МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

|  |  |
| --- | --- |
|  | |
|  | **Институт  интеллектуальных кибернетических систем** |
| **Кафедра №22 «Кибернетика»** |
| Направление подготовки 09.03.04 Программная инженерия | |
| **Отчет** | |
| об учебной практике на тему: | |

Анализ возможностей системы, построенной вокруг DSL-языка для описания эндпойнтов на стороне сервера

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Группа | | Б20-514 | | | |  | | | |
| Студент | |  | | | |  | Шенягин Даниил Михайлович | | |
|  | | (подпись) | | | | (ФИО) | | | |
| Руководитель | |  | | | |  | Петровская Анастасия Викторовна | | |
|  | | (подпись) | | | | (ФИО) | | | |
| Научный консультант | |  | | | |  |  | | |
|  | | (подпись) | | | | (ФИО) | | | |
| Оценка руководителя |  | | Оценка консультанта | | | | |  | |
|  | (0-100 баллов) | |  | | | | | (0-100 баллов) | |
|  |  | |  | | | | |  | |
| Итоговая оценка |  | | ECTS | | | | |  | |
|  | (0-100 баллов) | |  | | | | |  | |
| Комиссия | | | | | | | | | |
| Председатель |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |
|  |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |
|  |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |
|  |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |

**Москва 2023**

**ЗАДАНИЕ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п/п | Содержание работы | Форма  отчетности | Срок исполне­ния | Отметка о выполнении  Дата, подпись рук. |
|  | Изучение и сравнительный анализ подходов для описания эндпоинтов | Разделы ПЗ | 26.07.2023 |  |
|  | Изучение и анализ выполнения программного кода и постановка задачи данной темы | Разделы ПЗ | 29.07.2023 |  |
|  | Описание собственного DSL-языка и проработка его основных частей с примерами | Архитектура | 03.08.2023 |  |
|  | Архитектура системы с описанием взаимодействия модулей данной семтемы | Архитектура, рабочие материалы | 05.08.2023 |  |
|  | Архитектура модуля для интерпретации DSL-языка на уровне типов | Архитектура, рабочие материалы | 07.08.2023 |  |
|  | Архитектура модуля для интерпретации DSL-языка непосредственно на Typescript | Архитектура, рабочие материалы | 08.08.2023 |  |
|  | Описание алгоритма для ограничений сигнатур с их имплементациями | Алгоритм | 11.08.2023 |  |
|  | Описание модуля для интеграции системы с фреймворком Express | Архитектура, рабочие материалы | 12.08.2023 |  |
|  | Описание тестирования системы | Тестирование | 13.08.2023 |  |
|  | Подготовка к опубликованию системы как npm-пакета в npm-registry | Публикация библиотеки | 14.08.2023 |  |
|  | *Оформление пояснительной записки (ПЗ) и иллюстративного материала для доклада*. | Текст ПЗ, презентация | 16.08.2023 |  |

**Здесь будет содержание**

**Введение**

В современном программном разработке строгая типизация играет важную роль при создании эффективных и надежных систем. Одним из ключевых аспектов, требующих строгой типизации, является реализация эндпоинтов на стороне сервера. Эндпоинты представляют собой точки входа, через которые клиентские приложения могут обращаться к серверу или выполнять определенные функции. Моделирование и реализация точно типизированных эндпоинтов может быть сложной задачей, особенно при работе со сложными системами.

В данном реферате будет проведен анализ возможностей системы, основанной на DSL-языке (Domain Specific Language), для описания точно типизированных эндпоинтов на стороне сервера. DSL-язык предлагает специализированные синтаксические конструкции и абстракции для определенной предметной области, что упрощает и улучшает процесс создания точно типизированных эндпоинтов.

Цель данного исследования заключается в анализе возможностей системы, базирующейся на DSL-языке, в контексте объявления и реализации точно типизированных эндпоинтов на стороне сервера. Будут рассмотрены различные аспекты внедрения и использования DSL-языка для создания и конфигурирования эндпоинтов. Также будет изучена автоматизация и генерация кода для облегчения процесса разработки точно типизированных эндпоинтов на стороне сервера.

Для проведения исследования будет использована доступная литература, статьи, руководства и опыт практического применения DSL-языков. Будет осуществлен обзор существующих решений и проведена критическая оценка их преимуществ и ограничений. Также будут представлены примеры реального использования DSL-языка для объявления и реализации точно типизированных эндпоинтов на стороне сервера, а также приведены результаты проведенного анализа.

Целью данного исследования является предоставление полного обзора применения DSL-языка для создания точно типизированных эндпоинтов на стороне сервера и анализ его возможностей и ограничений. Результаты исследования помогут разработчикам и архитекторам программных систем принять обоснованные решения при выборе подходящего инструмента и управлении процессом создания точно типизированных серверных API.

# **1. Изучение и сравнительный анализ подходов для описания эндпоинтов**

## 1.1. Обзор основных подходов для описания эндпоинтов

- Фреймворки на основе аннотаций: Исследование фреймворков, которые позволяют описывать эндпоинты с использованием аннотаций непосредственно в коде. Рассмотрение преимуществ и ограничений этого подхода.

- Конфигурационные файлы: Изучение использования конфигурационных файлов для описания эндпоинтов, в том числе форматов файлов и возможных инструментов для работы с ними.

- DSL-языки: Анализ применения специальных DSL-языков для описания эндпоинтов на стороне сервера. Осмысление преимуществ и возможностей, которые предлагает этот подход.

## 1.2. Сравнительный анализ подходов

- Производительность: Сравнение производительности различных подходов при описании и обработке эндпоинтов. Исследование скорости выполнения запросов, объема используемой памяти и других показателей производительности.

- Гибкость и расширяемость: Оценка гибкости и расширяемости каждого подхода для адаптации к изменяющимся требованиям и добавлению нового функционала. Рассмотрение возможностей расширения и модификации системы.

- Уровень сложности: Анализ уровня сложности реализации и поддержки каждого подхода. Исследование необходимого количества кода, структуры проекта, используемых инструментов и технических требований.

- Экосистема и сообщество: Изучение наличия и развития экосистемы и сообщества вокруг каждого подхода. Анализ качества документации, наличия руководств, помощи и поддержки от сообщества.

## 1.3. Примеры применения подходов

- Рассмотрение реальных примеров использования фреймворков на основе аннотаций для описания эндпоинтов. Анализ сценариев, требований и результатов применения.

- Изучение примеров использования конфигурационных файлов для описания эндпоинтов. Анализ структуры и формата конфигурационных файлов, а также примеров успешного применения.

- Анализ и сравнение DSL-языков, используемых при описании эндпоинтов на стороне сервера. Исследование возможностей, гибкости и преимуществ каждого DSL-языка.

## 1.4. Оценка и выводы

- Сравнение и суммирование результатов изучения и сравнительного анализа подходов для описания эндпоинтов.

- Формулирование выводов о преимуществах и ограничениях каждого подхода.

- Рекомендации по применению того или иного подхода в зависимости от конкретных требований и особенностей проекта.

# **2. Изучение и анализ выполнения программного кода и позиционирование системы в этой схеме**

# 2.1. Выполнение программного кода

Код проходит стадии от самого его написания до работы на некоторой машине

0. Написание кода

1. Статический анализ IDE

2. Работа дополнений IDE (в нашем случаем Typescript Language Server)

3. Компиляция/интерпретация – compile-time

4. Run-time

(ТУДУ: нарисовать схему этапов и спозиционировать систему на этой схеме)

# 3. **Архитектура системы**

## 3.1. Описание собственного DSL-языка и проработка его основных частей

Один из ключевых аспектов изучаемой системы - разработка собственного DSL-языка для описания эндпоинтов на стороне сервера. Создание такого языка позволит разработчикам более эффективно и гибко описывать требуемые эндпоинты, улучшая читаемость, поддерживаемость и повторное использование кода.

Процесс описания собственного DSL-языка и проработка его основных частей включает следующие этапы:

1. Определение целей и требований
2. Проектирование синтаксиса и грамматики
3. Реализация парсера и лексера
4. Разработка семантики и интерпретации
5. Тестирование и отладка

В данном разделе будет описаны этапы «определение целей и требований» и «проектирование синтаксиса и грамматики».

### 3.1.1. Определение целей и требований для описания собственного DSL-языка

Для начала определяются цели и требования для создания DSL-языка. Формулируются функциональные и нефункциональные требования, а также учитываются особенности эндпоинтов на стороне сервера. Например, целью может быть создание простого в использовании и гибкого языка, который позволяет легко определить различные типы эндпоинтов и конфигурации.

#### 3.1.1.1. Функциональные требования

1. Возможность явного объявления эндпоинтов: язык должен предоставлять синтаксическую конструкцию для явного объявления эндпоинтов на стороне сервера. Разработчику должно быть позволено указать необходимые параметры, типы запросов и ответов, а также прочие дополнительные настройки.
2. Поддержка различных типов запросов: язык должен предоставлять возможность объявления эндпоинтов с различными типами запросов, такими как GET, POST, PUT, DELETE и другими. Разработчик должен иметь возможность указать соответствующие методы обработки запросов.
3. Возможность определения параметров: язык должен поддерживать объявление и использование параметров для эндпоинтов. Разработчик должен иметь возможность задавать параметры, их типы и значения, которые будут передаваться в запросах.
4. Поддержка валидации данных: язык должен обеспечивать возможность задания правил валидации данных для эндпоинтов. Разработчик должен иметь возможность проверять входные данные и обрабатывать ошибки валидации с помощью назначенных правил.
5. Механизм маршрутизации: язык должен предоставлять механизм маршрутизации, который позволяет разработчику определить, какой эндпоинт будет вызываться при обращении по определенному URL-адресу. Разработчику должно быть предоставлено исчерпывающее управление маршрутизацией.

#### 3.1.1.2. Нефункциональные требования

1. Читаемость и выразительность: язык должен быть читаемым и понятным для разработчиков. У него должна быть ясная синтаксическая структура и интуитивно понятные конструкции, позволяющие описывать эндпоинты наиболее естественным и лаконичным образом.
2. Гибкость и расширяемость: язык должен быть гибким и расширяемым, позволяя разработчику легко добавлять новые функциональные возможности и поддерживать различные стили и конвенции разработки. Он также должен предоставлять механизмы для переиспользования кода и композиции эндпоинтов.
3. Поддержка статической типизации: язык должен поддерживать статическую типизацию, позволяя разработчику проверять типы данных на этапе разработки. Это помогает обеспечить более надежный и безопасный код, а также упрощает сопровождение и рефакторинг.
4. Производительность: язык должен обеспечивать высокую производительность во время выполнения эндпоинтов. Чрезмерная сложность грамматики и интерпретации может привести к снижению производительности, поэтому разработчик должен уделить внимание оптимизации кода и алгоритмов обработки эндпоинтов.
5. Поддержка документации и инструментов разработки: язык должен быть хорошо документирован.

### 3.1.2. Проектирование синтаксиса и грамматики собственного DSL-языка

На данном этапе проектируется синтаксис и грамматика DSL-языка. Разрабатываются ключевые конструкции языка, такие как декларация эндпоинта, параметры, типы данных и их комбинации. Учитываются потенциальные операции и возможности, которые могут встретиться в эндпоинтах.

Существует как минимум два способа описания эндпоинтов на DSL-языке:

1. Описание обособленных эндпоинтов, которые сохраняют свою структуру после изменений, связанных с ними эндпоинтов
2. Описание полного *дерева эндпоинтов –* дерево зависимостей эндпоинтов с наследованием параметров потомками от предков.

В данной работе при выборе способа описания использовались критерии функциональных и нефункциональных требований, критичными критерием были «наследуемость» и «гибкость и расширяемость»

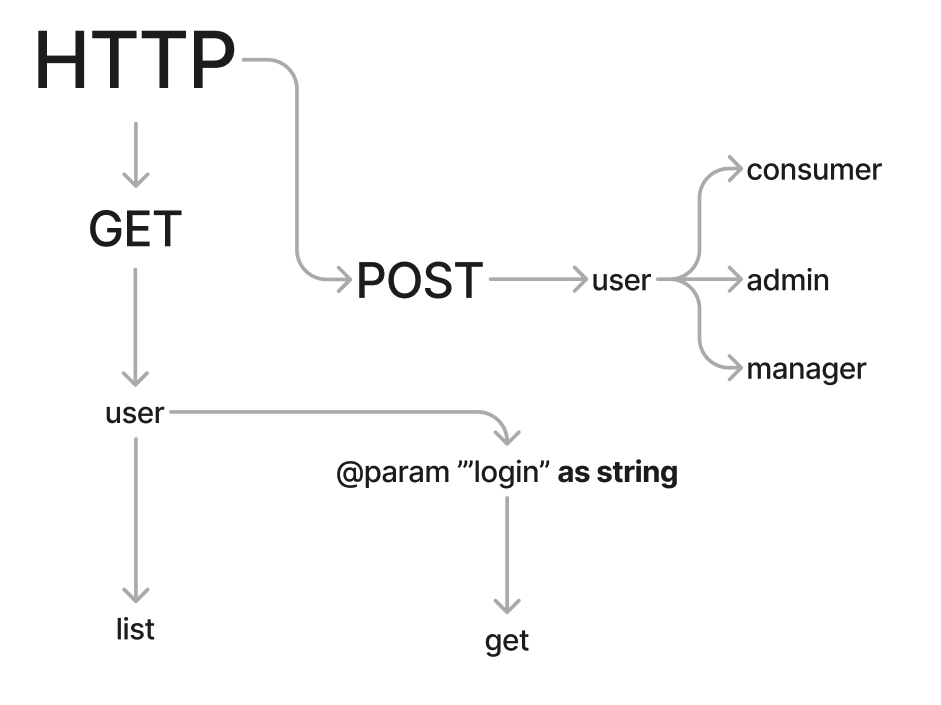


Рис. 1 Пример дерева зависимостей эндпоинтов без параметров

На рисунке 1 изображена схема зависимостей эндпоинтов в виде дерева. Данная структура позволяем гибко и расширяемо задавать эндпоинты, наследуя некоторые признаки и параметры родителей.

**|> HTTP/2**

**<|> POST**

**<|> user**

**<|> consumer**

**<|> admin**

**<|> manager**

**<|> GET**

**:> user**

**<|> list**

**<|> @param 'login' as String**

**:> get**

Пример кода 1. Синтаксис DSL-языка без вставки параметров

Как видно из примера кода 1 с помощью специальных знаков (знак разветвления) и (знак прямого наследования) можно задавать полноценное дерево эндпоинтов любого уровня вложенности. Для задания динамической части названия эндпоинта можно воспользоваться директивой *@param,* которая определят некоторый параметр типа *string*, как в примере, или для необходимости *number.* Данный пример задает следующие названия эндпоинтов:

* *POST /user/consumer*
* *POST /user/admin*
* *POST /user/manager*
* *GET /user/list*
* *GET /user/****Daniil****/get –* с параметром *{“login”: “Danill”}*

#### 3.1.2.1. Наследование эндпоинтов и дополнение дерева зависимостей параметрами

Для того, чтобы описать нескорый энпдоинт, необходимо задать следующие параметры:

1. Метод (POST, GET, PUT, DELETE, и тд)
2. Название эндпоинта (/user/list)
3. Схема input (то есть схема того, что мы должны прислать на вход) – *опционально*
4. Схема output (то есть схема того, что мы должны получить на выходе) – *опционально*
5. Заголовки (Connection, Host, user-agent, token, Authorization) – *опционально*

Так как в дальнейшем для данной работы будет разразрабатываться модуль для авто-кодогенерации сваггер-спеки, нужно передавать метаинформацию, которая также будет наследоваться, поэтому для вышенаписанного списка необходимо добавить пункт «метаинформация».

**|> HTTP/2**

**<|> POST**

**<|> user**

**@meta { swaggerTag: "User" }**

**@headers { token: String }**

**@input json { login?: String, email: String }**

**@output json { id: String }**

**<|> consumer**

**@input json #include(login) & { location: String }**

**<|> admin**

**@input json #include(login) & { rules: String }**

**<|> manager**

**@input json #exclude(email) & { managerRang: Number }**

**<|> GET**

**:> user**

**<|> list**

**@input json merge(#ListPaginationInput, #ListSortInput)**

**<|> @param 'login' as String**

**:> get**

Пример кода 2. Синтаксис DSL-языка с параметрами

В качестве входных параметров задается директива *@input,* для выходных *@output*, для заголовков *@headers*, для описания метаинформации – *@meta*, как это показано из примера кода 2.

Наследование осуществляется неявно: каждый последующий потомок наследует все параметры, заданные директивами. Также, чтобы выбирать нужные параметры из полного списка наследуемых параметров, необходимо ввести утилитарную директиву *#include* или для исключения определенных параметров – *#exclude.*

## 3.2. Общая архитектура системы

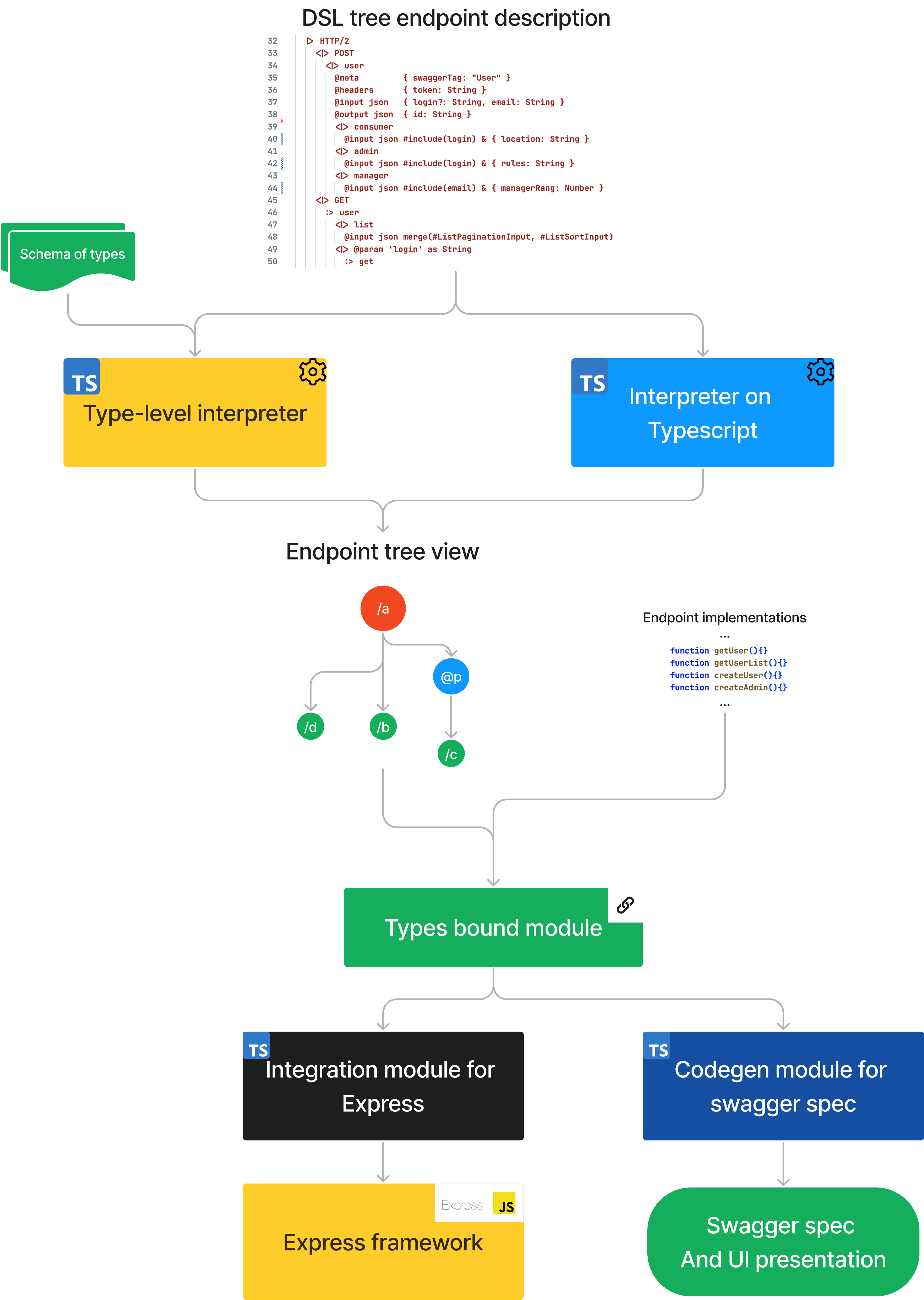


Рис. 2 Общая архитектура системы

## 3.3. Архитектура модуля для интерпретации DSL-языка на уровне типов

## 3.4. Архитектура модуля для интерпретации DSL-языка непосредственно на Typescript

## 3.5. Алгоритм для ограничений сигнатур эндпоинтов с их имплементациями

## 3.6. Модуль для интеграции системы с фреймворком Express

**Заключение**

**Список литературы**