Иван!"
"Сумма: {{ order.total }} руб." → "Сумма: 1500 руб."
```

Поддерживаемые выражения

- Доступ к полям: `{{ user.name }}`
- Вложенные поля: `{{ order.customer.email }}`
- Операции: `{{ price * quantity }}`

- Тернарный оператор: `{{ active ? "Да" : "Нет" }}`

Реактивность

При изменении данных выражения пересчитываются автоматически.

Test Console

- Поле ввода выражения
- Редактор контекста (JSON)
- Результат в реальном времени

Дополнительные требования (бонусы)

Требование	Описание	Баллы
Наследование	Entity extends другой Entity (наследование полей и constraints)	до 12
Computed Fields	Поля с автовычислением	до 10
Расширенные constraints	`min`, `max`, `pattern`	до 7
Валидация данных	Проверка по схеме сущностей	до 8
TypeScript генерация	Экспорт типов из схемы	до 7
SQL DDL генерация	CREATE TABLE из схемы	до 5

Критерии оценки

Критерий	Баллы	Что оценивается
Работа с сущностями	10	Создание, перемещение, удаление
Работа с полями	12	6 типов, редактирование
Связи	8	1:1, 1:N, N:M
Constraints	5	required, unique
Парсинг выражений	10	`{{ }}` вычисляются
Доступ к данным	8	Вложенные поля, операции
Обработка ошибок	7	Понятные сообщения
Полнота схемы	5	Сущности, поля, связи
Импорт/экспорт	5	JSON сохраняется
Внешний вид	5	Читаемость
Удобство	3	Интуитивность
Стабильность	2	Нет багов

Максимум: **80** базовых + **49** бонусных = **129**

Кейс 4: Dashboard Builder

Суть задачи

Разработайте конструктор аналитических дашбордов и отчётов.

Решение состоит из трёх модулей:

- **Dashboard Designer** — редактор для размещения виджетов на сетке
- **Query Builder** — конструктор запросов с поддержкой JOIN
- **Expression Engine** — движок вычисляемых полей и метрик

> **Ключевая сложность:** Expression Engine должен парсить и вычислять выражения с арифметикой, агрегациями и условиями. Query Builder должен поддерживать JOIN между источниками данных.

Обязательные требования

Dashboard Designer

Рабочая область

- Grid-сетка (12 колонок)
- Drag-and-drop размещение
- Resize виджетов
- Автосдвиг (без перекрытий)

Виджеты

Виджет	Описание	Настройки
KPI Card	Числовой показатель	`value`, `title`, `format`
Bar Chart	Столбчатая диаграмма	`xField`, `yField`, `color`
Line Chart	Линейный график	`xField`, `yField`
Pie Chart	Круговая диаграмма	`valueField`, `labelField`
Data Table	Таблица	`columns`, `sortable`
Text Block	Текст	`content`, `fontSize`

Query Builder

Источники данных

- Поддержка минимум 2 источников (таблиц)
- Визуальное отображение доступных источников
- Выбор источника для каждого виджета

JOIN между источниками

Тип	Описание
Inner Join	Только совпадающие записи
Left Join	Все из левой + совпадения из правой

- Визуальный выбор полей для связи
- Отображение связи между источниками

Операции

Операция	Описание
Select	Выбор полей
Filter	Фильтрация (=, !=, >, <, contains)
Group By	Группировка
Aggregation	SUM, AVG, COUNT, MIN, MAX
Sort	Сортировка ASC/DESC
Limit	Ограничение строк

Expression Engine

Поддержка вычисляемых полей (calculated fields) с выражениями:

```
{{ sum(amount) }}                    // агрегация
{{ amount * quantity }}              // арифметика
{{ sum(amount) / count(id) }}        // комбинация
{{ if(status == "paid", amount, 0) }} // условие
```

Обязательные возможности

- Арифметические операции: `+`, `-`, `*`, `/`
- Агрегатные функции внутри выражений
- Условный оператор `if(condition, then, else)`
- Ссылки на поля из JOIN: `orders.amount`, `customers.name`

Применение

- Calculated field в Query Builder (новое поле на основе существующих)
- Значение KPI Card: `{{ sum(orders.amount) }}`
- Условное форматирование (бонус)

> **Примечание:** синтаксис выражений — лишь пример. Мы ждём от участников собственные варианты (можно использовать другой синтаксис).

Фильтры дашборда

- **Date Range** — выбор периода
- **Select** — выпадающий список

Фильтры влияют на все виджеты одновременно.

Дополнительные требования (бонусы)

Требование	Описание	Баллы
Drill-down	Клик по элементу графика → детализация (переход к подробным данным)	до 12
Cross-filtering	Клик по графику фильтрует другие виджеты	до 10
PDF экспорт	Выгрузка дашборда в PDF	до 7
CSV экспорт	Выгрузка таблицы в CSV	до 5
Тёмная тема	Переключение темы	до 5
Доп. виджеты	Area Chart, Gauge, Progress Bar	до 10

Критерии оценки

Критерий	Баллы	Что оценивается
Grid и размещение	6	Drag-and-drop, resize
Виджеты	10	6 виджетов работают
Панель свойств	4	Настройки применяются
Экспорт/Импорт	4	JSON
Источники данных	6	2+ источника, выбор для виджета
JOIN	8	Inner/Left Join, визуальная связь
Select и фильтрация	6	Выбор полей, операторы
Группировка	6	GROUP BY + агрегации
Expression Engine	10	Вычисляемые поля, арифметика, `if()`
Ссылки на поля	5	Доступ к полям из JOIN
Date Range	4	Фильтр по периоду
Select фильтр	3	Выпадающий список
Внешний вид	5	Читаемость графиков
Удобство	2	Интуитивность
Стабильность	1	Нет багов

Максимум: **80** базовых + **49** бонусных = **129**

Mock-данные

Два источника для демонстрации JOIN:

```
// orders.json
[
  { "id": 1, "date": "2024-10-05", "customer_id": 101, "product": "Смартфон", "amount": 45000,
"quantity": 1 },
  { "id": 2, "date": "2024-10-08", "customer_id": 102, "product": "Куртка", "amount": 12000, "quantity":
2 },
  { "id": 3, "date": "2024-11-01", "customer_id": 101, "product": "Наушники", "amount": 8000, "quantity":
1 },
  { "id": 4, "date": "2024-11-15", "customer_id": 103, "product": "Кофе", "amount": 2500, "quantity": 5 }
]

// customers.json
[
  { "id": 101, "name": "Иван Петров", "city": "Москва", "segment": "Premium" },
  { "id": 102, "name": "Мария Сидорова", "city": "Санкт-Петербург", "segment": "Standard" },
  { "id": 103, "name": "Алексей Козлов", "city": "Москва", "segment": "Standard" }
]
```

Пример JOIN: `orders.customer_id` → `customers.id` для получения `customers.name` и `customers.segment` в отчёте.

> **Примечание:** структура mock-данных и JSON-схема дашборда — лишь примеры. Мы ждём от участников собственные улучшенные варианты.

Общие требования

Технологии

- **Бэкенд (если есть):** только Java или Kotlin
- **Фронтенд:** любой современный фреймворк (рекомендуем React, Vue.js)
- **Хранение данных:** JSON-файлы (без базы данных)

> Для Кейсов 1, 3, 4 допустимо решение полностью на фронтенде (без бэкенда). Если бэкенд присутствует — только Java/Kotlin.

> **Важно:** Решение должно запускаться локально по инструкции из README. Проверьте инструкцию заранее!

Формат сдачи

1. Полный архив с решением (zip, tar)
2. README.md с разделами:
   - Название и описание
   - Инструкция по запуску
   - Список реализованных функций
   - Описание JSON-схемы
   - Скриншоты (желательно)

Оценка

1. Жюри запускает приложение по README
2. Тестирует функциональность по критериям
3. Выставляет баллы

FAQ

Можно ли участвовать в нескольких кейсах?
Да, но лучше один кейс качественно, чем несколько поверхностно.

Обязательно ли Java/Kotlin на бэкенде?
Если бэкенд есть — да, только Java/Kotlin. Кейсы 1, 3, 4 можно реализовать полностью на фронтенде.

Можно ли использовать готовые UI-библиотеки?
Да, разрешено и приветствуется.

Что если не успею всё?
Сдавайте то, что есть. Частичное решение лучше, чем ничего.

Нужно ли деплоить?
Не обязательно, но ссылка на демо — плюс.

Как вводить conditions в Кейсе 2?
Простые выражения: `amount > 1000`, `status == "active"`.

Какие лицензии можно использовать?
Любые пермиссивные, кроме GPL и AGPL.

Успехов! Ждём ваши решения.