ГРАФИЧЕСКИЙ РЕДАКТОР ДЛЯ ПОСТРОЕНИЯ UML ДИАГРАММ И ГЕНЕРАЦИИ КОДА КЛАССОВ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVASCRIPT

Н. С. Стрелков, С. С. Сыпачев
Пермский государственный национальный исследовательский университет,
614990, Пермь, Букирева, 15

В разработке программного обеспечения огромную важность имеет процесс проектирования как системы в целом, так и отдельных ее частей. При проектировании необходимо учитывать парадигму, с использованием которой ведется разработка. Доминирующая парадигма сейчас – это ООП (объектно-ориентированное программирование). Для проектирования в этой парадигме самым популярным инструментом является язык UML (Unified Modeling Language), позволяющий строить диаграммы классов. Существуют также программные продукты (кодогенераторы), которые на основе этих диаграмм генерируют код классов на объектно-ориентированном языке:

- MS Visual Studio – C#
- Eclipse – Java
- Rational Rose – C++, Java, C#

Язык JavaScript – сценарный прототипно-ориентированный язык программирования применяющийся преимущественно в веб-разработке. До 2009 года был языком, выполняющимся только на стороне клиента в браузере. В 2009 году была разработана платформа Node.js, которая позволила исполнять JavaScript на стороне сервера. Следовательно, появилась возможность писать полноценные системы на этом языке. Однако программ для генерации кода классов на языке JavaScript еще нет.

Целью работы было проектирование и разработка веб-приложения, состоящего из двух подсистем:

- Подсистема построения диаграмм – графический редактор для построения UML диаграмм классов (клиентская часть),
- Подсистема генерации кода - кодогенератор классов на языке JavaScript (серверная часть).

В данной статье описана разработка подсистемы генерации кода. Для достижения цели необходимо было выполнить следующие подзадачи:

- Спроектировать протокол клиент-серверного взаимодействия;
- Спроектировать структуру и механизмы взаимодействия пользовательских классов;
- Подобрать свободно распространяемые программные продукты (БД, библиотеки, фрэймворки, платформы);
- Запрограммировать серверную часть.

Клиент-серверное взаимодействие реализовано по архитектуре REST. REST – (сокр. от англ. Representational State Transfer – «передача репрезентативного состояния») – метод взаимодействия компонентов распределённого приложения в сети Интернет, при котором вызов удаленной процедуры представляет собой обычный HTTP-запрос (в данной работе использовались только GET или POST), а необходимые данные передаются в качестве параметров запроса.

REST запросы имеют следующий вид:

GET http://<доменное имя сервера>/<путь к интересующему ресурсу>?<параметры запроса>

Например: запрос на получение всех проектов пользователя с уникальным идентификатором “example-u1” имеет вид:

GET http://localhost:3000/api/project/ example-u1

Формат данных для обмена между клиентом и сервером - JSON

На рис.1 представлена схема пользовательских классов, хранящих данные о диаграммах, созданных пользователями:

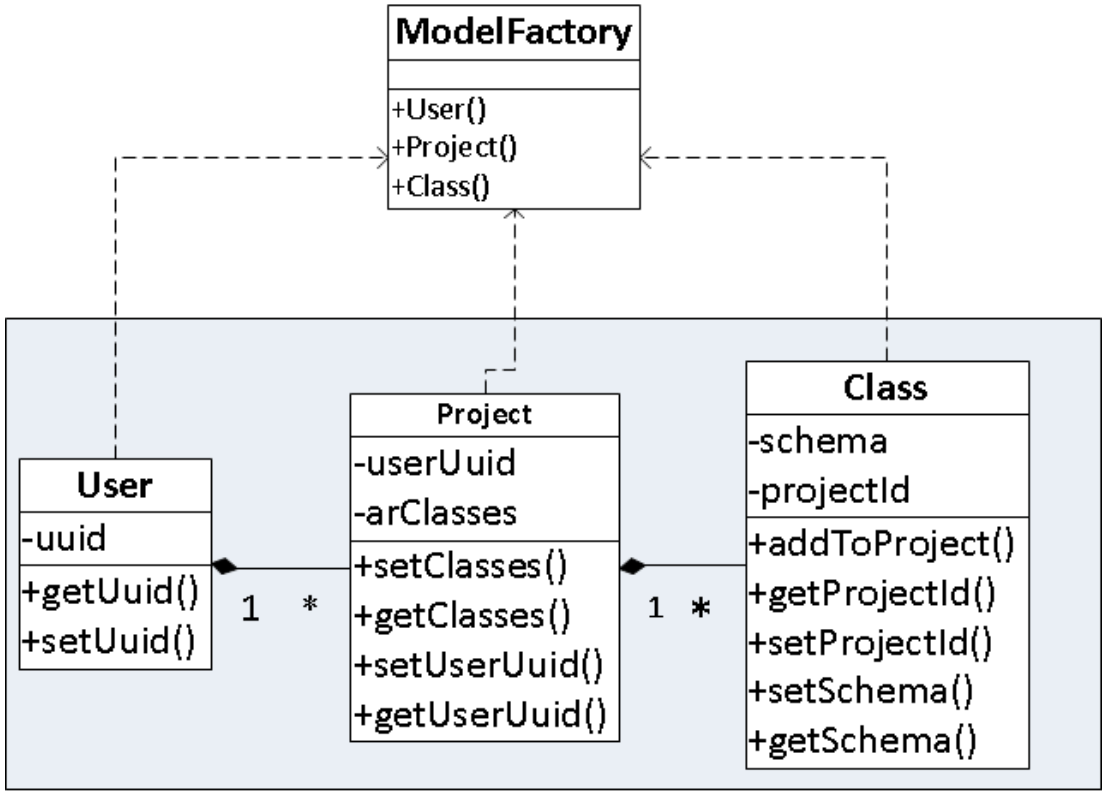

Рис. 1. *Структура пользовательских классов подсистемы, отвечающей за хранение данных.*

Пользователь (User) регистрируется в системе и может создать неограниченное количество проектов (Project). Проект подразумевает диаграмму классов, следовательно, проект может состоять из неограниченного числа классов. Объект Class – это набор связанных классов, данные о которых хранятся в свойстве Schema.

Для удобства работы с этими классами был использован вспомогательный класс ModelFactory, реализующий структурный шаблон проектирования «Фасад».

На рис. 2 представлена схема классов, которые отвечают за генерацию кода классов, из диаграмм, построенных пользователем.

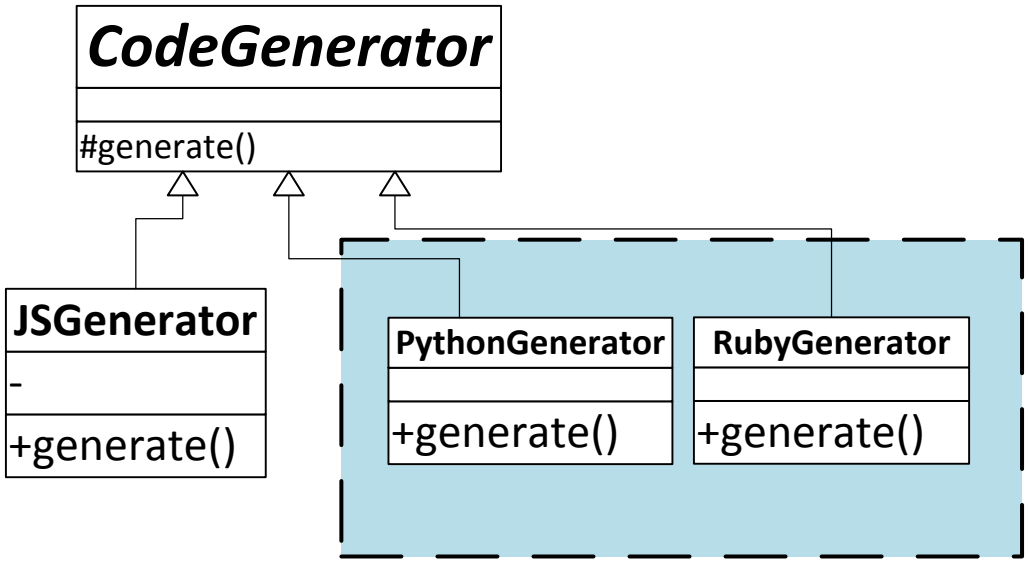

Рис. 2. *Схема классов подсистемы генерации кода*

Класс JSGenerator наследует от абстрактного класса CodeGenerator и переопределяет полиморфный метод generate(), который получает данные из свойства Schema объекта класса Class и генерирует на их основе JavaScript код. В блоке, выделенном пунктиром, представлены классы для генерации кода на языке Python и Ruby. Они еще не реализованы, но на данной схеме видно, что такая структура позволяет легко масштабировать приложение.

Для реализации приложения был выбран следующий стек технологий:

- Язык программирования – JavaScript
- Статический и проксирующий сервер Nginx
- Серверная платформа – Node.js
- Фрэймворк, имеющий набор уже реализованных объектов для работы с протоколом http – Express
- NoSQL база данных, для удобной работы с форматом JSON – MongoDB
- Библиотека для работы с базой данных – Mongoose

На рис. 3 представлена общая схема взаимодействия элементов данного стека в приложении:

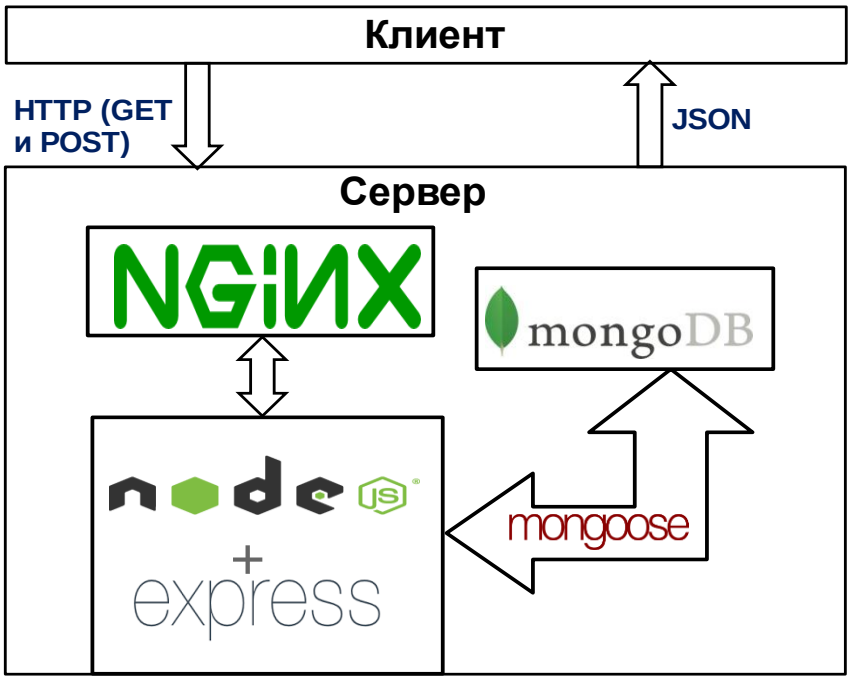

Рис. 3. *Схема взаимодействия элементов стека, на котором реализована серверная часть приложения*

Алгоритм работы серверной части приложения:

- Клиент отправляет HTTP запрос на сервер
- Запрос обрабатывается статическим сервером Nginx. Если пришел запрос на получение статической веб-страницы, то Nginx отправляет эту страницу клиенту в формате html. В противном случае перенаправляет запрос на динамический веб-сервер Node.js
- Node.js обрабатывает полученный запрос и делает запрос к базе данных.
- Если пришел запрос на получение данных о пользователе, проекте или классе, то node.js возвращает данные из базы клиенту в формате JSON
- Если пришел запрос на получение кода классов, то данные из базы обрабатываются классом JSGenerator и полученный код передается клиенту.

Список литературы

1. *Кантелон М., Хартер М., Головайчук Т., Райлих Н.* Node.js в действии. СПб.: Питер, 2014
2. Документация по Node.js для разработчиков [URL] – https://nodejs.org/documentation/
3. Документация по MongoDB для разработчиков [URL] – http://docs.mongodb.org/manual/