УДК 004.415.53

А. С. Зайцев

РАЗРАБОТКА ОБУЧАЮЩЕГО КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ ПО ИЗУЧЕНИЮ ЯЗЫКА ПРОГРАММИРОВАНИЯ JAVASCRIPT

Статья посвящена описанию разработанного в кроссплатформенной среде WebStorm и документоориентированной базе данных MongoDB клиент-серверного приложения Learn_JS для изучения синтаксиса языка программирования JavaScript и основ алгоритмизации путем решения задач. Приведена идея автоматизации проверки выполненного практического задания обучающимся и основные фрагменты взаимодействия с приложением.

При большом интересе к онлайн-обучению очевидно, что освоить язык программирования без практики невозможно. Работа посвящена описанию разработанного клиент-серверного приложения Learn_JS для изучения синтаксиса языка программирования JavaScript [1] и основ алгоритмизации путем решения задач. Для достижения цели необходимо было решить задачи: спроектировать базу данных; разработать методику автоматизации тестирования решения; разработать тесты; разработать приложение; выложить приложение в открытый доступ.

Структурно тестовые задания состоят из описания условия задачи и набора пар входных/выходных данных, называемых далее тестом. Для создания клиент-серверного приложения Learn_JS использовалась кроссплатформенная среда разработки WebStorm. Для хранения информации о тестах применялась документоориентированная система управления базами данных, не требующая описания схемы таблиц, MongoDB, считающаяся одним из классических примеров NoSQL-систем и использующая JSON-подобные документы.

Для удобства организации проекта файловая структура со всем содержимым контентом сайта была разделена на два блока: frontend – папка с файлами клиентского приложения и backend – серверного приложения. В папке public первого блока хранятся файлы, которые не подвергаются изменению в процессе сборки проекта, а в src –исходные файлы проекта: api (взаимодействие с серверной частью); assets (изображение, аудио и т. д.); components (React компоненты приложения); constants (константы); store (файлы для работы с состоянием приложения); utils (инструменты). В папке logs второго блока хранятся логи работы сервера; а в src – исходные файлы проекта: config (конфигурационные файлы); constants (константы); controllers (контроллеры, обрабатывающие запросы); dao (файлы работы с БД); models (модели БД); routes (точки входа на сервер); services (сервисы); types (типы и интерфейсы); utils (инструменты); middlewares (промежуточные обработчики запроса).

РЕПОЗИТОРИЙ ГГУ ИМЕНИ Ф. СКОРИНЫ

Разработанное приложение Learn_JS позволяет автоматически проверять правильность выполнения задания обучающимся. Пошаговая реализация процесса проверки решения задачи приложением Learn_JS после того, как пользователь ее решит и отправит свое решение в виде функции, приведена ниже.

1) будут найдены все тесты для этой задачи из коллекции тестов по id задачи, пришедшей на тестирование;

2) на основе пришедшего на тестирование текстового представления будет создана новая функция путем транслирования на язык программирования JavaScript;

3) для каждого теста функция получит исходные параметры теста в качестве входных параметров функции, далее сравнит полученный результат выполнения с ожидаемым результатом, который взят из БД тестов;

4) если задача прошла тест, количество пройденных тестов будет увеличено на 1;

5) если задача не прошла тест, количество непройдённых тестов будет увеличено на 1, и если тест помечен как доступный для демонстрационного примера, его входные и выходные значения будут добавлены в ответ сервера, который вернется на сторону клиента и будет показан пользователю;

6) после того, как решение будет протестировано всеми тестами, вернется ответ, содержащий информацию о количестве пройденных тестов, количестве непройденных тестов, а также входных и выходных данных непройденных тестов, которые доступны для показа пользователю.

Так как в модели документа теста задачи входные и выходные данные имеют тип Mixed, то имеется возможность создания большого числа задач и тестов для них из-за отсутствия регламента на данные тестов. В качестве данных могут использоваться числа, массивы, объекты и т. д.

Для проверки усвоения материала по языку программирования JavaScript нужно сформулировать постановку задачи и подготовить тест, состоящий из входных и выходных значений. Итак, для создания задачи необходимо ввести название и описание задачи, в котором содержатся принимаемые функцией входные параметры и ожидаемый результат; создать как минимум один тест для задачи, по желанию любое количество тестов можно отметить как тесты, доступные для показа.

Пример создания задачи и интерфейс страницы клиентской части разработки показаны на рисунках 1 и 2 соответственно.

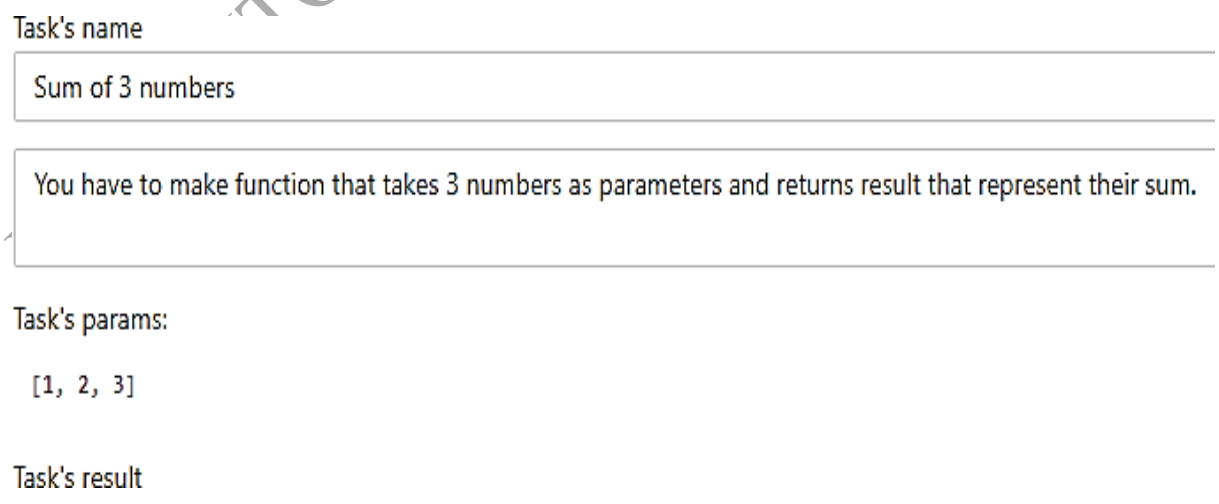

Рисунок 1 – Пример создания задачи

ТОРИЯ ГГУ ИМЕНИ Ф. СКОРИНЫ

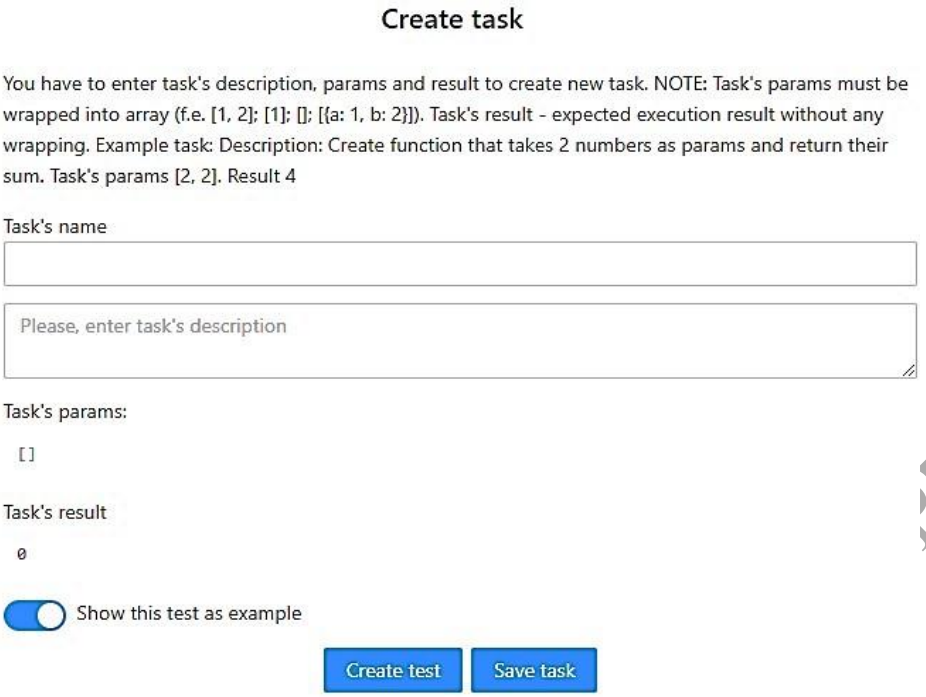

Рисунок 2 – Интерфейс страницы создания задач

Если при тестировании результата задача не проходит тест, отмеченный доступным для показа, он будет возвращен в теле ответа. После добавления тестов они будут отображаться (см. рисунок 3) вместо сообщения о том, что пока нет тестов.

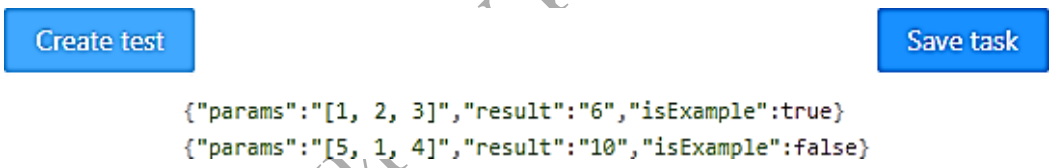

Рисунок 3 – Отображение созданных тестов

Если не выполнено хотя бы одно из обязательных условий, т. е. при неуспешном создании задачи, как и в случае успешного создания задачи, будут выданы соответствующие сообщения. Созданные новые задачи поступают в соответствующий контроллер на сервере и сохраняются. Решение пользователя тоже попадает в соответствующий контроллер на сервере для проверки, фрагмент кода которого приведен ниже:

```
export const sendSolution = async (req: Request, res: Response) => {
    const {code, id} = req.body;
    const task = await taskQueries.findById(id);
    if (!task) {
        return response(res, MESSAGES.SERVER_ERROR);
    }
    const tests = await findByTaskIdTestsForTesting(id)
    if (!tests || !tests.length) {
        return response(res, MESSAGES.SERVER_ERROR);
```

РЕПОЗИТОРИЙ ГГТУ ИМЕНИ П.О. СУХОГО

```
    }
    const func = new
    Function(code)(); let
    rightTests = 0;
    let failedExampleTests = [] as
    Array<exampleTest>; tests.forEach((test) => {
      const res = func(...JSON.parse(test.input)) ===
      parseInt(test.output, 10); if (res) {
        rightTests += 1;
      }
      else {
        failedExampleTests.push({input: test.input, output: test.output,
result: JSON.stringify(func(...JSON.parse(test.input))) });
      }
    })
    return response(res,
      CustomResponse(200, { right:
      rightTests,
      wrong: tests.length -
      rightTests, exampleTests:
      failedExampleTests
    }))}
```

В нем проверяется существование данной задачи, находятся все ее тесты. Полученное решение пользователя преобразуется в функцию для транслирования текстового отображения в функцию на языке программирования JavaScript. Далее для полученной функции, проверяемой на всех тестах, сравниваются полученные результаты с ожидаемыми. В случае совпадения результатов счетчик пройденных тестов увеличивается на 1, а непройденные тесты, доступные для примера, добавляются в ответ сервера клиенту.

В перспективе развития приложения Learn_JS запланировано разбиение теоретической части JavaScript на четыре теоретических блока: основы JavaScript, объекты, продвинутые типы данных, продвинутая работа с функциями; а также планируется выполнить совершенствование работы системы менторства, доработать вопросы администрирования приложения и отслеживания результата обучения. Приложение выложено в открытый доступ и доступно по ссылке https://javascript- learn.herokuapp.com.

Литература

1 Браун, Э. Изучаем JavaScript: руководство по созданию современных веб-сайтов / Э. Браун. –3-е. изд.– Санкт-Петербург : Диалектика, 2017. – 368 с.

РЕПОЗИТОРИЙ ГГУ ИМЕНИ Ф. СКОРИНЫ