|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА** – **Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ОТЧЕТ**

**ПО НАУЧНО-ПРАКТИЧЕСКОЙ РАБОТЕ**

**по теме**

«Конфигуратор мультимодульных программных систем»

Студент группы ИКБО-02-15 Апальков П.Ю.

Принял Плотников С.Б

Москва 2018

**Содержание**

Введение 3

1. Анализ мультимодульной архитектуры проектирования ПО 4

1.1. Сравнительный анализ монолитной и мултимодульной архитектуры проектирования 4

1.2. Описание и состав классического модуля системы 6

1.3. Описание методов межмодульного взаимодействия 7

1.4. Конфигурация модуля 9

2. Адаптация мультимодульной программной системы 10

3. Анализ процессов управления конфигурациями 13

3.1. Описание инфраструктурной модели системы 13

3.2. Описание метода управление сервисом с помощью конфигурации 14

3.3. Процесс управления конфигурацией модуля 15

4. Функциональные требования программной системы 18

5. Разработка программного продукта «Конфигуратор мультимодульных программных систем» 19

5.1. Проектирование и разработка серверной части 19

5.2. Проектирование пользовательского интерфейса 25

5.3. Обеспечение безопасности передачи данных и управления модулями 28

6. Функциональное тестирование программной системы 30

Заключение 32

Список литературы 33

Введение

Начиная с 80-х годов прошлого столетия началось активное использование языков программирования для разработки промышленных приложений. Очень долгое время при их создании использовали монолитную архитектуру проектирования, согласно которой вся система являлась большим набором взаимосвязанных элементов. Но в современном мире, когда такие системы должны выполнять все больше и больше различных функций, такой подход сильно усложняет их, в следствие чего появляется все больше багов и время на разработку новых требований так же увеличивается.

С недавнего времени для решения данной проблемы многие предприятия стали использовать мультимодульный принцип проектирования программных систем. В его основу заложено то, что каждая логически обособленная часть функционала выносится в отдельный независимый модуль. В итоге вся система строится на множестве независимых модулей, что делает ее легко расширяемой и легко управляемой [8].

В рамках выпускной квалификационной работы будет рассмотрен принцип мультимодульного проектирование программных систем, а затем, на основе применения конфигурационного моделирования, будет продемонстрирован способ управления настройками (конфигурациями) каждого модуля.

Итогом данной работы будет являться приложение «Конфигуратор» с помощью которого будет происходить управление модулями системы.

1. Анализ мультимодульной архитектуры проектирования ПО

1.1. Сравнительный анализ монолитной и мултимодульной архитектуры проектирования

«Мультимодульная» архитектура проектирования, которую так же называют «микросервисной», является относительно новым термином в области разработки программного обеспечения. К сожалению, довольно часто люди, использующие его, не имеют конечного понимания того, что же данная архитектура из себя представляет.

Таким образом, в первую очередь определим, какие особенности имеет мультимодульная архитектура и сравним ее с классическим «монолитом», т.е. системы, работающей как единое целое [1].

При использовании монолитной архитектуры разработчики сталкиваются со следующими проблемами:

* необходимо ориентироваться в большом объеме кода;
* при добавлении нового функционала, его декомпозиция сильно ограничена, что приводит к большим временным, и, следовательно, финансовым затратам со стороны заказчика;
* при выходе из стоя какого либо компонента большой системы, вся система выходит из строя, что является неприемлемым для многих предприятий;
* сильная ограниченность в используемых технологиях;
* масштабирование может быть осуществлено только посредством копирования системы на отдельный сервер. Данная проблема так же обусловлена тем, что если необходимо увеличить производительность только одного элемента системы, т.к. остальные вычислительные мощности расходуются впустую.

На рисунке 1.1 представлена иллюстрация системы, имеющую монолитную архитектуру.

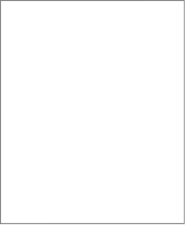
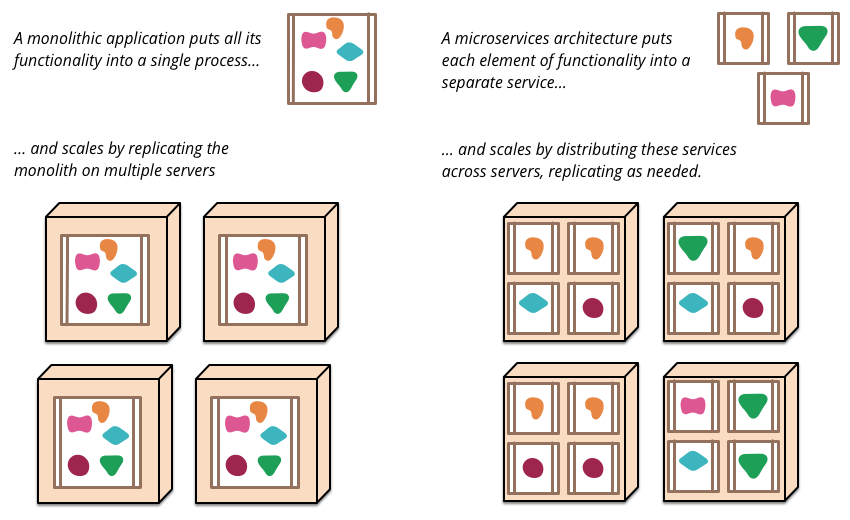


Рисунок 1.1 - Иллюстрация «монолитной» системы

Проектирование системы на основе независимых модулей по сути является решением вышеперечисленных проблем. Рассмотрим иллюстрацию построения системы с помощью мультимодульной архитектуры (рисунок 1.2).

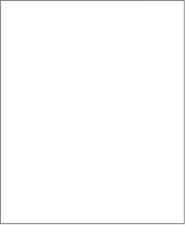
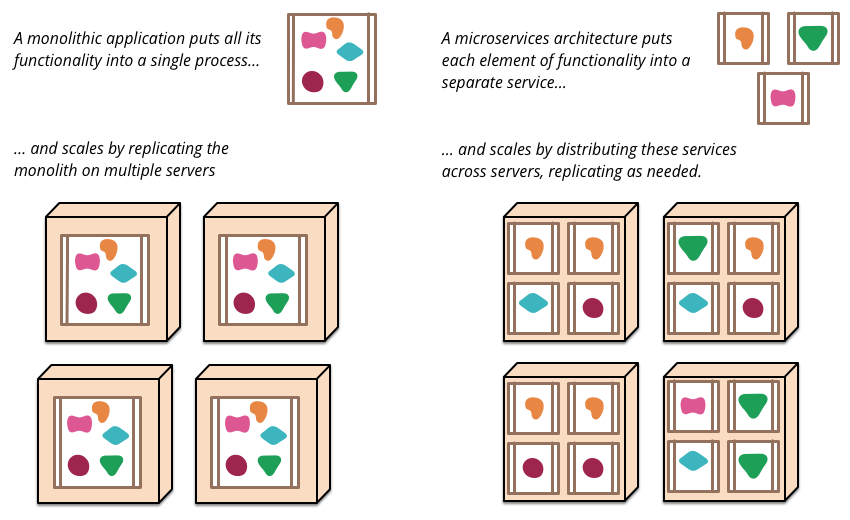


Рисунок 1.2 - Иллюстрация «монолитной» системы

На ней видно, что каждый компонент системы обособлен и не зависит от всех остальных компонентов. Так же при необходимости увеличения производительности можно делать копию только того модуля, для которого это требуется.

Еще одним преимуществом такой архитектуры является простая декомпозиция задач при реализации новых требований, т.к. каждый модуль может разрабатываться не зависимо от готовности других модулей.

Таким образом, можно сделать вывод, что проектирование систем на основе модулей является предпочтительным решением.

1.2. Описание и состав классического модуля системы

Рассмотрим более подробно принцип построения мультимодульных программных систем. Как говорилось ранее, в основе их архитектуры лежит понятие модуля, а так же его взаимодействия с остальными модулями системы.

Каждый модуль должен выполнять определенную, свойственную только ему, логику, реализация которой представлена набором определенных методов. Для удобства использования данных методов, их объединяют в единый «сервис».

Сервис представляет собой совокупность методов, а так же интерфейса для работы с ними (рисунок 1.3). Данный интерфейс по сути является спецификацией сервиса, в которой указан идентификатор самого сервиса, а так же идентификатор каждого его метода [5].

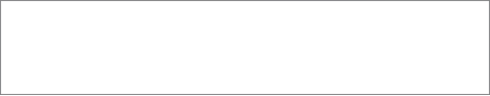
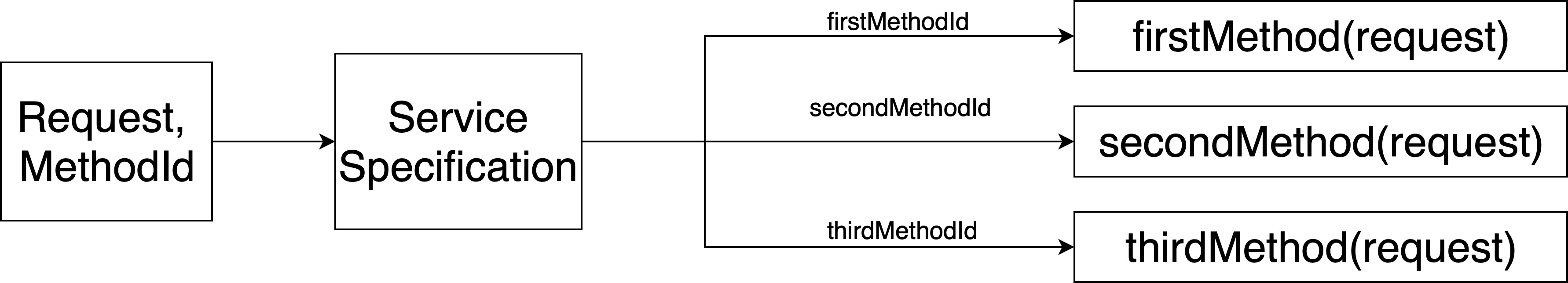


Рисунок 1.3 - Схема классического сервиса

Проектирование модуля на основе сервисов дает возможность другим модулям вызывать его методы, используя лишь вышеуказанные идентификаторы. Такой подход позволяет инкапсулировать логику работы модуля от внешних модулей, что является хорошим тоном в кругах разработки.

Таким образом мы плавно подошли к вопросу межмодульного взаимодействия.

1.3. Описание методов межмодульного взаимодействия

Полноценная работоспособность системы невозможна без модульного взаимодействия. Рассмотрим, каким образом можно обмениваться данными между модулями.

Первым, самым простым способом является включение сервисного модуля в состав другого модуля (включение его артефакта в программный код) и вызов метода сервиса напрямую (рисунок 1.4). Минусом данного подхода является то, что нарушается правило инкапсуляции, а так же сформированный артефакт исходного модуля будет сильно расширен за счет включения в него всего кода вызываемого им модуля. Но стоит отметить, что такой подход допустим при вызове методов из модулей, которые не содержат в себе сервисы, а только некоторые «библиотечные» функции.

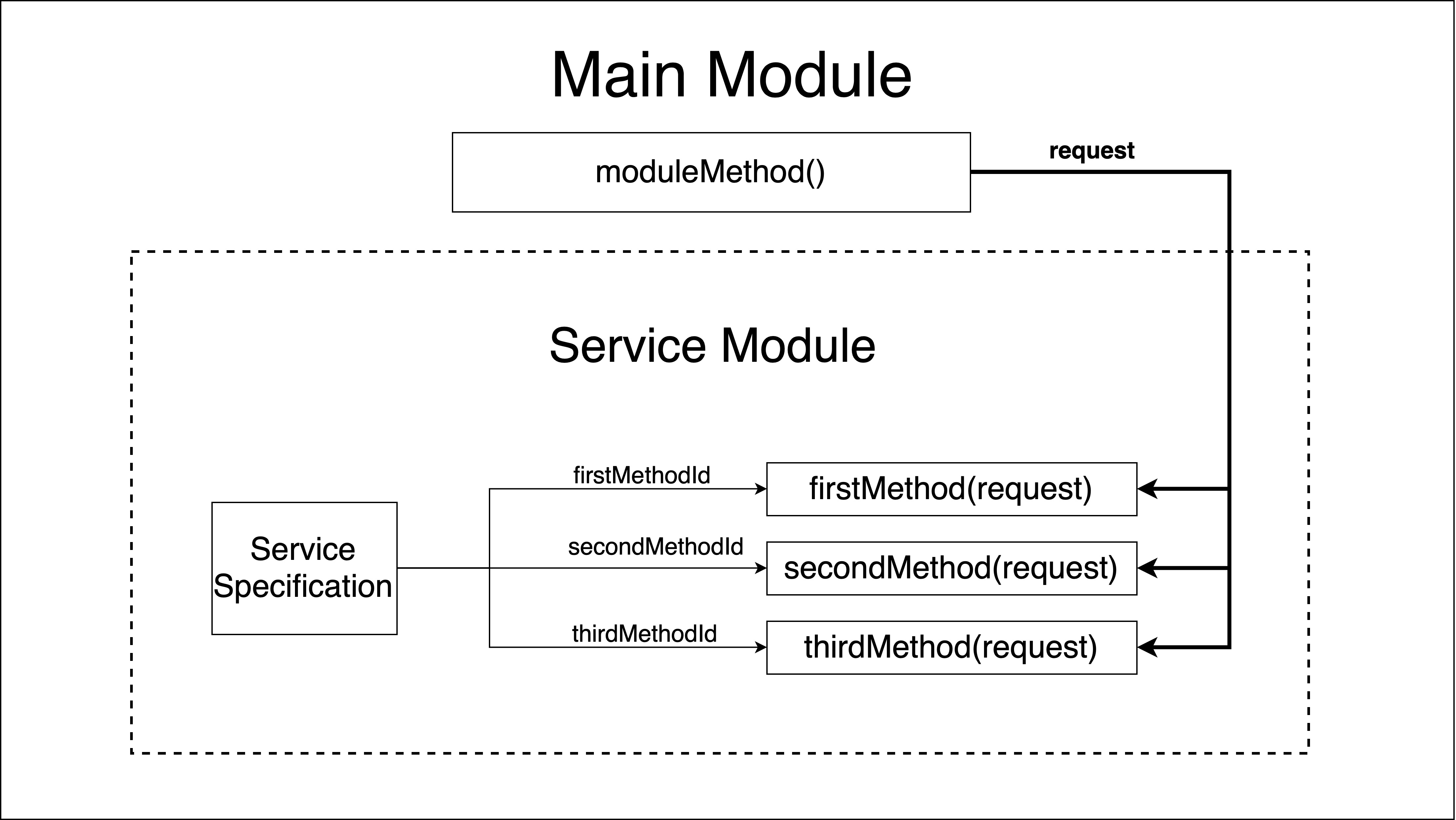


Рисунок 1.4 - Схема вызова метода сервиса напрямую

Второй способ взаимодействия состоит в том, что бы вызывать метод сервиса через выставленный интерфейс (рисунок 1.5). При этом подходе решается проблема инкапсуляции, однако, сервисный модуль так же необходимо включить в состав исходного модуля. Но в тех случаях, когда вызываемый сервис инкапсулирует небольшой объем логики и имеет маленький объем такой способ взаимодействия допускается.

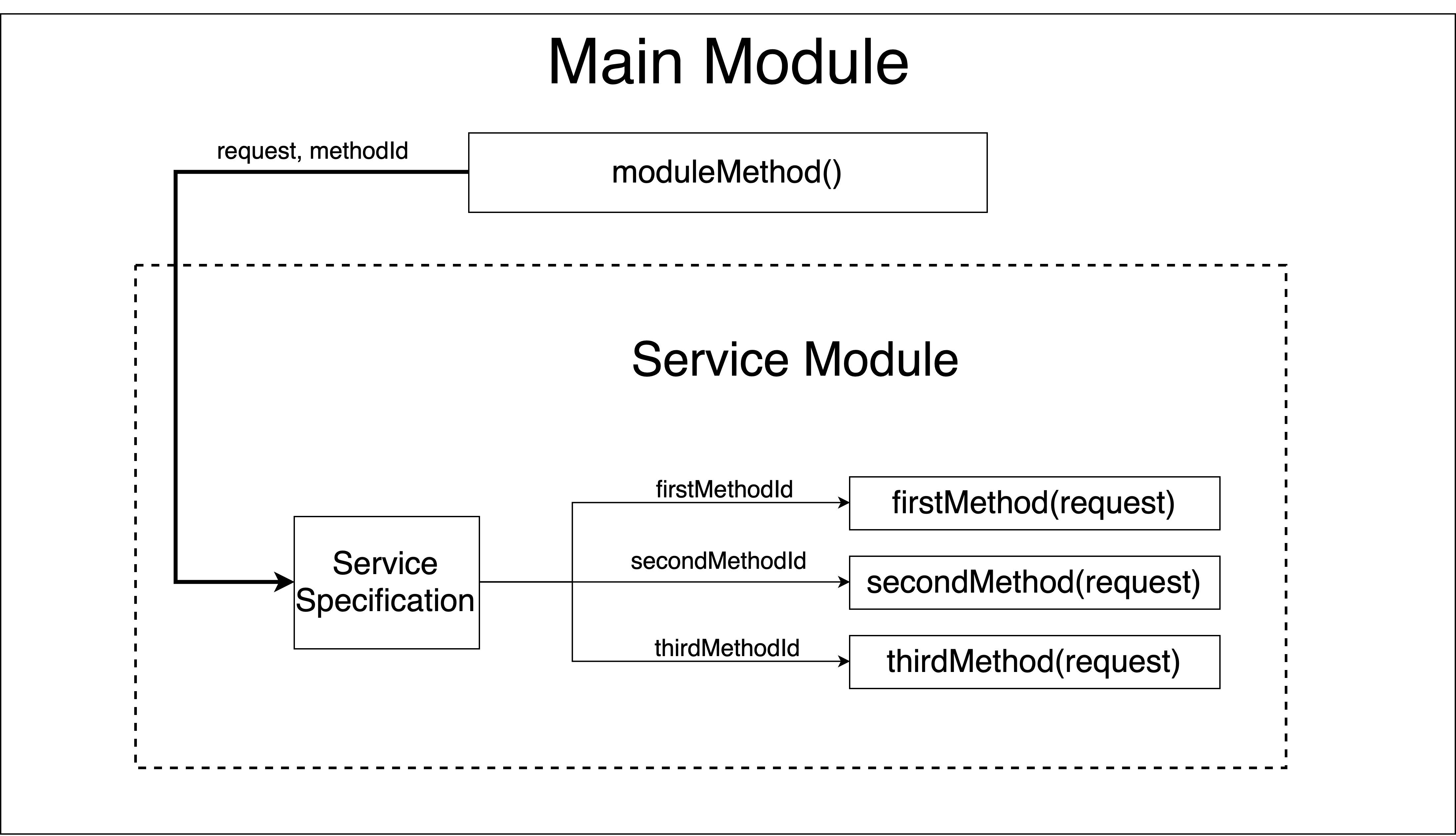


Рисунок 1.5 - Схема вызова метода сервиса через его интерфейс

Третьим и наиболее предпочтительным способом является организация сетевого взаимодействия между модулями (рисунок 1.6.). Для этого в каждый модуль включается http/protobuf сервер и выставляются соответствующие интерфейсы сервисов. На первый взгляд может показаться, что включение сервера в модуль усложнит его архитектуру, но на самом деле это заметно повысит удобство использования данного модуля. При таком подходе решаются все проблемы, описанные выше, а так же появляется возможность реализации модулей с помощью разных языков программирования и технологий. Но для такого взаимодействия необходимо определить единый формат передачи данных или выбрать единую структуру данных [7].

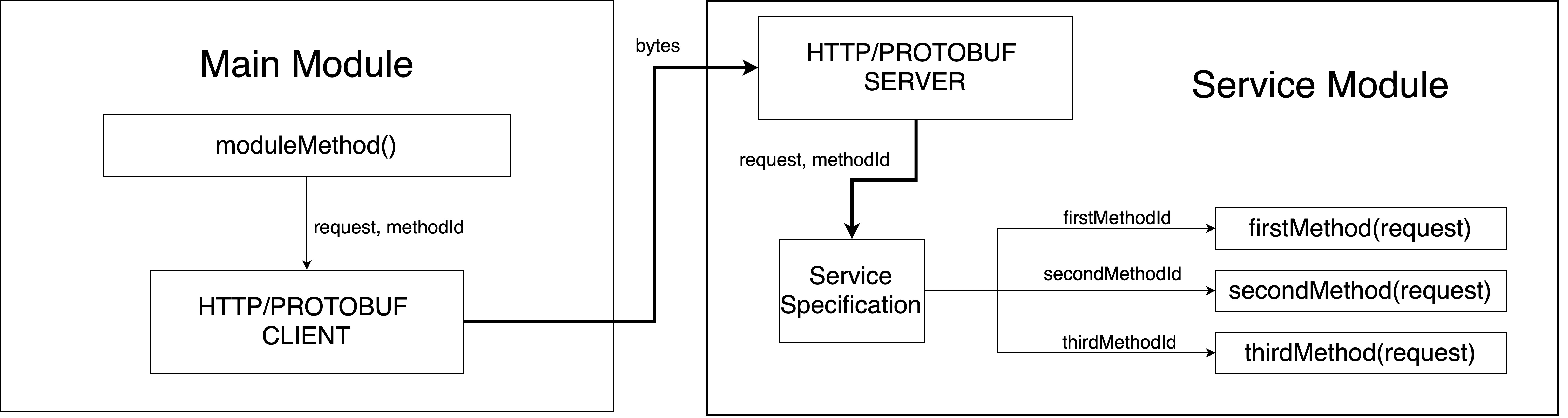


Рисунок 1.6 - Схема вызова метода сервиса по сети

1.4. Конфигурация модуля

В процессе работы сервис модуля может использовать параметры, которые можно отнести к настраиваемым параметрам, значения которых могут изменяться в зависимости от условий эксплуатации модуля. Примерами таких параметров могут являться различные «флаги», отвечающие за активацию/деактивацию какой-либо части функционала, ip адреса внешних модулей и систем, ip адрес балансировщика, а так же любые другие параметры, значение которых будет необходимо изменить в ходе работы модуля без дополнительный сборки его артефакта.

Совокупность таких параметров определяет конфигурацию модуля, которая по факту является его основной характеристикой [15].

Стоит отметить, что конфигурация не задаёт поведение работы модуля, а только лишь управляет значениями параметров, которыми оперирует программная логика модуля, но при этом с ее помощью можно переключаться между режимами работы модуля (активация / деактивация).

Рассмотрим, каким образом можно задать модулю его конфигурацию. В основном выделяют три способа:

* непосредственно в коде (дефолтная конфигурация);
* в ресурсном файле (файловая конфигурация)
* с помощью внешнего запроса (удаленная конфигурация)

Последний способ представляет особый интерес, так как таким образом можно менять значения параметров в режиме реального времени, соответственно не будет оказано никакого влияния на пользователей системы. Но для его реализации модуль необходимо адаптировать, используя подход конфигурационного моделирования.

2. Адаптация мультимодульной программной системы

Для решения задачи удаленного конфигурирования модуля, при его проектировании необходимо использовать методологии конфигурационного моделирования и адаптации. Рассмотрим более подробно эти понятия.

Конфигурационное моделирование предоставляет ряд методов, позволяющих смоделировать и спроектировать программный комплекс, содержащий в себе явный или же неявный механизм конфигурации. Ключевым понятием конфигурационного моделирования является понятие конфигуратора.

Конфигуратор – подсистема, ключевой целью которой является конфигурирование и реконфигурирование основной системы путем изменения значений настраиваемых параметров каждого модуля системы [13].

Таким образом, основным объектом, на который воздействует конфигуратор является программный модуль, так как именно их совокупность представляет собой полноценную работающую систему.

Сам конфигуратор, в свою очередь, может быть реализован в рамках отдельного конфигурационного модуля, в этом случае систему можно считать автономной с точки зрения конфигурирования, либо в роли конфигуратора может выступать другая программная система, работающая в той же среде, что и основная. Во втором случае случае появляется задача создания канала управления, который позволял бы без потерь передавать запросы конфигуратора к модулям системы, а также сигналы от них к конфигуратору.

Теперь, с учетом знаний о составе модулей и учитывая методологию конфигурационного моделирования, рассмотрим концептуальную модель конфигурируемой программной системы, состоящей из множества модулей (1).

CS = <M, MC, MR, C, B> (1)

* M – множество модулей
* SM – множество состояний модулей
* CM – множество конфигураций модулей
* DM – взаимосвязи модулей
* C – конфигуратор
* B – балансировщик запросов между модулями (речь о нем пойдет позже)

Основная реализация механизма управления конфигурациями ложится на плечи конфигуратора и балансировщика и осуществляется посредством асинхронного обновления параметров на стороне конфигуратора с последующим сигналом на запрос обновленной конфигурации ко всем модулям [2]. То есть модуль принимает запрос на обновление своей конфигурации и сам запрашивает ее у конфигуратора. При этом он может запрашивать только те элементы своей конфигурации, которые ему необходимы. Схематично данный процесс можно отобразить следующим образом (рисунок 2.1)



Рисунок 2.1 - Схема последовательных операций

в управлении конфигурациями

Так как модуль должен иметь возможность запросить у конфигуратора необходимые ему параметры конфигурации, этот модуль необходимо адаптировать. В виду того, что существует несколько уровней адаптации, рассмотрим их более подробно и определим, какой из них необходим для решения поставленной задачи.

Существует пять основных уровней адаптации программных систем:

* Аппаратный уровень. На данном уровне регулируются аппаратные параметры и аппаратная структура системы.
* Алгоритмический уровень. Здесь объектом адаптации является алгоритмическая составляющая системы, ключевая задача управления на этом уровне – реорганизация используемых в системе алгоритмов под новые требования решаемой задачи.
* Программный уровень адаптации обеспечивает регулирование используемого множества доступных программ под требования к системе в данный момент времени, а так же включает в себя адаптацию системы на уровне кодовой базы.
* Системный уровень адаптации улучшает функционирование системного обеспечения программной системы.
* Сетевой уровень адаптации связан с адаптацией процессов передачи данных между программными системами.

Исходя из перечисленных уровней можно сказать, что адаптация модуля для взаимодействия с конфигуратором затрагивает сетевой уровень (для создания возможности передачи данных по сети), а так же программный уровень (для обработки входящих запросов и инициализации запроса на получение конфигурации).

Проанализировав состав модуля и уровни его адаптации изобразим схему типового модуля системы (рисунок 2.2)

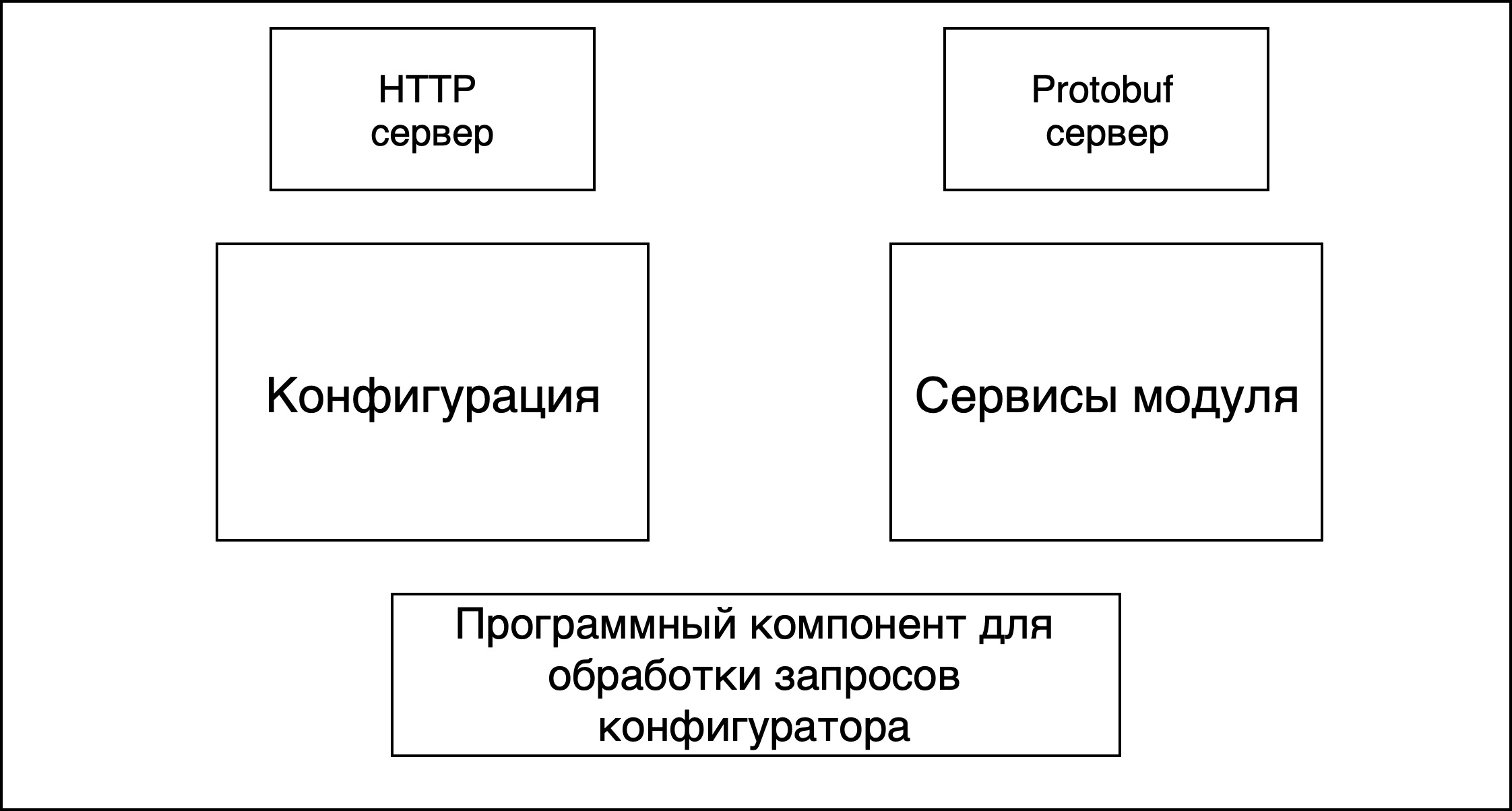


Рисунок 2.2 - Схема типового модуля системы

3. Анализ процессов управления конфигурациями

3.1. Описание инфраструктурной модели системы

Для анализа процесса управления конфигурациями рассмотрим систему, состоящую из нескольких модулей. При этом будем учитывать место конфигуратора в этой системе. Определим инфраструктуру модель системы (рисунок 3.1).

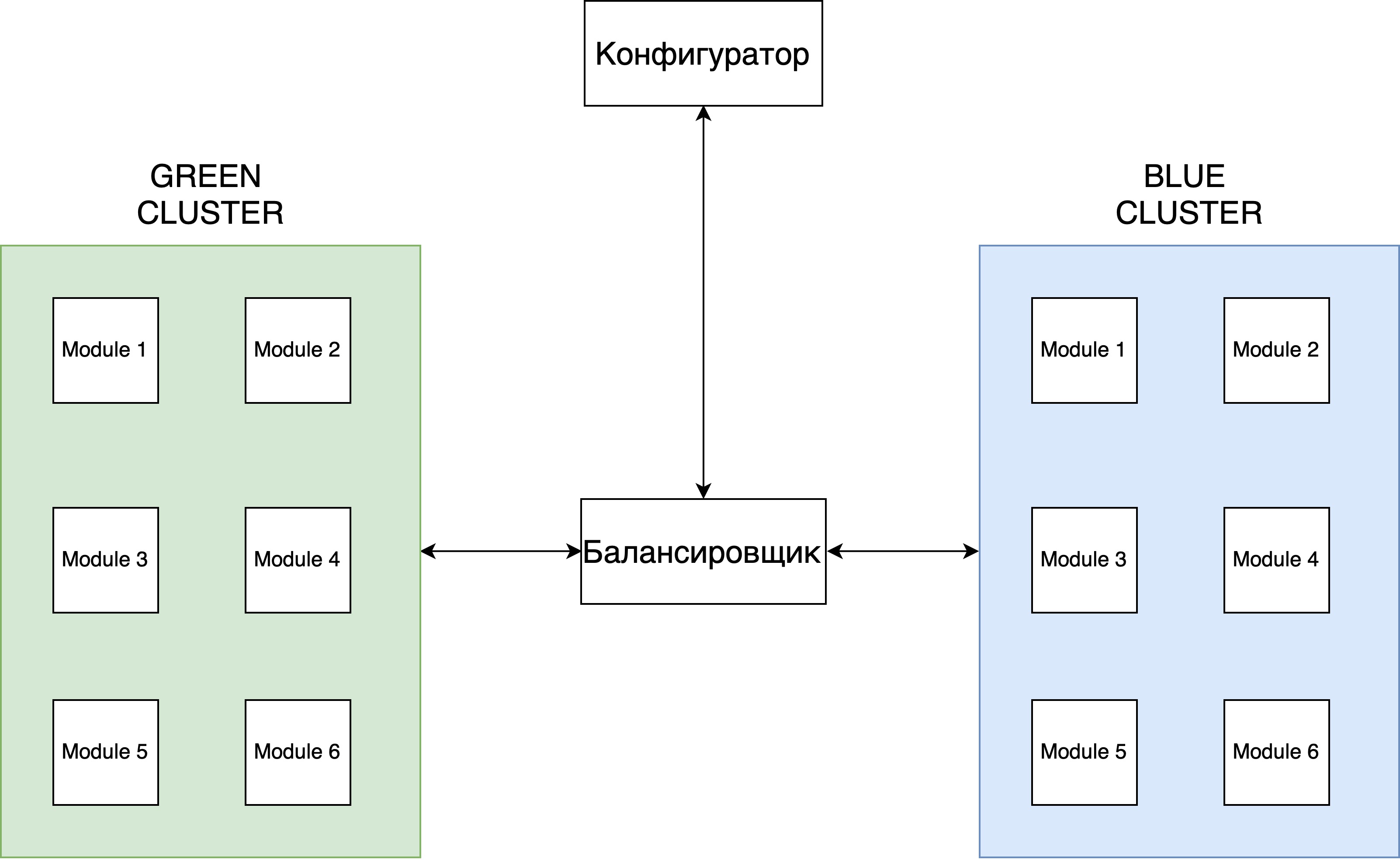


Рисунок 3.1 - Схема инфраструктурной модели системы

На данном рисунке продемонстрирована схема современной развертки приложений по blue-green схеме, представляющие собой два кластера с запущенными приложениями. Такая схема удобна тем, что в любой момент времени активен только один из кластеров и в случае необходимости идет быстрое переключение на второй кластер. Это обеспечивает бесперебойную работу системы в случае возникновения каких-либо внештатных ситуаций или же при проведении на кластере технических работ.

Как видно из схемы, все взаимодействие между кластерами происходит с помощью балансировщика, но так как конфигуратор не относится к системе, он вынесен за пределы данных кластеров.

Что бы конфигуратор мог изменить конфигурацию определенного модуля в нужном кластере, в запросе на обновление конфигурации он будет так же передавать информацию с идентификатором модуля и необходимым кластером на балансировщик, где данный запрос будет переправляться (рисунок 3.2).

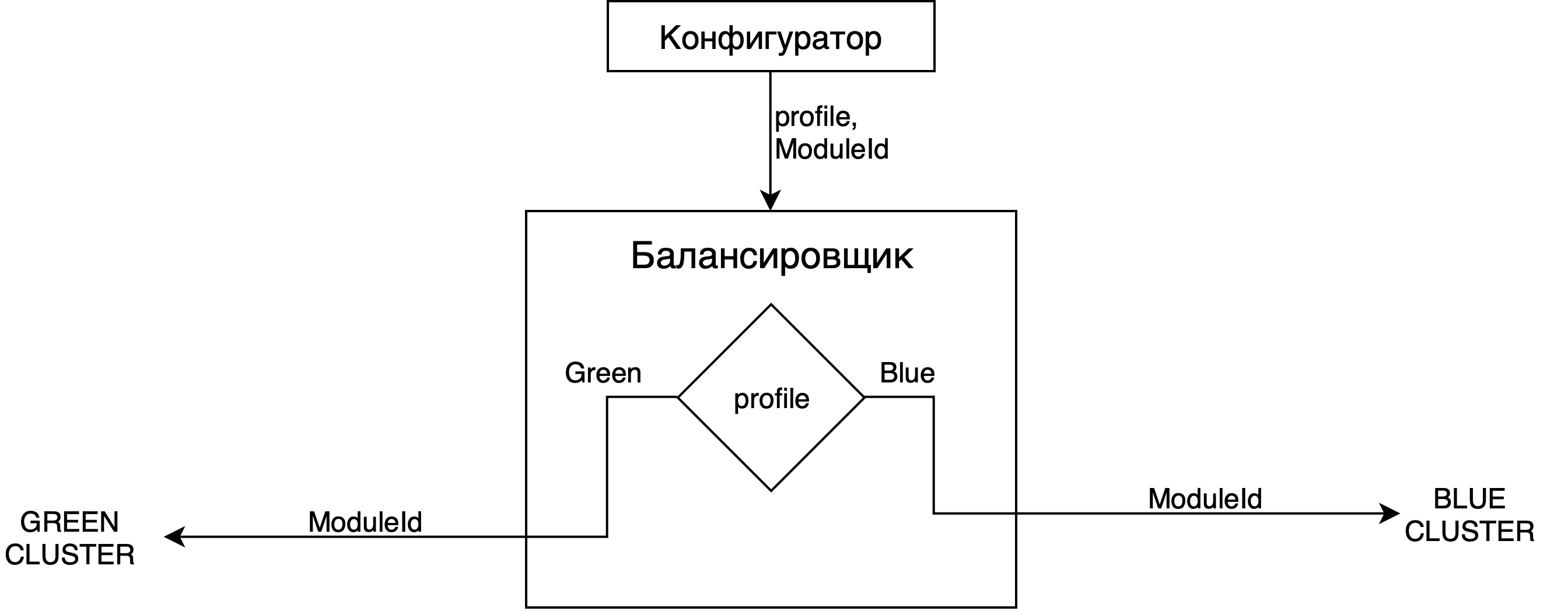


Рисунок 3.2 - Схема перенаправления запроса от конфигуратора

3.2. Описание метода управление сервисом с помощью конфигурации

Основным объектом, с которым взаимодействует конфигуратор является модуль, поэтому далее будем рассматривать именно это взаимодействие, опуская промежуточные узлы.

Перед тем, как рассмотреть подробное взаимодействие конфигуратора и модуля, опишем, как конфигурация последнего может влиять на его работоспособность. Для этого изобразим на схеме (рисунок 3.3) конфигурацию с параметрами, отвечающую за активацию сервиса и его методов и место этих параметров в структуре сервиса.

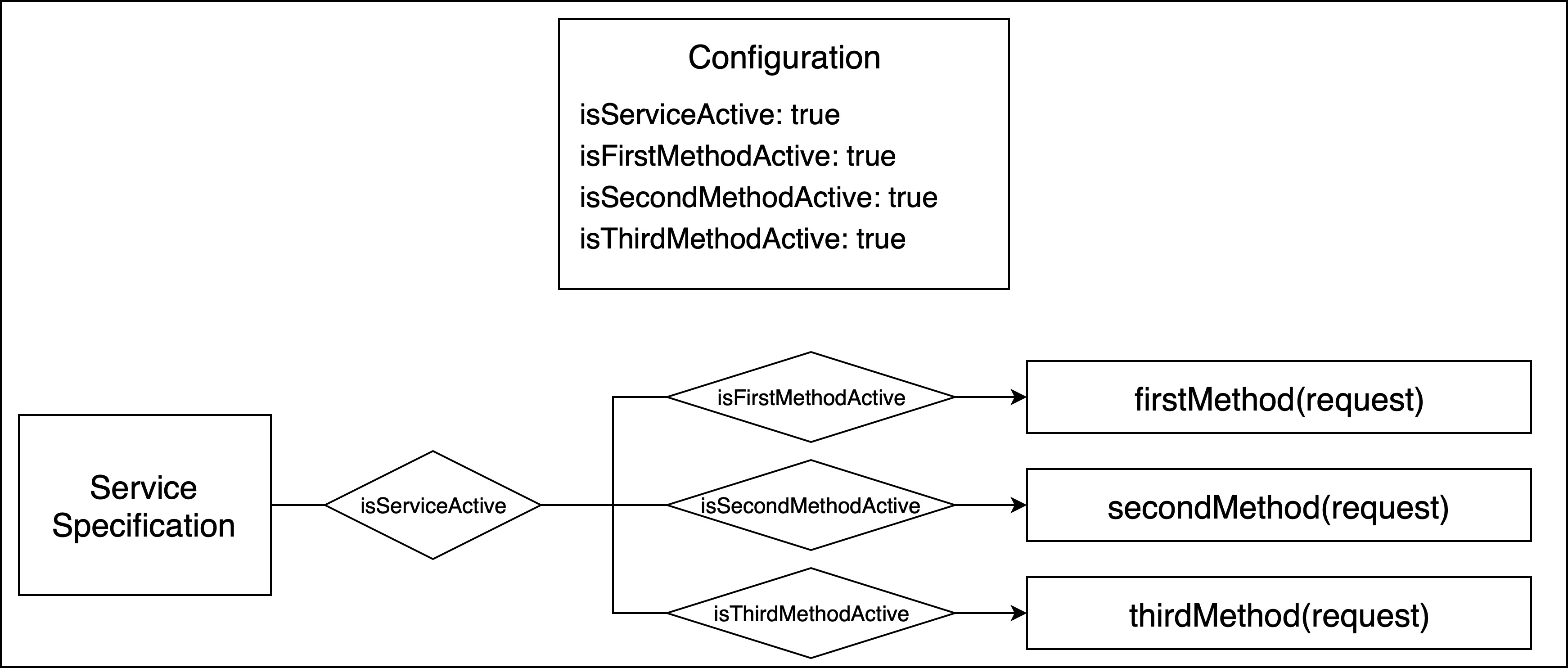


Рисунок 3.3 - Схема модуля с подробной конфигурацией

Перед тем, как вызвать любой из методов сервиса идет проверка на активность этого сервиса (проверяется значение параметра из конфигурации), а затем аналогичным образом идет проверка на активность метода сервиса. На рисунке 3.4 показан пример, где один из методов сервиса деактивирован, а на рисунке 3.5 продемонстрирована деактивация всего сервиса.

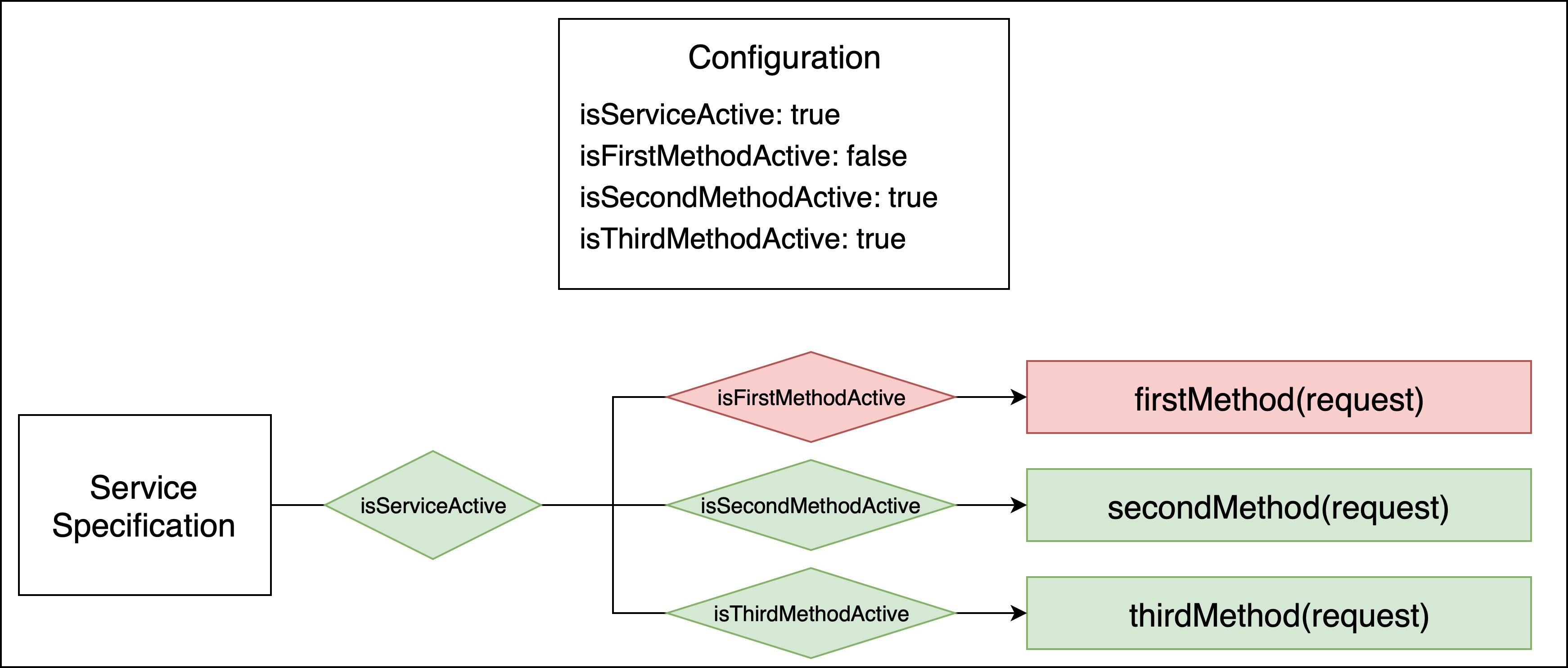


Рисунок 3.4 - Схема модуля с деактивированным методом

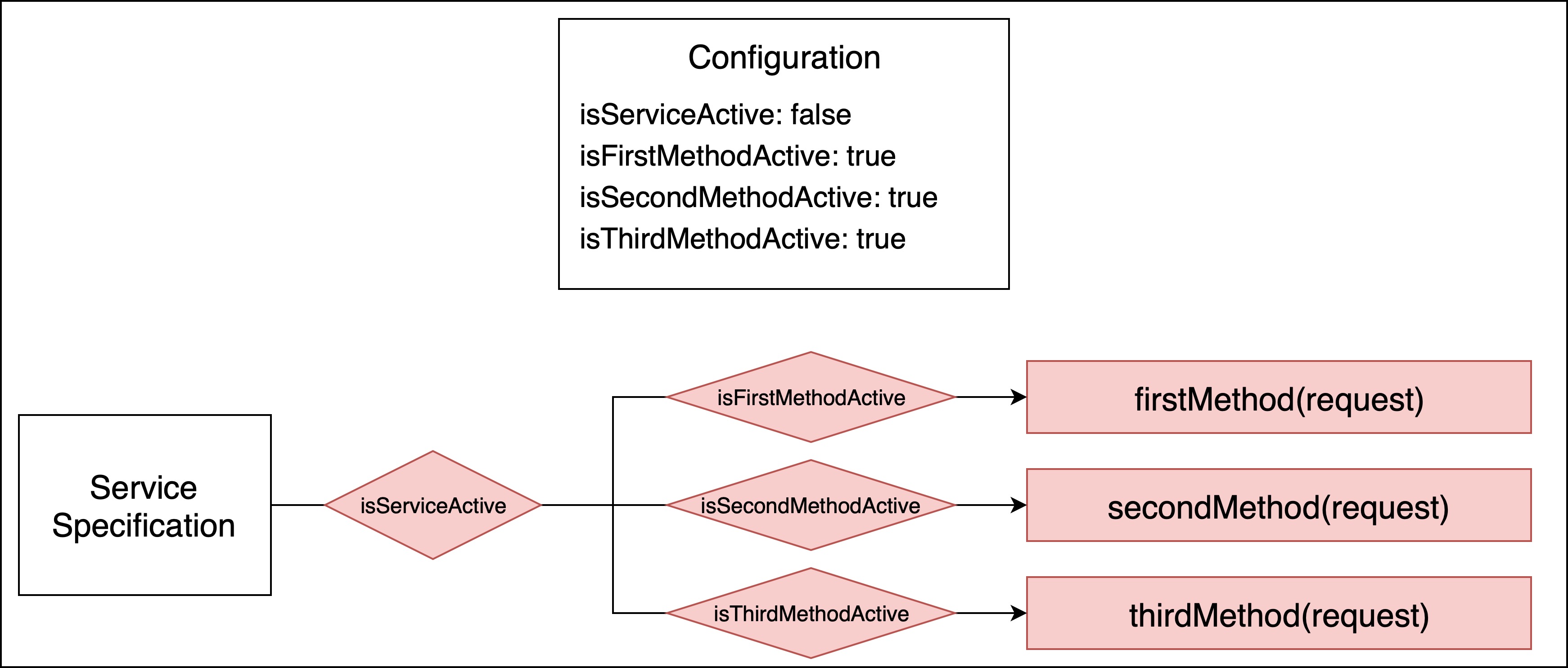


Рисунок 3.5 - Схема модуля с деактивированным сервисом

3.3. Процесс управления конфигурацией модуля

Теперь, когда рассмотрен пример использования конфигурации модуля, рассмотрим процесс управления конфигурацией с помощью конфигуратора.

Для того что бы модуль мог принять запрос на обновление всех его параметров, в него необходимо включить соответствующий компонент, который представлен в виде «библиотечного модуля» (рисунок 3.6). При получении запроса от конфигуратора, данный компонент вызывает метод refresh, присутствующий в классе конфигурации, где происходит запрос значения каждого параметра.

Для осуществления данного запроса в модуль так же необходимо включить компонент, в котором будет происходит генерация запроса в конфигуратор и его отправка с дальнейшим получением необходимого значения.

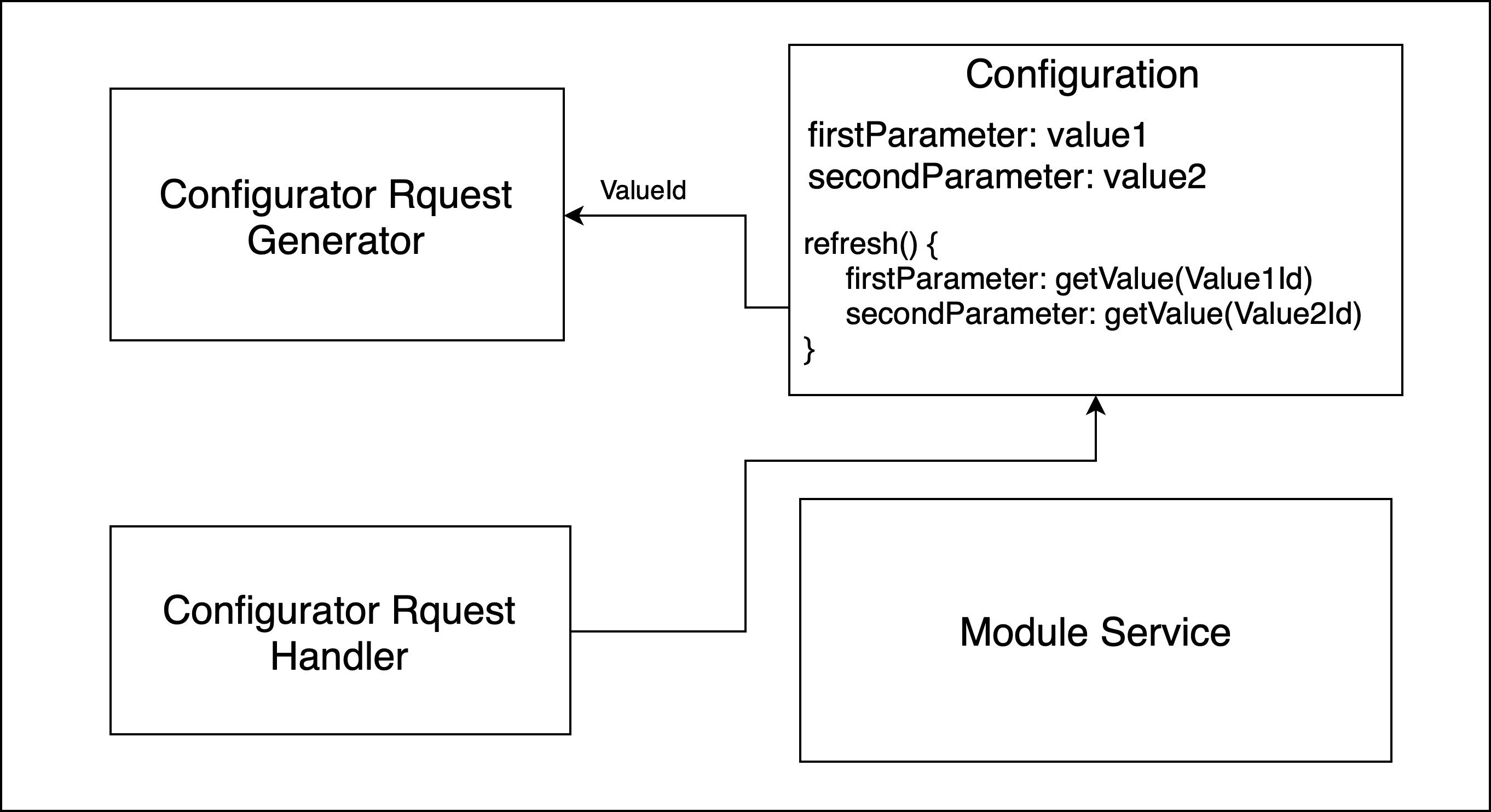


Рисунок 3.6 - Схема модуля с компонентами

для работы с конфигуратором

Добавим в эту схему сам конфигуратор (рисунок 3.7) и опишем процесс управления конфигурацией модуля с его помощью.

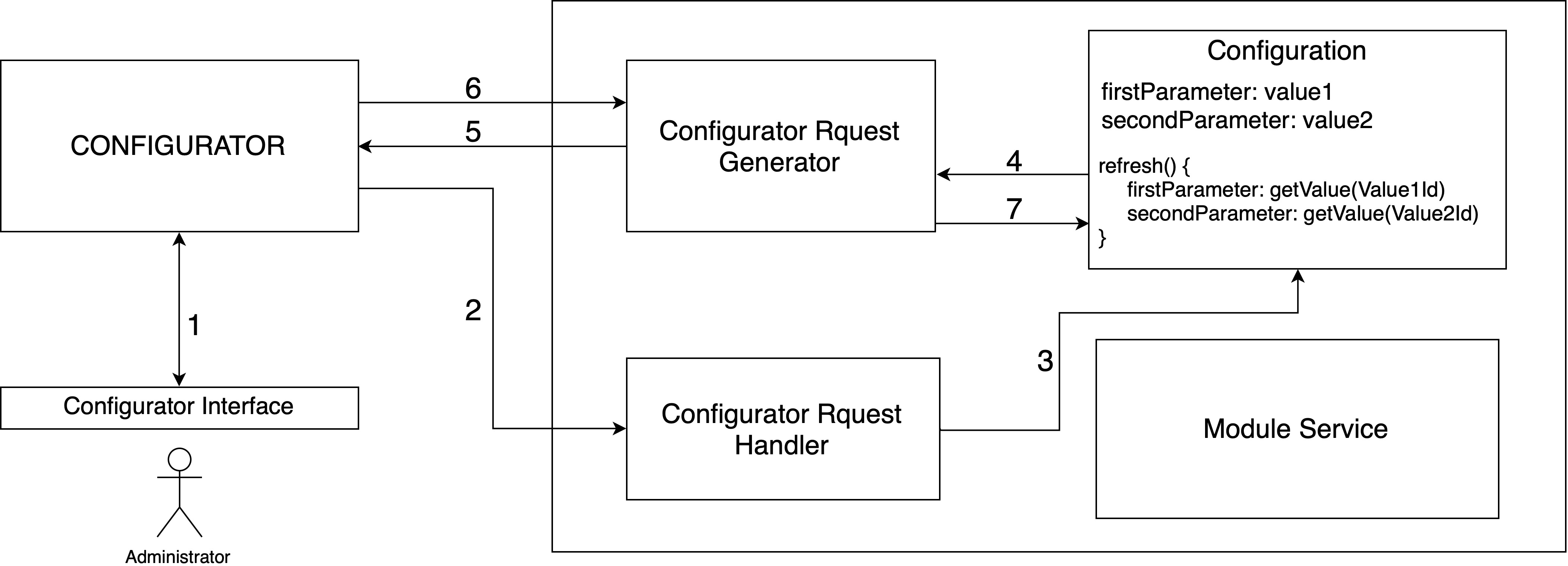


Рисунок 3.7 - Схема управления конфигурацией модуля

Цифрами обозначены следующие события:

1 - администратор с помощью интерфейса обновляет конфигурацию у определенного модуля в определенном кластере и отправляет ее в конфигуратор,

2 - конфигуратор отправляет запрос на необходимость обновления конфигурации,

3 - внутри модуля вызывается метод refresh, в котором происходит обновление каждого параметра,

4 - в компонент генерации запроса в конфигуратор передается информация о том параметре, значение которого необходимо запросить,

5 - запрос информации в конфигураторе,

6 - ответ конфигуратора,

7 - параметр обновляет свое значение.

Таким образом, был описан процесс управления конфигурацией модуля в большой системе и влияние конфигурации на работу сервиса модуля.

4. Функциональные требования программной системы

Описав, каким образом происходит обновление конфигурации модуля с помощью конфигуратора, приступим к формализации требований, которым он должен удовлетворять.

Программная система «Конфигуратор мультимодульных программных систем» должна:

1. Иметь графический интерфейс, обеспечивающий удобство выполнения следующих действий:
   * просмотр доступных профилей,
   * добавление нового профиля,
   * просмотр всех модулей определенного профиля,
   * добавление модуля в определенный профиль,
   * просмотр конфигурации определенного модуля,
   * изменение и сохранение конфигурации определенного модуля,
   * просмотр параметров балансировщика (хост/порт),
   * изменение и сохранение параметров балансировщика.
2. Обеспечивать хранение конфигураций модулей в базе данных с быстрым доступом.
3. Обновлять конфигурации модулей.
4. Передавать данные между пользователем и системой в формате JSON, а между модулями в виде protobuf bytes.
5. Иметь возможность изменения конфигурации любого модуля.
6. Иметь выставленный http интерфейс для использования конфигуратора не только через интерфейс, но и через другие системы (к примеру Postman).
7. Иметь выставленный protobuf интерфейс для возможности использования сервисами конфигуратора другими модулями.

5. Разработка программного продукта «Конфигуратор мультимодульных программных систем»

Учитывая требования, предъявляемые к системе, а так же принцип, по которому происходит обновление конфигурации модуля, разберем реализацию программной системы «Конфигуратор». В начале разберем то, каким образом будет реализована серверная часть данного приложения, так как она является ключевым элементом, а затем спроектируем её графический вэб-интерфейс.

5.1. Проектирование и разработка серверной части

Перед тем, как непосредственно приступить к разработке серверной части, опишем те технологии, которые будут принимать участие в разработке.

5.1.1. Универсальная структура данных

Как говорилось ранее, для работы системы на мультимодульной архитектуре необходимо выбрать единый формат передачи данных или же единую структуру данных. В результате проектирования была создана универсальная структура данных «Value», способная инкапсулировать в себе любой тип данных.

Разберем, из чего данная структура состоит. Для возможности хранения любого типа данных было реализовано несколько классов, имеющих один общий интерфейс Value (рисунок 5.1).

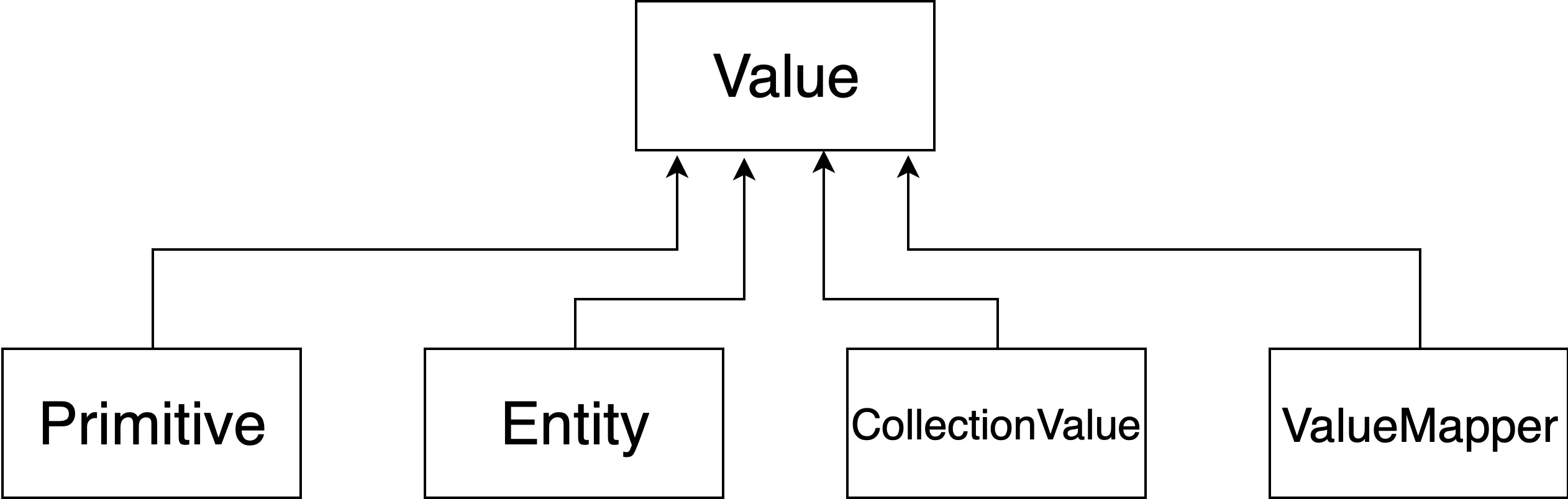


Рисунок 5.1 - Схема универсальной структуры данных

Разберем содержание каждого класса:

* Value - интерфейс, который реализуется всеми обертками над данными,
* Primitive - класс-обертка над примитивными типами данных, но так же включает в себя класс String.
* CollectionValue - обертка над классами-коллекциями. Содержит функционал по работе как с массивами, так и с Collection.
* Entity - наиболее функциональная структура. Представляет собой пару String -> Value, где Value может быть реализовано любым вышеперечисленным классом.
* ValueMapper - функциональный объект, выполняющий две задачи в зависимости от использования:

- создание Value из объекта определенного класса,

- создание объекта класса из Value.

Роль мапперов для данной структуры имеет высокое значение. Так как данная структура данных выбрана для передачи любой информации между модулями, необходимо, что бы у тех классов, которые используются внутри модулей (если предполагается обмениваться объектами данных классов) были мапперы в Value и обратно.

Рассмотрим, каким образом на уровне мапперов происходит передача объекта типа String между модулями (рисунок 5.2).

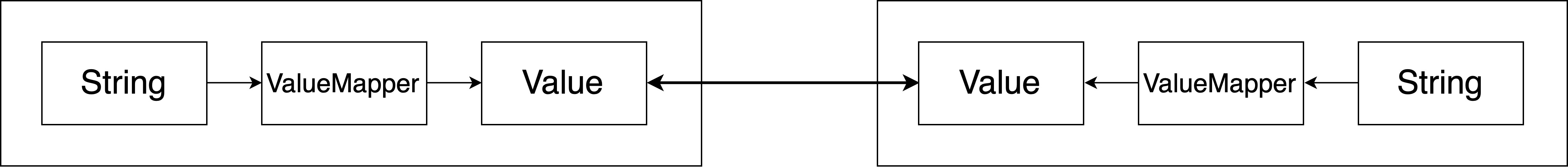


Рисунок 5.2 - Схема передачи объекта между модулями

В исходном модуле создается объект типа String, с которым происходят какие-либо действия. Для того, что бы передать этот объект другому модулю, с помощью StringMapper.fromModel() его конвертируют в Primitive (который является Value), и передают другому модулю. Стоит отметить, что при передачи объектов с помощью Value, оба модуля, использующие этот объект, должны знать его структуру и использовать одинаковые мапперы. При приеме объекта другим модулем происходит его конвертирование в исходный тип String (с помощью StringMapper.toModel()), после чего модуль может обрабатывать данных объект.

5.1.2. Проектирование хранилища данных c быстрым доступом (RocksDb)

Для хранения конфигураций модуля было выбрана база данных RocksDb. RocksDB представляет собой гибкое производительное встраиваемое NoSQL хранилище, интерфейс которого представлен в таблице 5.1. Особенностью данного хранилища является то, что хранение данных в нем реализовано с помощью key-value метода, в котором каждому элементу данных присваивают уникальный ключ. Так же положительным моментом в использовании данного хранилища является API для языков C++ и Java (на котором будет разработка).

Таблица 5.1.

Методы интерфейса базы данных RocksDB

| **Метод** | **Описание метода** |
| --- | --- |
| put(key, value) | Добавление значения и указание его ключа |
| get(key) | Получение значения по его ключу |
| delete(key) | Удаление значения по его ключу |
| merge(key, value) | Добавление значения в конец предыдущего значения (хранения в виде списка). |

Рассмотрим сущности, которые необходимо хранить для обеспечения работы системы, более подробно (таблица 5.2).

Таблица 5.2.

Описание сущностей в базе данных

| **Ключ** | **Значение (данные)** | **Комментарий** |
| --- | --- | --- |
| Ключ модуля  (идентификатор модуля + идентификатор профиля) | Конфигурация модуля | Основная сущность. Будет использоваться для определения конфигурации модуля по его ключу |
| «APPLICATION» | Хост/порт балансировщика | Данная информация необходима для отправки запросов во внешние системы |
| «MODULES» | Список моудлей (в виде его ключа), входящих в состав приложения | Данные списки необходимы для удобства пользованием приложения. В дальнейшем понадобятся для реализации сервиса конфигуратора. |
| «PROFILES» | Список всех известных профилей приложения |

5.1.3. Протоколы передачи данных

Для передачи информации между модулями и вызовами методов сервисов разных модулей было принято решение использовать фрэймворк gRPC, детище компании Google для удаленного вызова процедур (Remote Procedure Call).

Эффективность данного фреймворка обусловлена тем, что для передачи данных используется «Protocol Buffers» или же просто Protobof - протокол сериализации (передачи) структурированных данных, предложенный Google как эффективная бинарная альтернатива текстовому формату XML. Разработчики сообщают, что Protocol Buffers проще, компактнее и быстрее, чем XML, поскольку осуществляется передача бинарных данных, оптимизированных под минимальный размер сообщения.

Так же данный протокол не зависит от тех средств, с помощью которых реализован модуль приложения (рисунок 5.3)

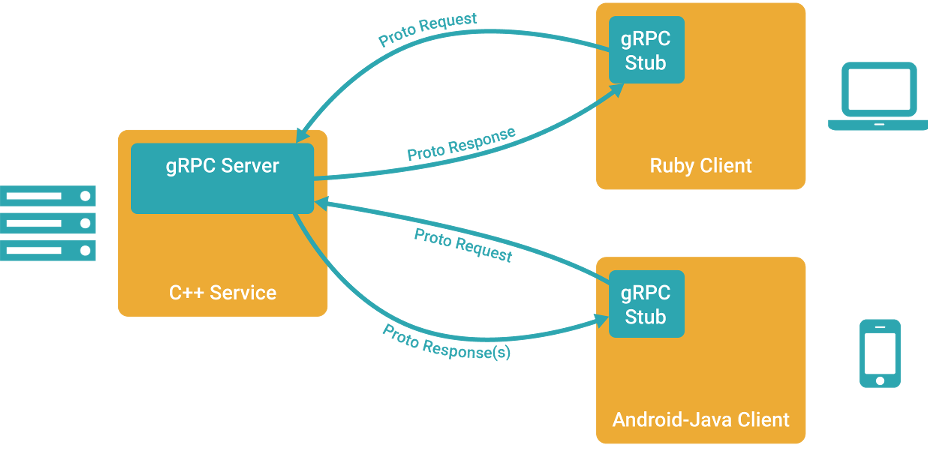


Рисунок 5.3 - Схема взаимодействия между

модулями на разных языках прогаммирования по Protobuf

Однако, невозможно полностью отказаться от стандартного протокола передачи данных HTTP. С его помощью будет передаваться вся информация между администратором и конфигуратором.

В качестве клиента, помимо вэб-интерфейса, для взаимодействия с конфигуратором можно использовать такое приложение как Postman.

5.1.4. Описание реализации серверной части приложения

Описав универсальный тип данных для передачи между модулями, используемую базу данных и протоколы передачи данных между модулями, мы плавно подошли к разработке самого модуля конфигуратора.

В первую очередь, определим, какие методы должны быть предоставлены интерфейсом сервиса конфигуратора, а затем проанализируем каждый из них.

Интерфейс сервиса конфигуратора должен иметь следующие методы:

* Добавление/обновление конфигурации;
* Получение конфигурации модуля по его ключу;
* Обновление конфигурации модуля;
* Получение списка всех профилей;
* Получение списка всех модулей;
* Обновление значений хост/порт балансировщика;
* Получение значений хост/порт балансировщика.

Добавление и обновление конфигурации модуля является одним из основных методов сервиса конфигуратора. Он принимает на вход конфигурацию в виде json (рисунок 5.4), которая преобразовывается в Entity.

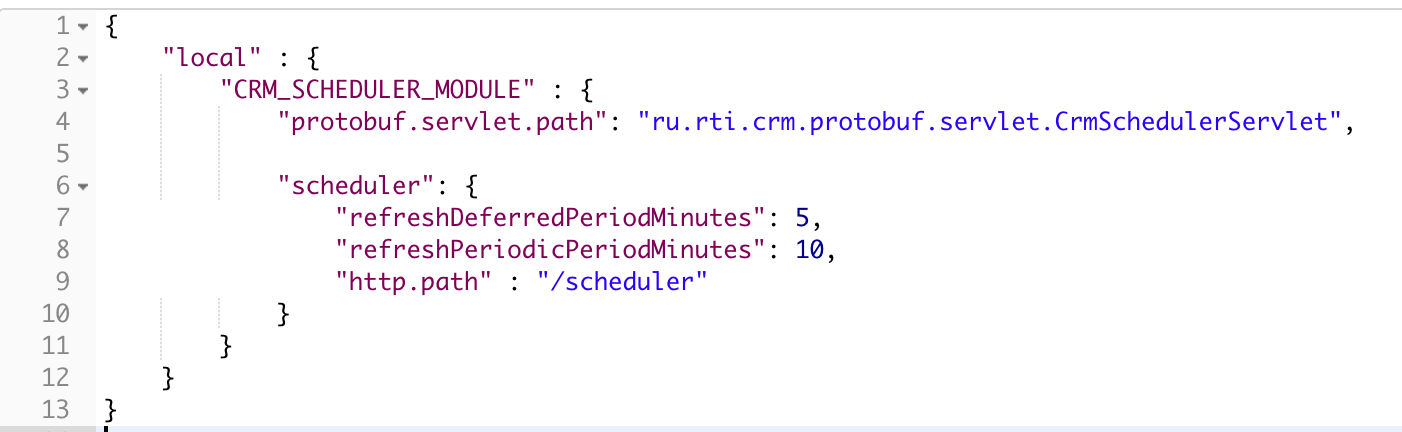


Рисунок 5.4 - Пример конфигурации модуля.

На рисунке видно, что за один раз можно послать конфигурацию сразу для нескольких профилей и модулей, соответственно задача данного метода состоит в том, что бы разбить данную конфигурацию на элементы и сохранить её в базу данных в том виде, что описывался ранее. Так как данный метод является самым нестандартным, приведем его упрощенный псевдокод в листинге 5.1.

Листинг 5.1.

|  |
| --- |
| uploadConfiguration(configuration) {  putToDb(«PROFILES», configuration.profile)  moduleNames = configuration.modules  for module in moduleNames {  moduleKey = join(configuration.profile, «\_», module)  putToDb(moduleKey, configuration.module.configuration)  }  } |

Получение конфигурации модуля по его ключу является необходимым методом, так как администратор должен иметь возможность посмотреть актуальную конфигурацию каждого модуля, а так же сравнить конфигурацию одного и того же модуля, развернутого на разных профилях. Так же данный метод используется другими модулями для актуализации их конфигурации.

Обновление конфигурации модуля представляет собой метод, принимающий на вход ключ модуля и, использую gRPC, вызывает у данного модуля метод refresh, который запрашивает конфигурацию у системы.

Получение списков всех профилей и всех модулей являются важными методами, так как с их помощью можно отслеживать текущие модуля в системе. При вызове данных методов возвращаются значения, лежащие в базе данных с ключами «PROFILES» и «MODULES» соответственно.

Хост и порт балансировщика являются необходимыми параметрами для работоспособности конфигуратора и остальных модулей системы. Как говорилось ранее, с его помощью происходит перенаправление запросов между кластерами, а так же идет маппинг ключа модуля и его ip адреса. То есть для того, что бы обновить конфигурацию конфигуратор не обязан знать, где развернут необходимый ему модуль.

Запись и получение этих из базы данных осуществляется соответствующими методами сервиса, при этом в хранилище для них выделен специальный ключ «APPLICATION»

Таким образом, были рассмотрены методы сервиса конфигуратора, а так же для наиболее важных и сложных из них был приведен псевдокод. Теперь, зная, какой интерфейс имеет конфигуратор, перейдем к проектированию его интерфейса.

5.2. Проектирование пользовательского интерфейса

При проектировании пользовательского интерфейса необходимо учитывать требования, предъявляемые к системе. Ниже приведена структуруа страниц проекта, обеспечивающую решение всех описанных требований (рисунок 5.5). Дизайн всех страниц будет иметь минималистический вид, что позволит без лишних усилий разобраться в интерфейсе.

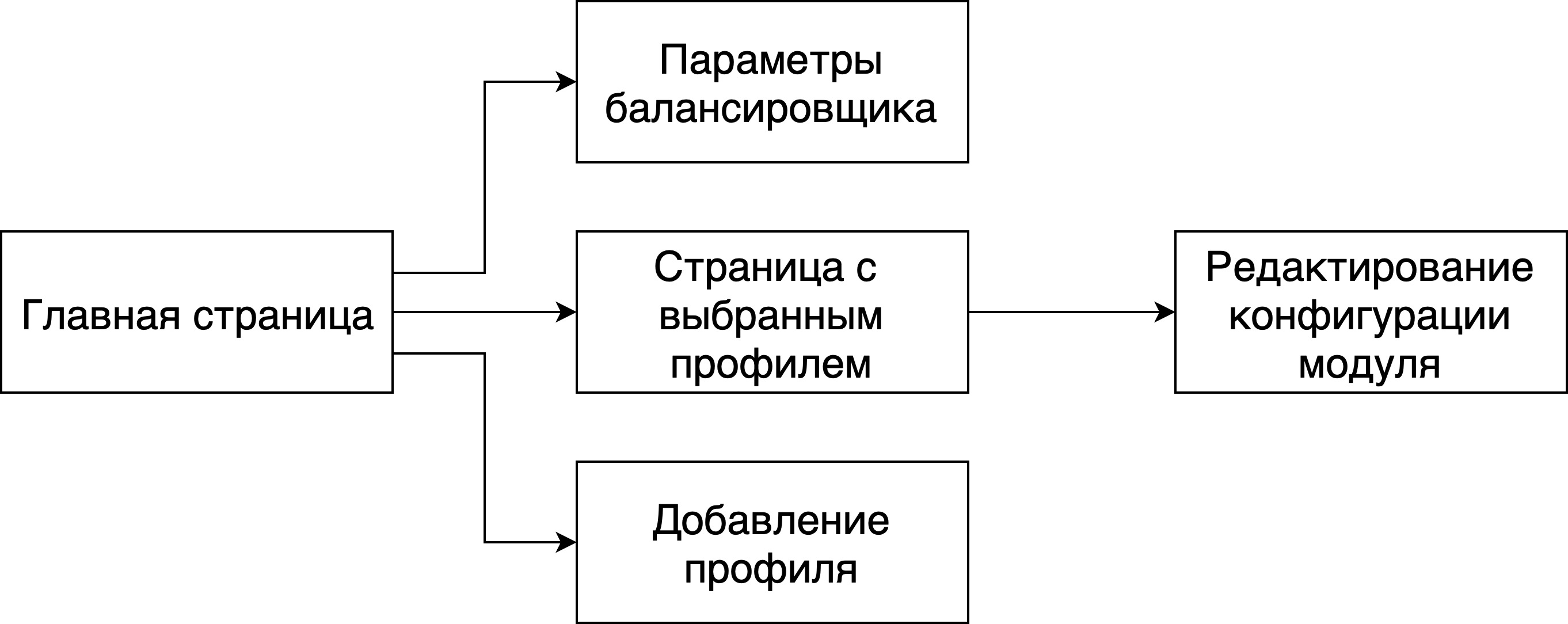
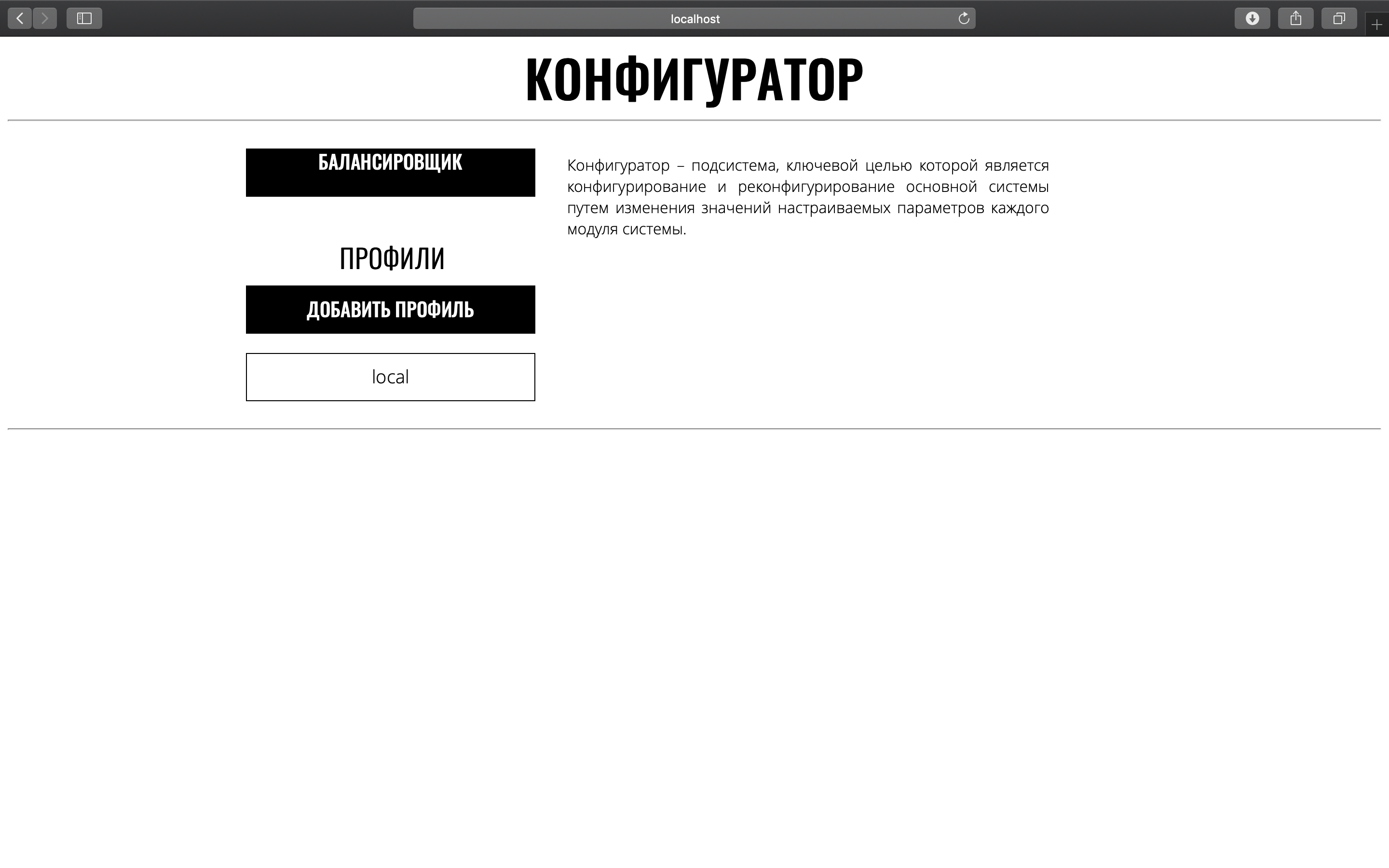


Рисунок 5.5 - Структура страниц проекта

Рассмотрим каждую из страниц, представленных выше более подробно.

На главной станице проекта (рисунок 5.6) приведено краткое описание приложения, кнопка для перехода на страницу с параметрами балансировщика, кнопка для добавления профиля, а также список доступный профилей. У всех страниц проекта в их заголовке с надписью «КОНФИГУРАТОР» есть ссылка на главную страницу проекта.

Рисунок 5.6 - Скриншот главной страницы интерфейса



Страница с параметрами балансировщика (рисунок 5.7) содержит в себе значения хоста и порта сервера, на котором работает балансировщик. При нажатии на кнопку «Изменить», станет доступна возможность изменения данных параметров (рисунок 5.8). При нажатии на кнопку «Сохранить», данные изменения сохранятся в системе.

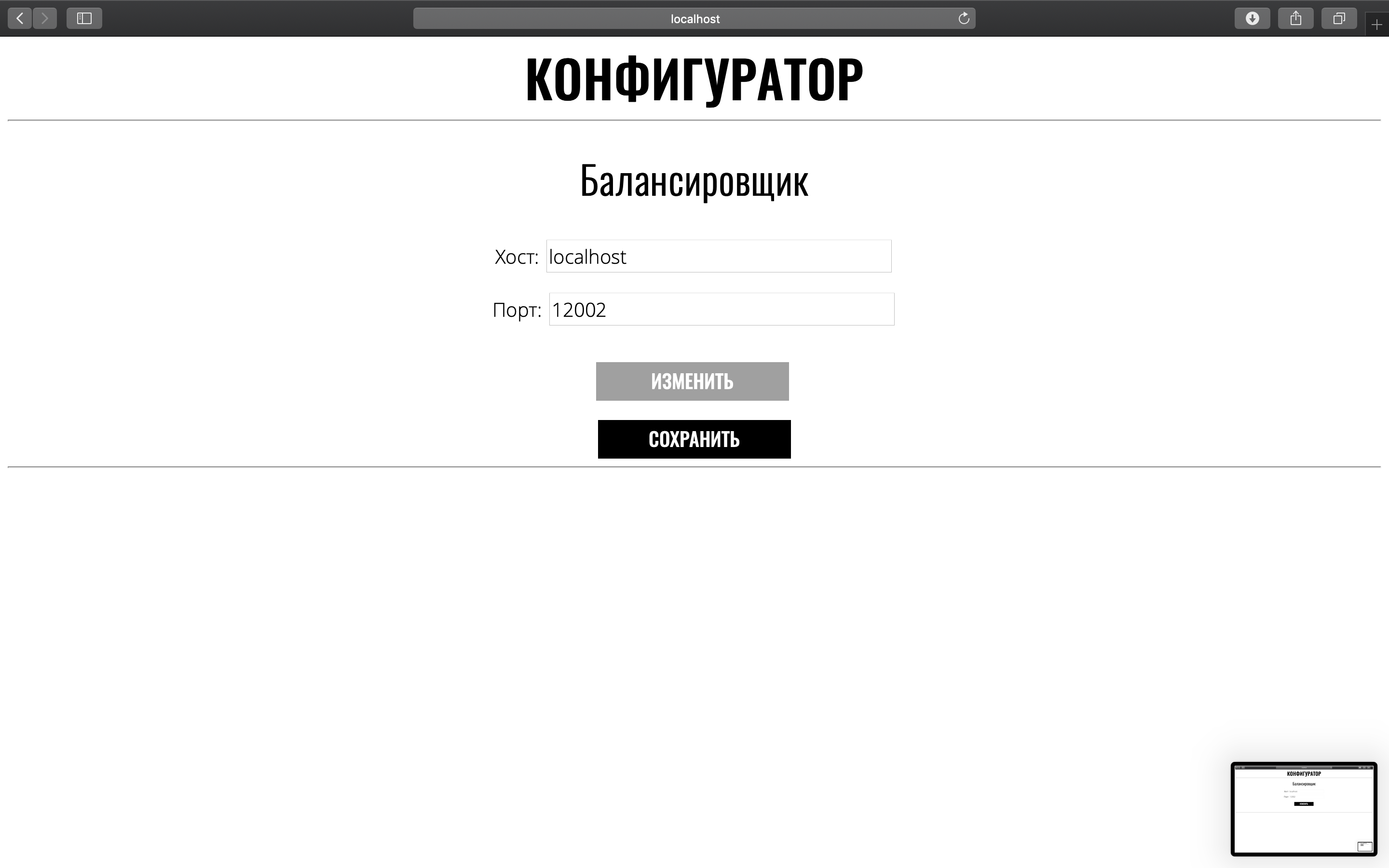
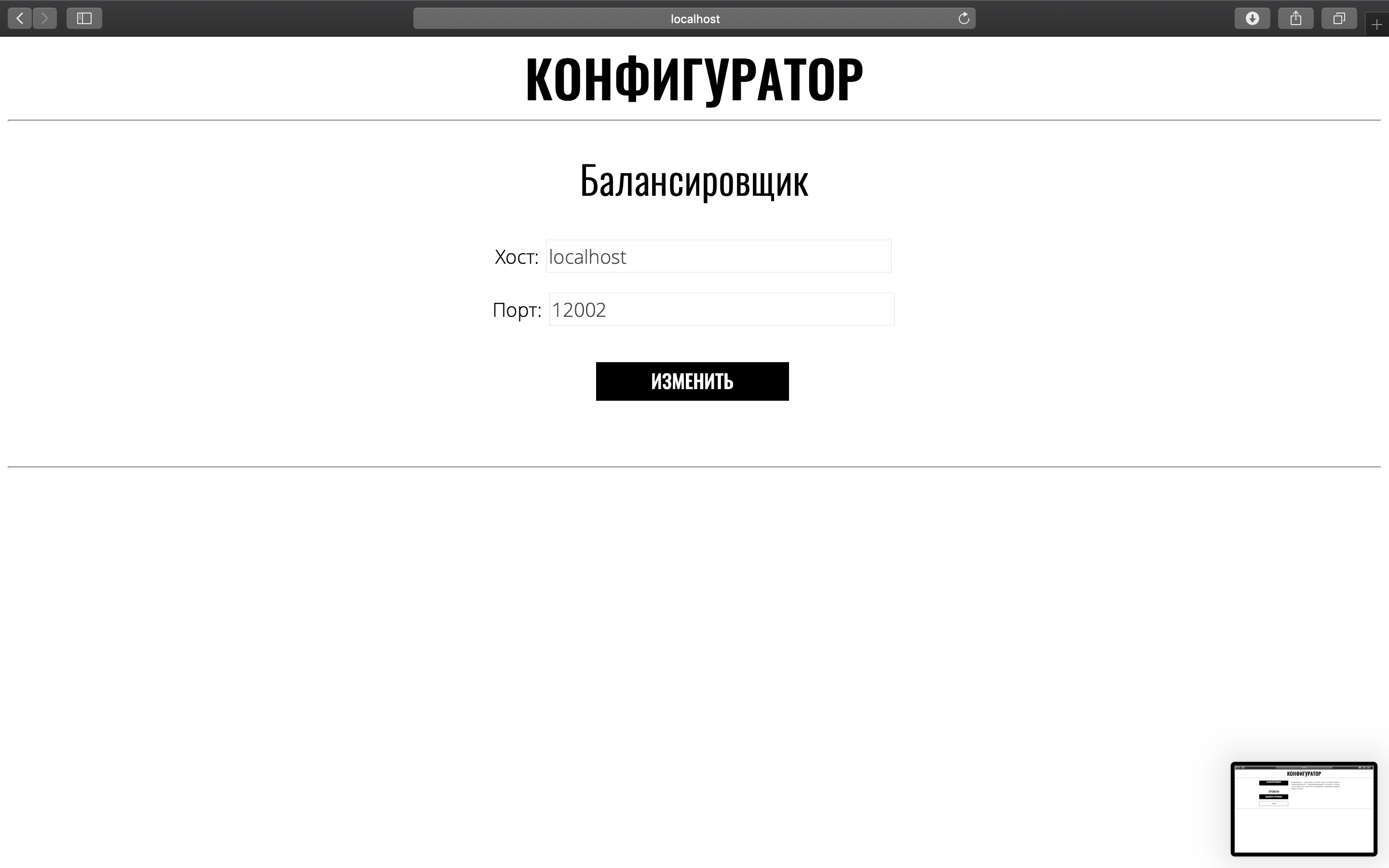


Рисунок 5.7 - Скриншот страницы с параметрами балансировщика

Рисунок 5.8 - Скриншот страницы с параметрами   
балансировщика в режиме редактирования

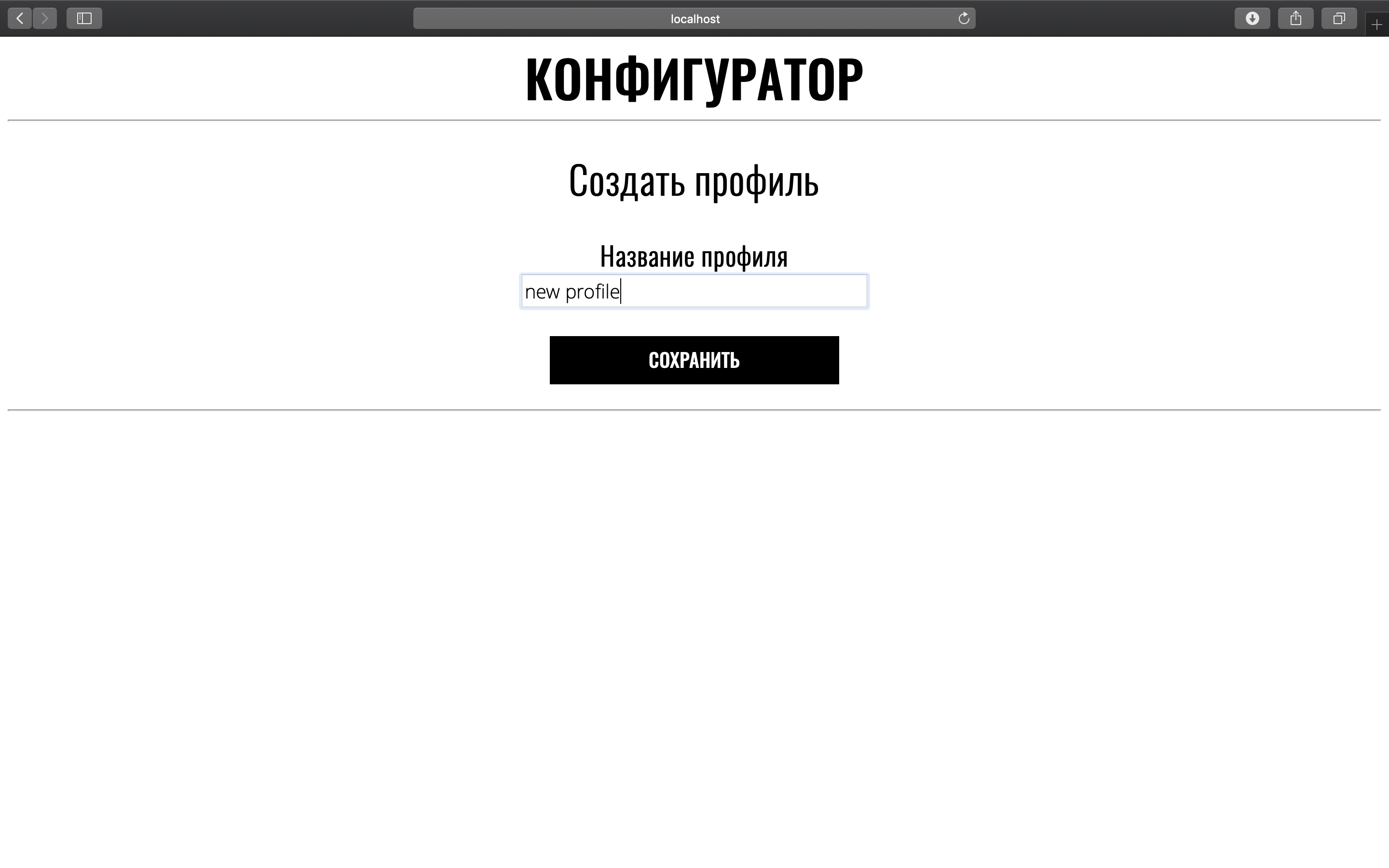
Страница с добавлением профиля (рисунок 5.9) позволяет задать имя профиля и сохранить его.

Рисунок 5.9 - Скриншот страницы создания профиля

После того, как был создан новый профиль, в него можно добавить новый модуль, перейдя на форму просмотра модулей и нажав на кнопку «Добавить модуль» перейдем на страницу создания модуля (рисунок 5.10), где указываются идентификатор модуля, а также его конфигурация в формате json.

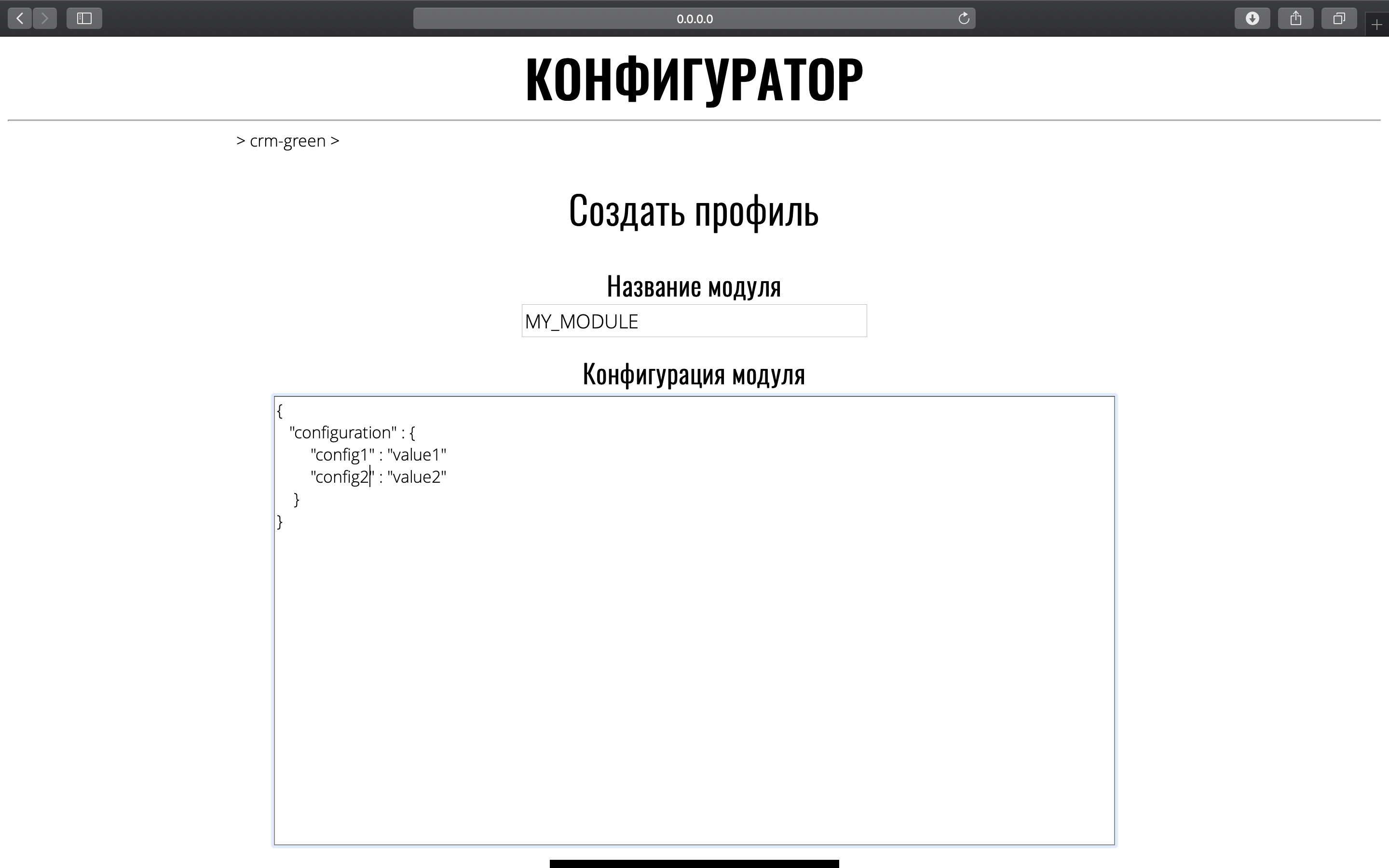
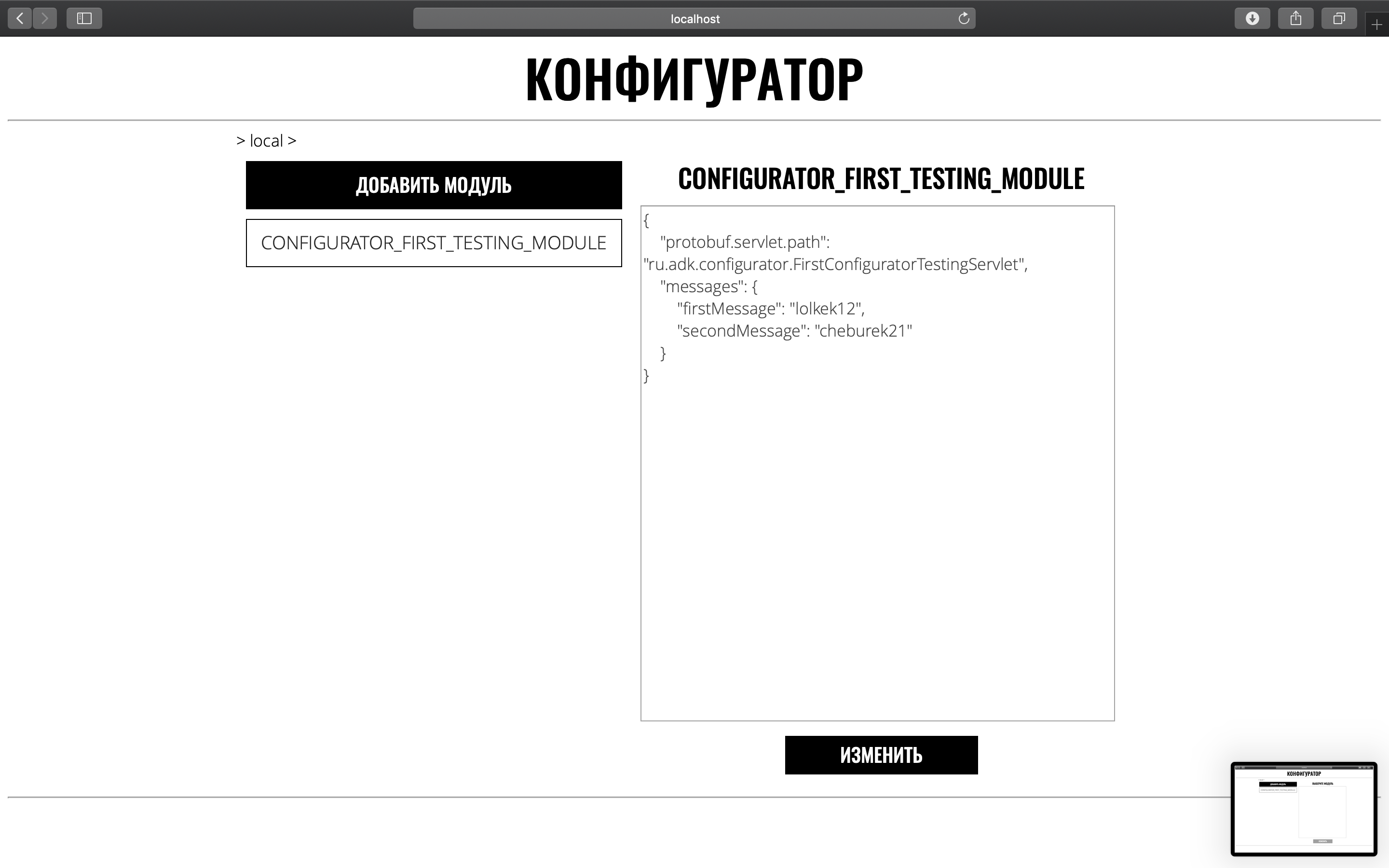


Рисунок 5.10 - Скриншот страницы создания модуля

После того, как модуль был создан, можно просмотреть все модули, входящие в данный профиль на странице просмотра модулей (рисунок 5.11).

На данной странице при нажатии на кнопку с названием модуля в окне справа отобразится его конфигурация.

Рисунок 5.11 - Скриншот страницы просмотра модулей

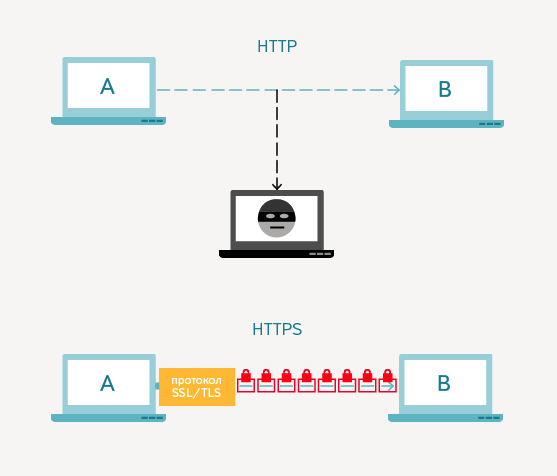


Таким образом, был рассмотрены все страницы пользовательского интерфейса для взаимодействия пользователя и системы, а так же приведена структура всех страниц проекта.

5.3. Обеспечение безопасности передачи данных и управления модулями

Для обеспечения безопасности передачи данных между администратором и конфигуратором, в первую очередь, стоит использовать не просто HTTP протокол, а его более современную версию HTTPS. Протокол HTTPS обеспечивает шифрование передаваемой информации, что позволяет обезопасить соединение между браузером и сервером. На рисунке 5.12 продемонстрирована схема передачи данных по HTTP и HTTPS. Как видно, при использовании первого протокола, злоумышленники могут просмотреть передуваемую информацию, а так же могут изменить ее.

Рисунок 5.12 - Схема передачи данных по HTTP и HTTPS



Дополнительным средством обеспечения безопасности может стать аутентификация каждого пользователя системы. Данный подход не позволит сторонним лицам использовать приложение «Конфигуратор» в своих целях.

6. Функциональное тестирование программной системы

Для тестирования работоспособности разработанной системы был разработан отдельный модуль, хранящий в своей конфигурации два параметра, представляющих собой информационные сообщения. Для того, что бы была возможность проверять значения данных параметров в режиме реального времени, в модуль был добавлен метод сервиса, возвращающий эти значения. На рисунке 6.1 представлена дефолтная конфигурация модуля.

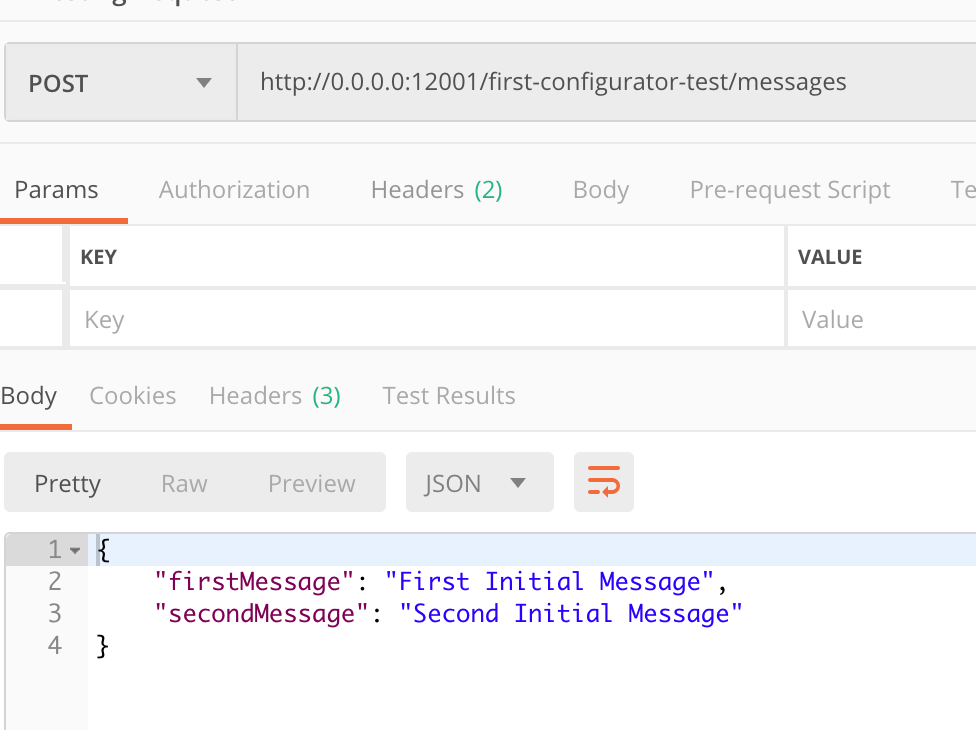


Рисунок 6.1 - Скриншот дефолтной конфигурации модуля

Теперь, с помощью графического интерфейса изменим текст сообщений на «Configurator first changed message» и «Configurator second changed message» и нажмем кнопку сохранить (рисунок 6.2) [11].



Рисунок 6.2 - Изменение конфигурации модуля

Используя сервис тестового модуля, проверим, изменились ли параметры его конфигурации на те, что были заданы выше. Для этого с помощью Postman выполним вызов метода сервиса (рисунок 6.3).

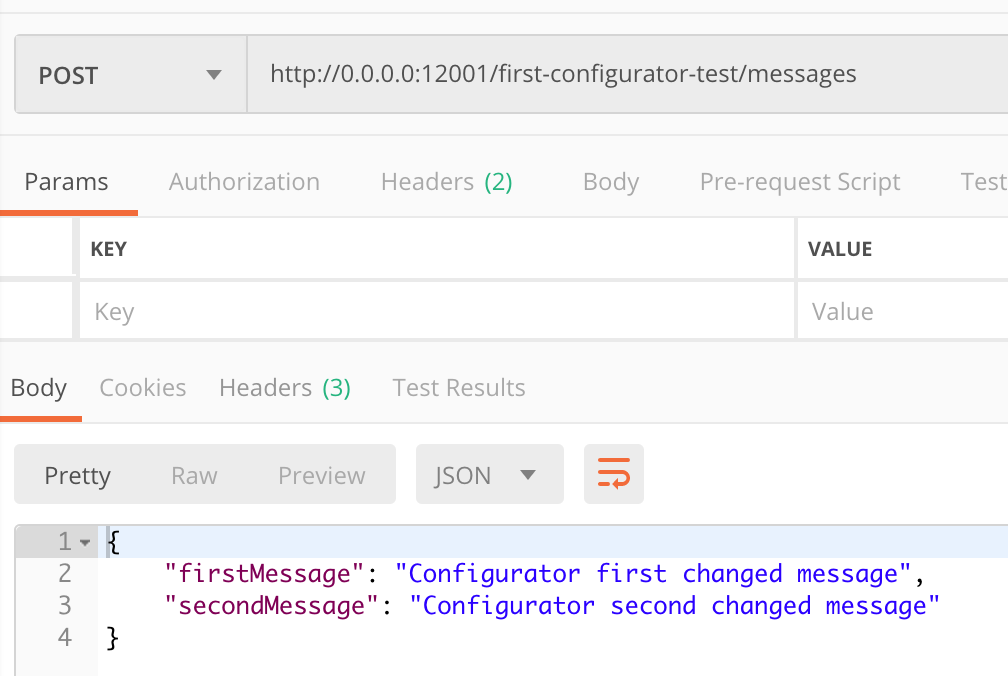


Рисунок 6.3 - Скриншот с обновленной конфигурацией модуля

Как видно из рисунка выше, параметры конфигурации изменились, что свидетельствует о корректной работе конфигуратора.

Заключение

В ходе данной работы были рассмотрены две основные архитектуры проектирования программного обеспечения: монолитная и мультимодульная. В ходе их сравнительного анализа было показано, что мультимодульная система имеет ряд преимуществ над системой, спроектированной по классической монолитной архитектуре.

Затем было рассмотрено устройство типичного модуля системы, показан способ активации/деактивации его сервисов и методов, а так же общее значение конфигурации в процессе управления работы модуля.

Итогом всей проделанной работы стало приложение «Конфигуратор», позволяющее без перезагрузки приложения изменять значения параметров конфигурации. Была описана схема его взаимодействия с модулями, описаны методы его сервиса, а так же спроектирован интерфейс для удобной работы с приложением.

Список литературы

1. С. В. Шибанов, А. А. Мезенков, О. А. Шевченко, А. С. Илюшкин - Принципы организации и функционирования активных пакетов для обмена информацией и конфигурирования распределенных приложений // Известия высших учебных заведений. Поволжский регион. Технические науки. - 2013. - № 1 (25). - С. 5-18.

2. Л. А. Растригин. «Адаптация сложных систем» // Рига: Зинатне, 1981. — 375 с.

3. Баканов А. Б., Дрождин В. В., Зинченко Р. Е., Кузнецов Р. Н. - Методы адаптации и поколения развития программного обеспечения // Известия ПГПУ им. В. Г. Белинского. 2009. № 13 (17). С. 66-69.

4. Х. Груневеген. Конфигурирование или адаптация? // Открытые системы. – 2008. – № 6 – С. 37-41

5. Создание модульных структур программного продукта – Студопедия [Электронный источник] http://studopedia.ru // URL: http://studopedia.ru/ 18\_64582 \_sozdanie-modulnih-struktur-programmnogo-produkta.html

6. Фаулер Мартин. Архитектура корпоративных программных приложений. Пер. с англ. – M. : Издательский дом «Вильямс», 2006. – 544 с.

7. Создание архитектуры программы или как проектировать табуретку – Хабрахабр [Электронный источник] https://habrahabr.ru // URL: https://habrahabr.ru/post/276593

8. Модульное программирование | Технологии программирования | Языки программирования Pascal и Delphi [Электронный источник] http://www.maksakov-sa.ru // URL: http://www.maksakov-sa.ru/TehProgram/ ModulProgram/index.html

9. Нечаев В.В. Конфигурационное моделирование: часть 1. Теоретические аспекты: Учебное пособие/ Государственное образовательное учреждение высшего профессионального образование «Московский государственный институт радиотехники, электроники и автоматики (технический университет)». – М.: 2007. – 92 с.

10. Нечаев В.В. Концептуальное метамоделирование структур. // Международная Академия информатизации; Отделение «Математическое и компьютерное моделирование». – М.: Международное изд. «Информация», 1997. – 52 с.; илл. 2; табл. 11; библ. 22 назв.

11. Большой словарь иностранных слов: - М.: ЮНВЕС, 1998. – 784 с.

12. Courtois, P. June 1985. On Time and Space Decomposition of Complex Structures. Communications of the ACM vol.28(6), p.596.

13. Нечаев В.В. Классификация задач синтеза структур в системах эволюционного моделирования. В кн.: Перспективы развития вычислительных систем (Применение идей эволюции и адаптации). II всесоюзный семинар (Рига, май 1985): РПИ, 1985. – с. 133 – 135.

14. Сайт по управлению услугами – Сервис менеджмент [Электронный источник] http://www.smlogic.ru // URL: http://www.smlogic.ru /g-it-s/itsm/protsess-upravleniya-konfiguratsiyami-configurat

15. Д. Лапыгин, А. Новичков. Конфигурационное управление проектами разработки программного обеспечения (2004). [Электронный источник] http://citforum.ru // URL: http://citforum.ru/SE/quality/configuration\_management