РЕФЕРАТ

Магистерская диссертация содержит 42 страницы, 15 рисунков. Список использованных источников содержит 15 позиций.

НЕЙРОМОРФНЫЕ СИСТЕМЫ, МИКРОСЕРВИСЫ, ДОКУМЕНТНЫЕ БАЗЫ ДАННЫХ, ОДНОСТРАНИЧНЫЕ ПРИЛОЖЕНИЯ, ПРОГРАММНЫЕ ИНТЕРФЕЙСЫ, ПОЛЬЗОВАТЕЛЬСКИЕ ИНТЕРФЕЙСЫ

Магистерская диссертация посвящена анализу работы нейроморфных систем, аспектов разработки современных прикладных программных комплексов, а также реализации программного и пользовательского инерфейсов для системы нейроморфного моделирования.

Были рассмотрены аспекты работы с документной базой данных MongoDB, разработки Rest API интерфейсов на языке программирования Kotlin с использованием библиотеки Ktor, разработки пользовательских интерфейсов на языке программирования TypeScript с использованием библиотек React и Redux для управления состоянием клиенсткого приложения и хранения данных в нем. С использованием этих технологий, был реализован программный комплекс для запуска сценариев распознавания шаблонов системой нейроморфного моделирования. Выстроено взаимодействие данной системы с серверной частью программного комплекса.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Искусственные нейронные сети используются во многих областях современной жизни и позволяют решать актуальные, важные и практически значимые задачи, которые зачастую не поддаются решению с помощью классических подходов. Одним из динамично развивающихся направлений в настоящее время является разработка нейроморфных системы на базе мемристоров [1]. Для математического моделирования работы таких систем были разработаны программные средства [2], которые представляют собой вычислительный модуль на C++. Однако, для конечных пользователей-исследователей необходим удобный интерфейс, а для интеграции расчетного модуля в многомасштабные сценарии расчетов [3] необходим программный интерфейс (API). Текущая работа посвящена разработке пользовательского и программного интерфейсов для подобной системы.

Целью выпускной квалификационной работы является анализ деталей реализации программного комплекса, предусматривающего все возможные аспекты взаимодействия с системой нейроморфного моделирования, обладающее гибкостью и расширяемостью для минимизации затрат на поддержку развивающегося функционала модуля.

В качестве решения было предложено разработать веб-приложение, состоящее из трех частей: база данных, серверного приложения, пользовательского интерфейса. Были учтены лучшие практики современной разработки программного обеспечения: использование документных баз данных[4], микросервисной архитектуры[5], современных библиотек разработки веб интерфейсов[11], а также контейнеризация приложения для стабильного развертывания на сервере [12].

Для этого потребовалось выполнить следующие задачи:

1. Изучить принципы работы нейроморфных систем в целом и решение, разработанное на кафедре в частности.

2. Реализовать серверное приложение с Rest API интерфейсом на языке программирования Kotlin с использованием библиотеки Ktor. Предусмотреть использование OpenAPI документации и интерфейса Swagger. Наладить взаимодействие с базой данных MongoDB. Произвести интреграцию с нейроморфной системой.
3. Подготовить дизайн графического интерфейса, реализовать его с помощью библиотеки React. Реализовать управление данными в пользовательском интерфейсе с помощью библиотеки Redux. Выстроить взаимодействие с Rest API интерфейсом серверного приложения, реализовать методы для HTTP запросов.

В итоге были разработаны все части программного комплекса с использованием перечисленных технологий.

В разделе 1.1 описаны основы работы нейроморфных систем, история возникновения и развития данной области исследований, а также проанализированы существующие решения. В разделе 1.2 описаны требования к разрабатываемому программному комплексу, а также приведен обзор технологий, с помощью которых можно реализовать программное обеспечение, удовлетворяющее требованиям. Раздел 2 полностью посвящен аспектам реализации практичекой реализации, начиная от описания архитектуры и ее обоснования (раздел 2.1), заканчивая детальным описанием каждой части комплекса(разделы 2.2, 2.3). В разделе 2.4. описан и обоснован выбранный вариант взаимодействия серверного и клиентского приложений. В заключении подведены итоги работы и сделаны основные выводы.

ОСНОВНАЯ ЧАСТЬ

1.ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Системы нейроморфного моделирования

В актуальности темы нейросетевых технологий не приходится сомневаться, они постепенно входят в обыденную жизнь и уже даже самые далекие от мира информационных технологий люди зачастую знают о подобных вещах. Однако, куда больший интерес представляют так называемые нейроморфные сети. Также, как и искуственные нейронные сети, нейроморфные системы проектируются, повторяя структуру человеческого мозга, нейроморфные чипы моделируют работу нейронов и их отростков — аксонов и дендритов — отвечающих за передачу и восприятие данных. Связи между нейронами образуются за счет синапсов — специальных контактов, по которым транслируются электрические сигналы.

Потенциал разработки нейроморфных устройств заключается в ускорении обучения сверточных нейронных сетей для распознавания изображений. У систем искусственного интеллекта такого типа нет нобходимости использовать хранилища больших размеров с тренировочными данными по сети, поскольку сами искусственные нейроны хранят нужную информацию, благодаря использованию мемристоров. Мемри́стор (от англ. memory — память, и англ. resistor — электрическое сопротивление) — пассивный элемент в микроэлектронике, способный изменять своё сопротивление в зависимости от протекавшего через него заряда (интеграла тока за время работы). Может быть описан как двухполюсник с нелинейной вольт-амперной характеристикой, обладающий гистерезисом и являющийся элементарной ячейкой долгосрочной энергонезависимой памяти . Объединение мемристоров в матрицу (кроссбар) позволяет выполнять быстрое аналоговое произведение матрицы на вектор. За счет определенного сходства мемристивных элементов с биологическим синапсом перспективным представляется их использование для аналоговой реализации самообучающихся импульсных нейронных сетей[2].

В итоге технология позволяет запускать алгоритмы машинного обучения локально. Согласно этому, очевидно главное преимущество таких систем – нет необходимость использовать мощные кластере, нейросети такого типа можно запускать даже на своей машине, более того, есть все основания полагать, что нейроморфные чипы в будущем будут примененяться в мобильных устройствах, IoT-гаджетах, а также центрах обработки данных[13].

1.1.1. История возникновения вопроса

Впервые, исследования в области нейроморфных систем попытались релизовать на практике в 60-х годах двадцатого века. Тогда одним из будущих изобретателей микропроцессора Тэдом Хоффом вместе с профессором из Стэнфорда Бернардом Уидроу была разработана одноуровневая нейросеть с использованием мемристоров.

Считается, что эта разработка положила начало нейроморфной инженерии. Затем, спустя несколько десятилетий, в 80-х годах инженер Карвер Мид предложил использовать транзисторы в качестве аналоговых компонентов, а не цифровых переключателей. Позже, в 90-х годах командой во главе с Мидом был разработан искусственный синапс, который способен продолжительное время хранить некоторую информацию, а также нейроморфный процессор на основе транзисторов с плавающим затвором. В то же время президент США Джордж Буш-старший заявил о старте «Десятилетия мозга» и предложил выделять финансовую помощь под программы, целью которых является изучение этого органа. Все эти факторы в совокупности дали толчок к старту бурного прогресса в области нейроинформатики и вычислительной нейробиологии, а также созданию инфраструктуры для дальнейшего изучения темы.

В последнее десятилетее представления человечества о структуре и принципах устройства мозга достигли новых высот. Начиная с 2013 года, Швейцария занимается активным продвижением проекта Human Brain Project

(HBP). Также в США была запущена программа BRAIN Initiative. Эти инициативы оказали серьезное влияние на сферу систем искусственного интеллекта и привели к появлению новых нейроморфных технологий.

1.1.2. Существующие решения

На текущий момент можно отметить следующие решения в данной области:

- В 2008 году инженеры компании IBM при поддержке DARPA приняли участие в программе SyNAPSE, в рамках нее велось проетирование электронно вычислительных машин, чьи архитектуры, отличаются от фон Неймановских. В течении трех лет данной компании удалось реализовать ядро с 256 искусственными нейронами (у каждого из них были 256 синапсов). Затем, спустя еще три года компанией был представлен процессор TrueNorth, который включает в себя 4096 таких ядер — то есть количесто нейров превышает миллион. Данный процессор уже нашел применение в задачах распознавания жестов и речи.
- В 2017 году корпорацией Intel был представлен чип Loihi. Он базируется на 128 нейроморфных ядрах, каждое из которых симулирует 1024 нейрона. Для данного проессора существует библиотека на языке python, с помощью которой можно вести работу с ним. Первые экземпляры этих устройств уже используются в центра по обработке данных нескольких ведущих университетов для проведения тестов на реальных задачах.
- В 2017 году инженеры манчестерского вуза представили архитектуру SpiNNaker, которая включает в себя миллион ядер, способных эмулировать работу ста миллионов нейронов. Для программирования данного компьютера был разработан специальный язык - PyNN. На сегодняшний день машина

используется для симуляции процессов, происходящих в мышином мозге.

- Система нейроморфного моделирования, разработанная на кафедре. Согласно статье [2], архитектура данной системы базируется на однослойной нейроморфной сети состоящей из 1T1R мемристорного кроссбара и нейронов. Мемристоры по сути выступают в качестве синапсов. Есть разные математические модели мемристоров, которые можно подстраивать под экспериментальные данные. Используются математические модели, описывающие мемристоры на основе оксида титана и оксида гафния.

Как было отмечено выше, область нейроморфных систем пользуется достаточно большой популярностью, зачастую такие системы используются в прикладных компьютерных системах[3], которыми могут пользоваться не только разработчики этих систем, но и сторонние пользователи, проводящие с помощью них вычислительные эксперименты. Из этого возникает потребность в наличии интуитивно понятного пользовательсекого интерфейса, работая с которым пользователь мог бы сосредоточиться только на значениях входных параметров системы, не задумываясь о том, как она устроена технически. Однако на текщий момент количество существующих решений в данной области достаточно невелико. Среди основных решений можно рассмотреть следующие:

- NNTool - предоставляет расширение для matlab, которое позволяет задавать конфигурации нейронных сетей.
- ENNU (ENNUI Elegant Neural Network User Interface) - предоставляет Web интерфейс для построения моделей нейросетей с заданным количеством слоем и их конфигурацией.

1.2. Разработка программного комплекса

1.2.1. Требования к разрабатываемому программному обеспечению

Несложно заметить, что все существующие решения заточены под определенные процессоры, что добавляет требование к моделированию – оно должно производится с использованием определенных процессоров, что не очень удобно. К тому же, только у одного из перечисленных вариантов существует доступное API, интеграция же с остальными даже при условии, что все остальные нюаннсы удастстя разрешить, вызывает трудности. В связи с этим было принято решение использовать модуль, разработанный на кафедре.

Анализ существующих решений для обеспечения системы нейроморфного моделирования интуитивно понятным интерфейсом дал понимае, что подходящего решения, удовлетворяющего требованиям(распознавание шаблонов нейросетями с использованием математических моделей мемристоров) в свободном доступе не существует, в связи с этим было принято решение разработать свой программный комплекс.

Данный комплекс должен поддерживать два типа интерфейса:

1. Программный интерфейс для будущей интеграции в платформу многомасштабного моделирования [3], а также для взаимодействия с графическим интерфейсом.
2. Пользовательский интерфейс для работы конечных пользователей, не знакомых с способом работы системы нейроморфного моделирования.

Работа с сервисом будет вестись, согласно пользовательским сценариям, описанным на рис.1.1.

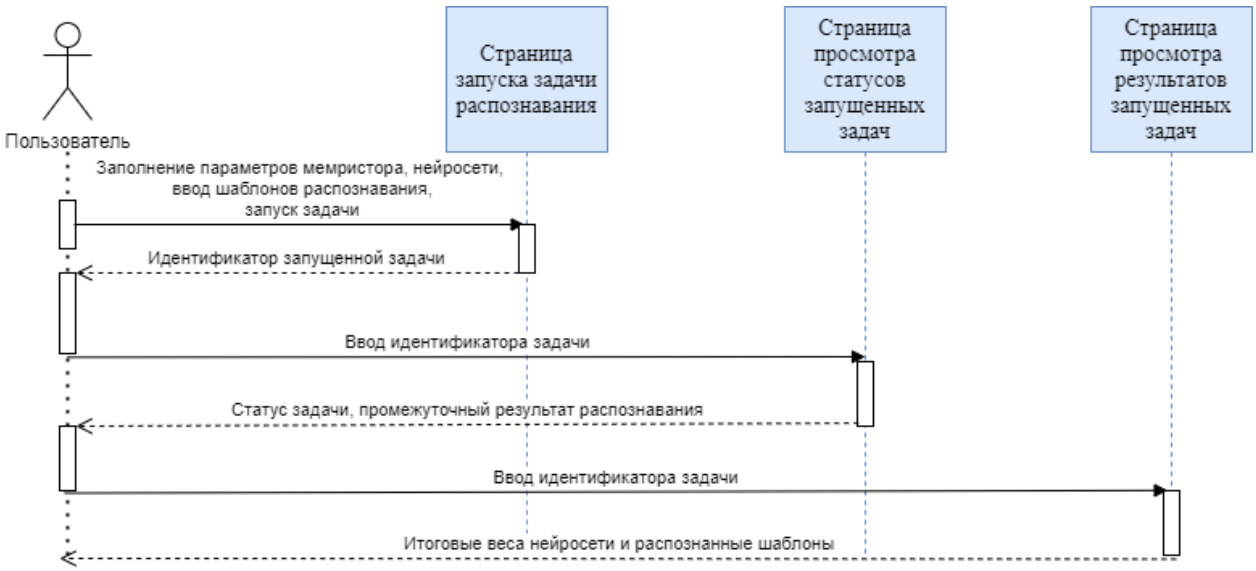


Рис.1.1.Диаграмма прецедентов

При реализации программного обеспечения для удовлетворения данным сценариям, необходимо учесть следующие функциональные требования:

1. Корректная обработка входных данных, валидация всех значений.

2. Отсутствие “магических” чисел, то есть возможность для пользователя вводить не численные характеристики, а название константы, по которой будет понятно ее значение.

3. В качестве шаблонов для распознавания нейросети дать возможность пользователю интерактивно рисовать пиксельное черно-белое изображение.

4. Возможность пользователю вводить, как параметры нейросети, так и параметры математической модели мемристора.

5. Возможность просматривать промежуточные результаты работы сети в течении ее работы, а также итоговые результаты в наглядном виде (представляя полученные сетью веса градациями серого в пиксельном изображении).

6. Возможность сохранения промежуточных и итоговых результатов работы модуля, импорт и экспорт входных параметров.

7. Воспроизводимость экспериментов. При вводе одних и тех же данных, результат не должен меняться.

Для наиболее быстрой и надежной реализации решения, удовлетворяющего всем перечисленным требованиями, был предложен

следующий спектр технологий и принципов, базируясь на которых должен быть реализован программный комплекс:

- Микросервисная архитектура для серверного приложения - набор небольших, слабо связанных и легко изменяемых модулей — микросервисов[6].
- Rest(REpresentational State Transfer) API(application programming interface) в качестве интерфейса взаимодействия клиентского и серверного приложений. Расширение протокола HTTP, предполагает рассматривать приложения с точки зрения управления ресурсами[13]. В данном случае в качестве ресурсов выступают задачи распознавания шаблонов
- Swagger интерфейс - возможность выполнять HTTP запросы по конечным точкам Rest API без дополнительного программного обеспечения через браузер.
- Подход SPA(single page application) для клиентского приложения. Позволяет строить более гибкие с точки зрения загрузки web приложения[11].
- Использование нереляционной базы данных, поскольку такой тип баз данных упрощает развертывание на сервере, снижает затраты на написание миграций(скрптов по установке схемы базы данных), а также упрощает конвертацию данных из внутреннего формата серверного приложения в формат базы данных[5].

1.2.2. Хранение данных

С учетом описанных выше причин, для хранения данных был выбран нереляционный тип базы данных. Среди данного типа баз данных также можно выделить несколько групп:

- Документно ориентированные базы данных. Подобная база данных оперирует документами, которые могут быть представлены в виде

XML, JSON, BSON и т.д. Эти документы представляют собой самоописывающиеся иерархические древовидные структуры данных, которые могут состоять из словарей, коллекций и скалярных значений. Хранящиеся документы похожи друг на друга, но не обязательно должны быть точно такими же[5]. Примеры реализаций – ElasticSeasrch, MongoDB.

- Колоночные базы данных. Подобный тип базы данных позволяет хранить данные с ключами, сопоставленными значениям, и значениями, сгруппированными в несколько семейств столбцов, причем каждое семейство столбцов представляет собой словарь[5]. Пример реализации – Cassandra. В данном случае использование подобной базы данных в целом может подойти, однако ее функционал избыточен для нужд разрабатываемого микросервиса.
- Базы данных типа “ключ-значение”. Подобная тип базы данных представляет собой обычную хэш-таблицу, которая в основном используется, когда весь доступ к базе данных осуществляется через первичный ключ[5], к примеру, для хранения кэша в приложении. Примеры реализаций – Hazelcast, Redis. В данном случае использование подобной базы данных не подходит, поскольку не все сценарии использования подразумевают получение данных по ключу.
- Графовые базы данных - подразумевает хранение данных в виде графов. В таких системах хранятся вершины и ребра. У ребра может быть неограниченное количество атрибутов(аналог таблицы в базе данных), оно может иметь ребра(но не обязательно). Также ребро может иметь характеристуку отношения, к примеру – вес. Подобные базы данных могут иметь встроенне реализации нужных алгоритмов, к примеру, обход в ширину или глубину. Пример реализации базы данных такого типа – neo4j. В данном случае

использование подобной базы данных не подходит, поскольку среди задач нет иерархической структуры, подобной графами.

В итоге, был выбран документный тип базы данных и его реализация - MongoDB за счет простоты конфигурации, легкого масштабирования и взаимодействия с клиентским приложением. База данных будет отвечать за хранение данных по проведенным экспериментам, а также справочников.

MongoDB — документоориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++. Используется в веб-разработке, в частности, в рамках JavaScript-ориентированного стека MEAN.[14]

1.2.3. Серверное приложение

С учетом того, что предлагается реализовывать приложение согласно микросервисной архитектуре с использованием Rest API интерфейса, можно выделить следующие наиболее популярные языки программирования и библиотеки, которые позволяют это сделать:

- Язык программирования python с использованием библиотеки Flask/Django. Позволяет строить Rest API приложения, а также графиеские интерфейсы. Из плюсов можно отметить простоту синтаксиса языка python и соответствующих фрайевморков, большое количество документации. Из минусов же – динамическая типизация(понижает стабильность работы сервиса), использование технологии server side rendering для отрисовки графических интерфейсов(данный тип интерфейсов устраревает и не обеспечивает требуемою гибкость и скорость работы приложения).
- Spring Boot**.** Один из наиболее популярных фраемворков при разработке на языке программирования Java/Kotlin. Из плюсов можно отметить то, что в данном фреймворке реализовано почти

все, что может понадобится при разработке серверного приложения(работа с базой данных, rest api контроллеры и т.д.). Из минусов стоит отметить избыточность: в приложении запускаются те модули, что могут даже не потребоваться, а также итоговый размер приложения после сборки.

- ASP.NET Core – фреймворк для разработки web приложений на языке c#. Также, как и django, позволяет строить графические интерфейсы с использованием библиотеки Blazor. Из плюсов данного фреймворка можно отметить готовую реализацию некоторых необхимых в серверном приложении вещей, а также строгую типизацию. Из минусов – не самый гибкий синтаксис языка, малое количество библиотек, совместимых с данной платформой.
- Язык программирования kotlin с использованием библиотеки Ktor. Язык Kotlin [5] сочетает в себе преимущества Java, но в тоже время обладает большим количеством современных синтаксических конструкций, облегчающих поддержку и сопровождение кода. Библиотека Ktor в свою очередь позволяет быстро и стандартным способом реализовать часть инфраструктурных функций приложения, которая отвечает за взаимодействие по HTTP протоколу.

В итоге, опираясь на проведенный анализ технологий, был выбран язык программирования kotlin. Во-первых, за счет того, что он компилируемый и реализует статическую типизацию, что позволяет разрабатывать более стабильные приложения. А во-вторых, большая часть модулей в системе уже написана на этом языке, а значит модуль, разрабатываемый в рамках диссертационной работы будет куда проще встраивать в общую систему. В качестве фреймворка был выбран Ktor, он предоставляет очень простые и функциональные методы для создание конечных точек API.

1.2.4. Графический интерфейс

Среди основных библиотек разработки одностраничных приложений, можо выделить следующие:

- React. Наиболее популярная библиотека. Позволяет представить графический интерфейс в виде составных частей – компонент, которые впоследствии можно переиспользовать без каких либо затрат.
- Angular. Позиционируется, как фреймворк разработки веб приложений для программистов, имеющих опыт в работе с серверными приложениями. Из минусов данного фреймворка можно отметить чрезмерную сложность и высокий порог входа по сравнению с React.

Для клиентского приложения был выбран React, поскольку он является самым простым фреймворком и в тоже время самым удобным, также для него существует огромное количество библиотек, которые можно использовать, вместо того, чтобы реализовывать что-то самому, это позволяет строить приложения более быстро и увеличивает их надежность.

Среди двух возможных языков программирования: JavaScript и TypeScript был выбран последний, поскольку он позволяет писать более надежные приложения, за счет того, что о многих ошибках становится известно еще на стадии компиляции, до непосредственного запуска. React же позволяет строить приложение, разбивая его на составные части – компоненты, которые в последствии удобно переиспользовать.

TypeScript — язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript[9]. TypeScript является обратно совместимым с JavaScript и компилируется в последний. Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js. Код

экспериментального компилятора, транслирующего TypeScript в JavaScript, распространяется под лицензией Apache.

Для отрисовки шаблонов распознавания использовался Canvas API .

Redux - храннеие данных в приложении

На клиентское приложение возложена обязанность предоставить пользователя максимально интуитивно понятный интерфейс для управления нейроморфной системы.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Архитектура комплекса

Реализуемый программный комплекс, согласно задумке должен стать одним из сервисов в существующем решении платформы нейроморфного моделирования. Оно представляет собой некое веб приложение, в котором пользователя может выбрать необходимый ему модуль для выполнения расчетов/моделирования и перейти на нужную страницу, иными словами интеграционная оболочка представляет собой каталог существующих сервисов. Место реализуемого сервисе в общей архитектуре можно увидеть на рис. 2.1.

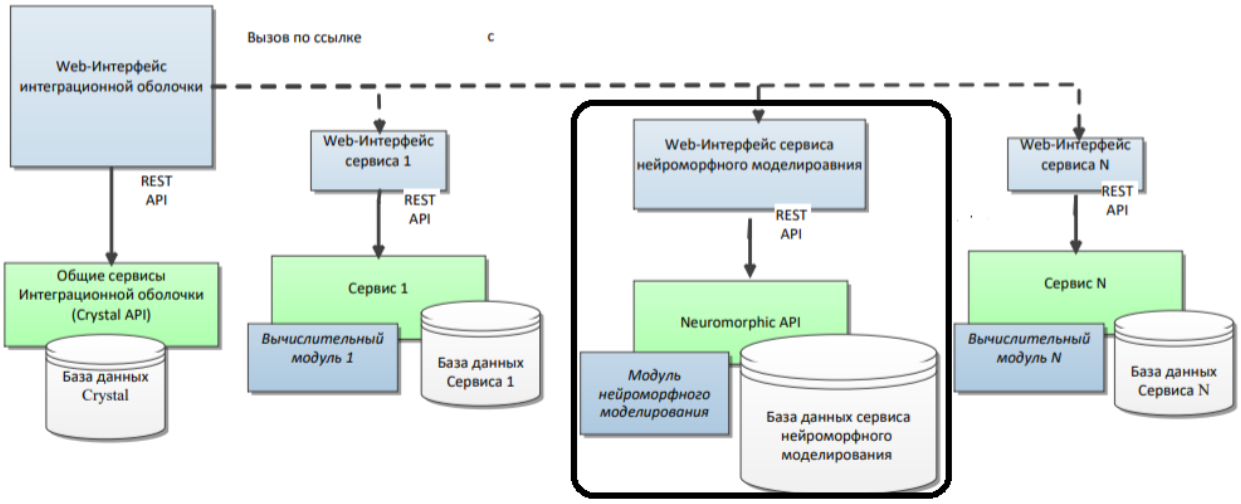


Рис. 2.1. Архитектура платформы многомасштабного моделирования

Основные компоненты реализуемого решения:

- External Service - существующий сервис для нейросетевого моделирования. В рамках разработки рассматривается, как черный ящик.
- Internal Service – реализуемый программный комплекс.
- External service (Neuro Modelling Module) – внешний сервис, с помощью которого можно выполянть нейромоморфное моделирование. Представляет собой исполняемый файл

скомпилированный из кода, написанного на языке программирования c++.

- Service API – серверное приложение, является “ядром” программного комплекса. Взаимодействует с внешним сервисов, сохраняет нужные данные в базу данных, реализует REST API, по которому User Interface получает данные. Реализовано на языке kotlin с ипользованием библиотеки ktor для построения REST API.
- Database – база данных MongoDB, в которой хранятся данные серверного приложения.
- User interface – пользовательский web интерфейс, взаимодействующий с серверным приложением по протоколу HTTP(Rest API). Реализован на языке программирования TypeScript с ипользованием библиотеки React.

В данном разделе приведена общая схема взаимодействий компонентов, с которой можно ознакомится на рис. 2.2., подробности деталей их реализации и протоколов, по которым они взаимодействуют приведены далее.

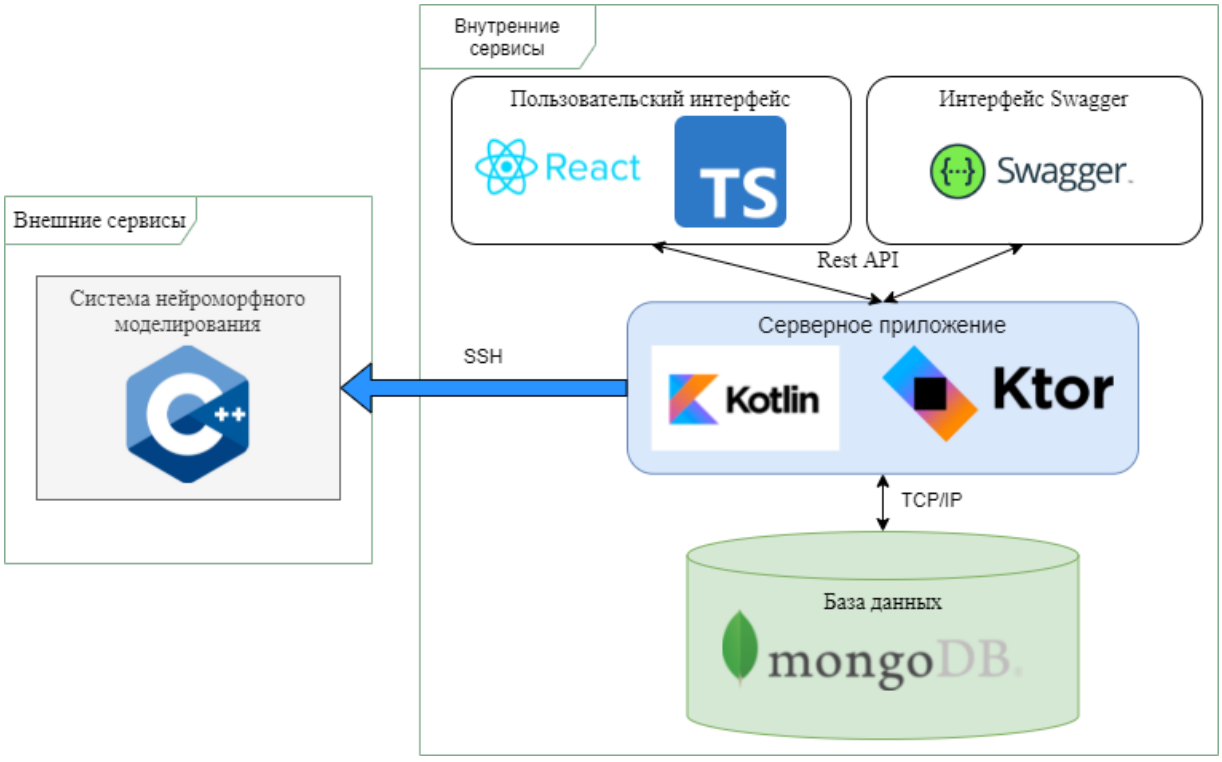


Рис. 2.2. Архитектура сервиса нейроморфного моделирования

2.2. Разработка серверного приложения

В реализуемом приложении на серверную часть возложена следующие обязанности: принимать HTTP запросы от клиентского приложения, сохранять обрабатываемые данные в базу и отправлять команды на исполнение модулю нейроморфного моделирования посредством протокола SSH.

Разработка серверной части программного комплекса согласно шаблону проектирования “Controller-Service-Repository” была разделена на нескольк о условных частей: интергация с модулем, интерфейс для взаимодействия с пользовательским приложением(Controller), реализация бизнес логики(Service) и построение слоя доступа к данным(Repository). С общей схемой компонент внутри серверного приложения можно ознакомиться на рис.2.3. :

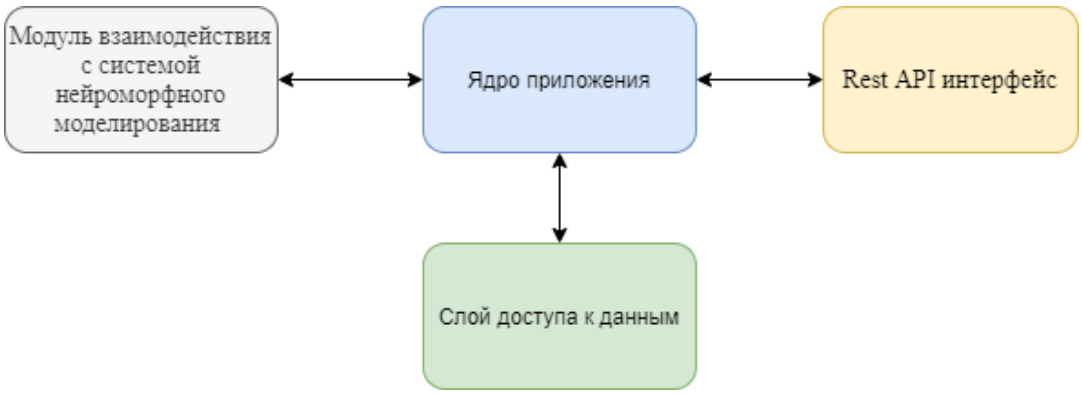


Рис. 2.3. Структура серверного приложения

2.2.1. Интеграция с модулем нейроморфного моделирования

Модуль нейрофорного моделирования представляет собой приложение, позволяющее распознавать заданные изображения (шаблоны) с учетом заданных параметров (seed, количество эпох), используя математические модели мемристоров и нейронов. На вход которому подаются параметры мемристора, нейрона, с помощью которых будет производится распознавание

и сами шаблоны, которые необходимо распознать. Подробнее с структурой входных данных можно ознакомиться в приложении. При запуске приложения, запускается некий итерационный процесс, согласно алгоритму распознавания. На каждой итерации происходит запись лога в стандартный поток вывода(stdout), в котором можно увидеть результат распознавания для текущей итерации алгоритма в виде псевдографики. Также результат каждой итерации записывается в файл out.json, но в другом формате: вместо псевдографики там используется текущие значения весов модели. После окончания процесса распознования, итоговые значения весов, количество итераций, а также полученные изображения(шаблоны) записываются в файл out.json.

Изначально предполагалось произвести интеграцию с модулем нейроморфного моделирования посредством протокола HTTP с помощью REST API. Существовала интеграция модуля с серверным приложением, написанным на языке python с использованием библиотеки django. Данное серверное приложение импортировало модуль, как библиотеку в формате DLL. Соответсвенно, при таком подходе, интеграция с модулем заключалась бы в разработке http клиента для отправки необходимых REST запросов(GET, PUT, POST, DELETE) для запуска/остановки процесса моделирования, а также просмотра результатов и статуса нужных задач. Архитектура такого решения представлена на рис. 2.4.:

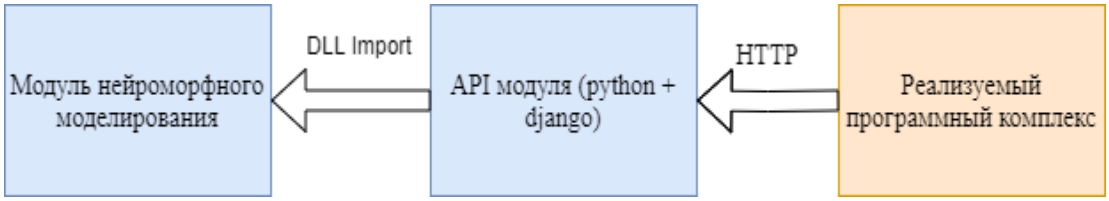


Рис. 2.4. Интеграция с модулем нейроморфного моделирования через внешний API

Для взаимодействия по http требовалось бы реализовать следующие компоненты:

1. Класс ApiSettings с настройками для доступа к API модуля. Файл кофигурации, чтобы можно было менять их, не изменяя исходный код приложения.
2. Классы DTO(data transfer object), позволяющие сохранять данные в формате, пригодном для взаимодействия с API.
3. HTTP клиент для взаимодействия с API. Он должен оперировать данными в формате DTO, выполнять rest запросы согласно параметрам, указанным в ApiSettings(адрес сервера, логин, пароль). Также данный не должен “знать” ничего о том, что происходит уровнем выше, только получение/отправка данных, это позволит избежать замешивания логики, разделить ответственности, упростить разработку и написание тестов.
4. Предусмотреть конвертацию между DTO сущностями и внутренним форматом данных.
5. Юнит тесты, покрывающие разработанный фукционал локальное тестирование и(или) написание интеграционных тестов.

Однако, в связи с рядом причин, было решено убрать из этой схемы лишнее звено и взаимодействовать напрямую с исполняемым файлом модуля на удаленной машине посредством протокола ssh. Тогда архитекура упрощается и имеет вид, показанный на рис.2.5.:

Рис. 2.5. Интеграция с модулем нейроморфного моделирования без посредников

Причины, по которым была выбрана именно эта схема интеграции:

- Устраняется лишнее звено во взаимодействии сервисов, а значит сокращается и число потенциальных источников ошибок и в целом проблем с работоспособностью программного комплекса, также упрощается схема развертывания комплекса: вместо двух серверных приложений, остается одно.
- Происходит разграничение обязанностей: разработчик модуля моделирования беспокоится только о работе своего модуля, как он принимает данные и возвращает результаты. Разработчики же сервисов занимаются вопросам работы сервиса и API, которое он предоставляет.
- Поскольку разработка сервиса ведется не обособленно, а в рамках существующей архитектуры, этот подход позволит снизить затраты на написание нового кода. Так как вся логика взаимодейтсвия по ssh уже реализована, снижается риск получения багов и некорректной работы приложения.

Соответственно для интеграции с модулем, требовалось воспользоваться готовой библиотекой для работы по протоколу ssh и адаптировать ее под необходимый формат. Библиотека предсталяет из себя обобщенный набор классов, для того, чтобы ее можно было бы использовать с приложением нейросетевого моделирования, были реализованы следующие компоненты серверного приложения:

1. Классы на языке kotlin для описания входных/выходных типов данных(с ними можно ознакомиться в приложении).
2. Конверторы из форматов данных внутри приложения в формат, пригодный для модуля моделирования. Поскольку модуль использует json в качетсве формата данных, в качетсве конверторов выступили сериализатор и десериализватор библиотеки org.apache.jackson.
3. Bash скрипты: start.sh, stop.sh, status.sh для запуска, остановки, просмотра статуса задачи соответственно.

Место данного компонента в общей схеме сервиса можно увидеть на рис.2.6.:

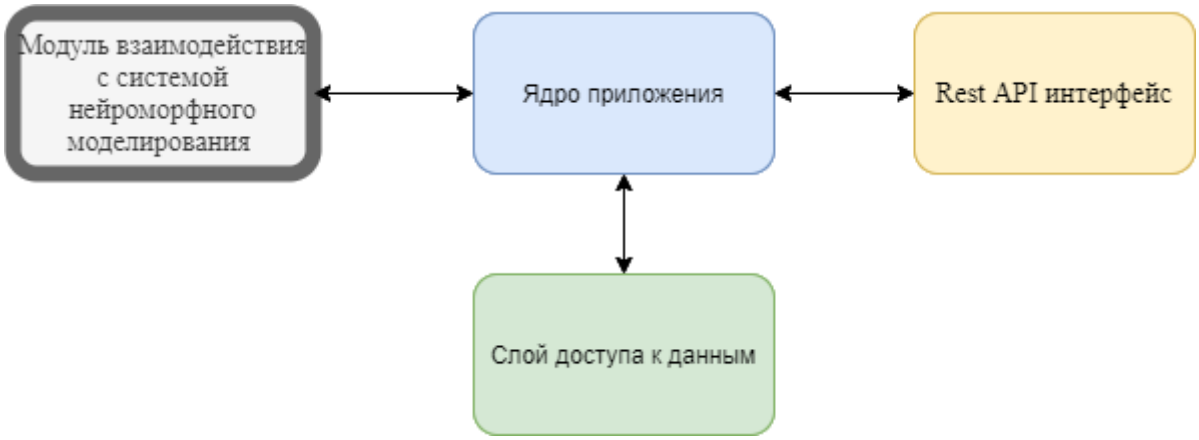


Рис. 2.6. Модуль взаимодействия с системой нейроморфного моделирования.

2.2.2. Бизнес логика

Смотря на схему реализации серверного приложения в разрезе, несложно заметить, что по сути оно состоит из двух основных частей: взаимодействие с модулем и хранение данных. Какой-то насыщенной бизнес логики с разграничением ролей пользователей, различного функционала, не предусматривается, однако для корректного управления приложением необходимо некое ядро, которое в данном случае можно назвать бизнес логикой.

В рамках реализации бизнес логики приложения были предусмотрены следующие моменты:

1. Методы для запуска задачи, просмотра ее статуса и результатов.
2. Выстроено корректное взаимодействие слоя доступа к базе данных, взаимодействие с модулем моделирования слоя бизнес логики.
3. Возможность использовать профили для запуска задач на различных окружениях.
4. Протоколирование для получения информации о ходе работы приложения.

5. Запись запросов на запуск задачи, статуса и результатов задачи, протоколов работы модуля в базу данных.

Место данного компонента в общей схеме сервиса можно увидеть на рис.2.7.:

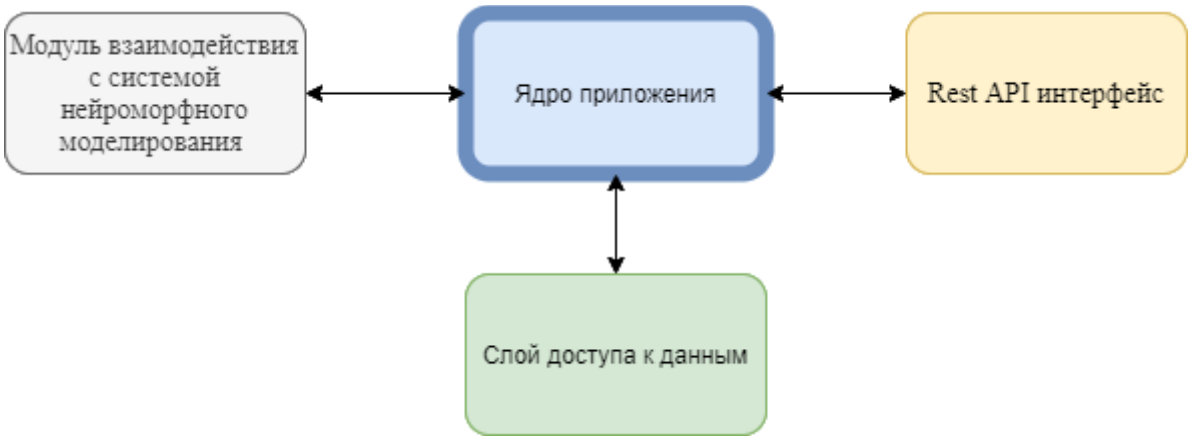


Рис.2.7. "Ядро" приложения

2.2.3. Хранение данных

База данных программного комплекса отвечает за хранение данных по проведенным экспериментам, а также справочникам, ее схема представлена на рис. 2.8.

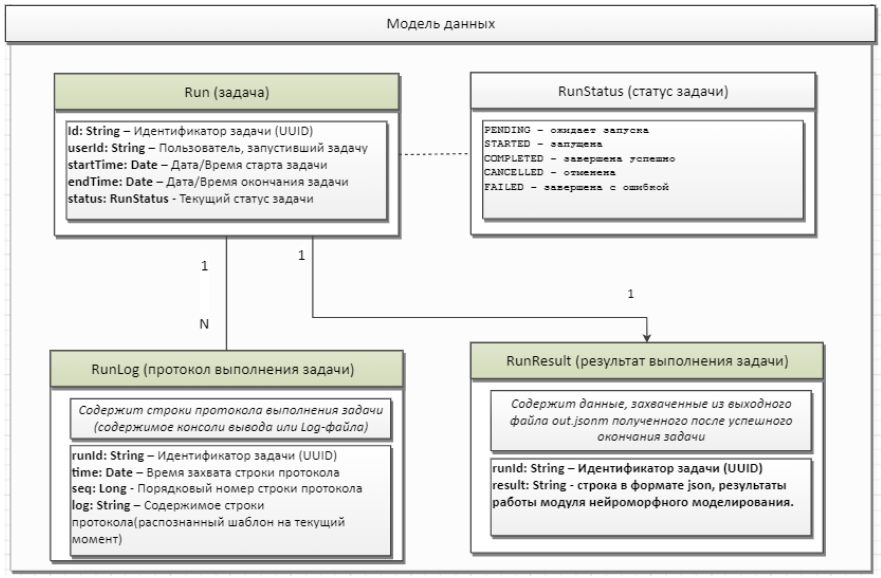


Рис. 2.8. Схема базы данных

В серверном приложении за доступ к данным отвечает компонент Repository. Данные компонент содержит исключительно функционал по работе с базой данных, без какой либо бизнес логики, только чтение/запись/обновление данных.

Фунционал компонента Repository:

1. Сохранение данных по запущенным задачам.
2. Сохранение логов работы модуля нейроморфного моделирования.
3. Получение всех задач с указанным статусом.
4. Получение данных о задаче по Id.
5. Удаление данных о задаче по Id.

Место данного компонента в общей схеме сервиса можно увидеть на рис.2.9.:

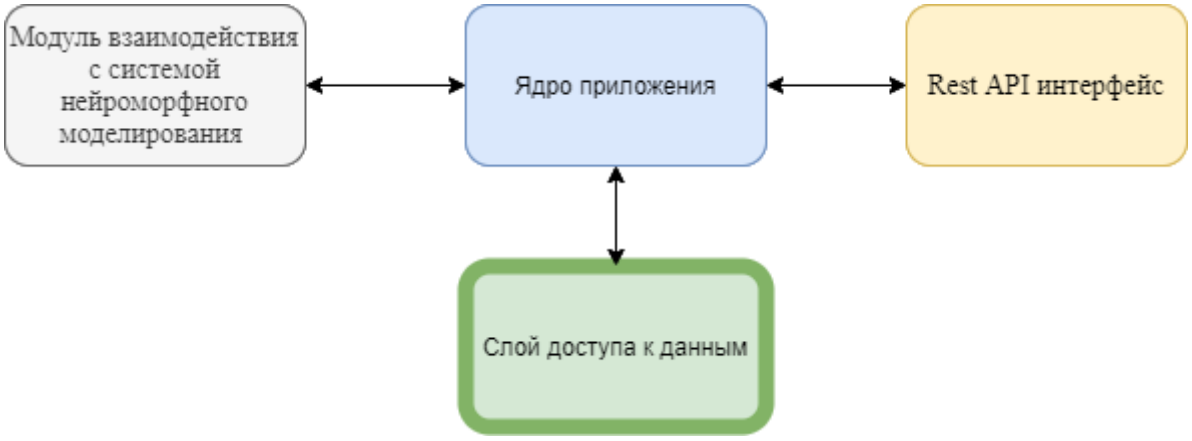


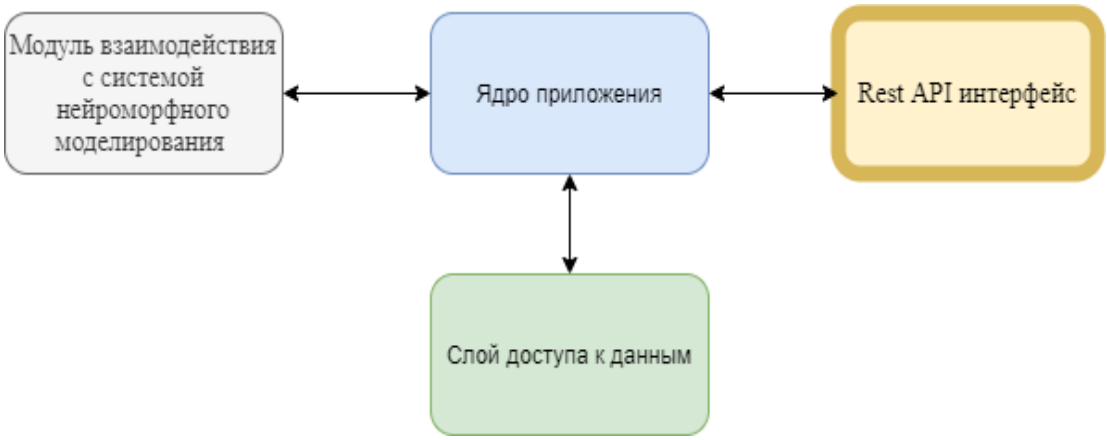
Рис. 2.9. Слой доступа к данным

2.2.4. Программный интерфейс

При реализации API были предусмотрены следующие моменты:

- Описание OpenAPI спецификации, поддержка Swagger.
- Использование endpoint /heartbeat для определения жизнеспособности сервиса.

Для реализации API в серверном приложении использовалась библиотека ktor. Место данного компонента в общей схеме сервиса можно увидеть на рис.2.10.:



2.10. Программный интерфейс

Описание конечных точек программного интерфейса:

- POST /run - запуск процесса распознавания
- GET /run - запрос статусов по всем задачам
- GET /run/{id} - запрос статуса задачи по идентификатору
- GET /run/{id}/result - запрос результата задачи по идентификатору
- DELETE /run/{id}/ - принудительное завершение задачи с переданным иеднтификатором

Примеры HTTP запросов представлены в приложении.

2.3. Разработка клиентского приложения

Структуру клиентского приложения можно условно разделить на две части:

- Графический интерфейс – только отрисовка компонентов “чистыми” функциями, используется так называемый React.FunctionalComponent.
- Бизнес логика – работа с состоянием приложения, то есть управление данными в нем. Используется Redux. При необходимости отправляются соответствующие rest запросы.

Итоговая схема приложения представлена на рис.2.11.

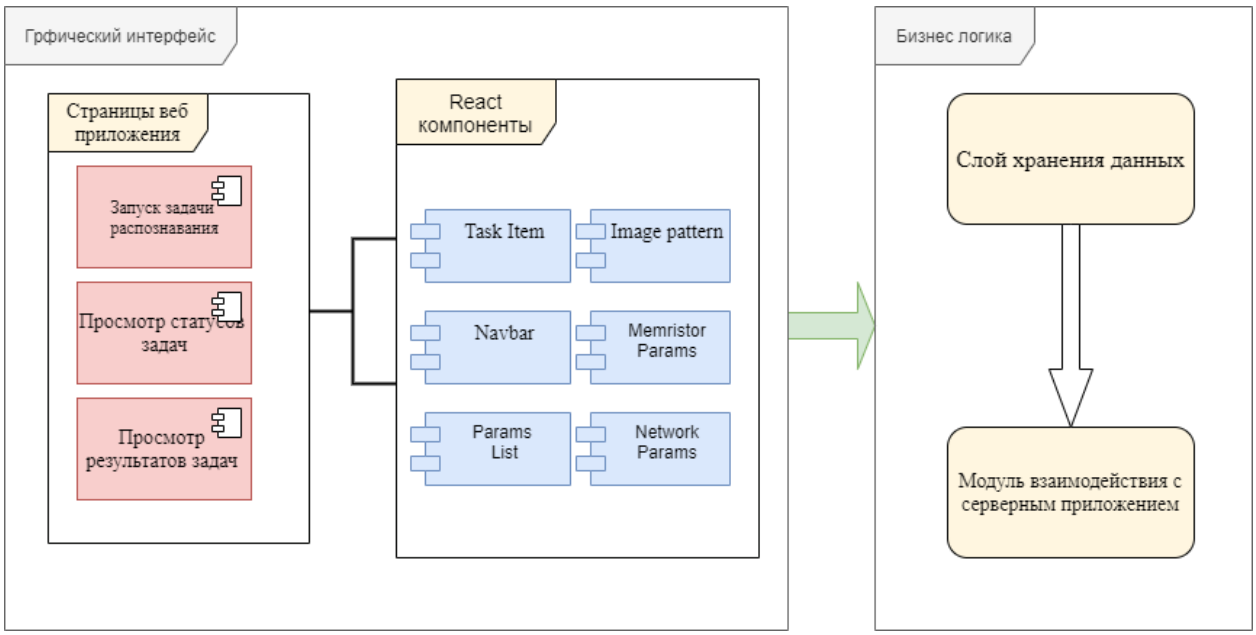


Рис. 2.11. Компоненты веб приложения

Страницы приложения:

1. Run task – запуск задачи, согласно переданным параметрам.
2. Tasks status – просмотр статусов запущенных задач.
3. Tasks results - Просмотр результатов запущенных задач.

React компоненты:

- Task Item – информация о задаче: время запуска, статус.
- Image Pattern – компонент, отвечающий за отрисовку шаблонов распознования, а также результатов распознавания, используется на страницах Run task, Tasks results. Реализован с использованием встроенного в язык java script Canvas API.
- Memristor Params – динамический список параметров математической модели мемристора.
- Network Params – динамический список параметров математической нейросети.

Для взаимодействия с API серверного приложения был реализован класс – обертка над методом fetch. Описание типов данных для взаимодействия с серверным приложением на языке TypeScript приведено в приложении.

2.4. Взаимодействие серверного и клиентского приложений

Для того, чтобы выстроить взаимодействие между серверным и клиентским приложением были рассмотрены следующие варианты:

1. Использование для взаимодействия серверного и клиентского приложений только REST API

Плюсы:

- Простота реализации
- Возможность пользователя коммуницировать с сервером не только через UI, но и через Swagger, либо выполнять запросы с помощью утилиты curl (если, к примеру на операционной системе нет возможности использовать UI).

Минусы:

- Скорость меньше, чем при использовании WebSocket.

2. Использование для взаимодействия backend-frontend только Web Socket.

Плюсы:

- Высокая скорость взаимодействия.
- Двунаправленный режим передачи данных.

Минусы:

- Более сложный в написании и поддержке код, как backend, так и frontend.
- Нет возможности использовать swagger или выполнять запросы через curl.

3. Использование для взаимодействия backend-frontend Web Socket, но также оставить rest endpoints

Плюсы:

- Сочетает в себе все плюсы использования технологий по отдельности

Минусы:

- Слишком много кода для написания, поддержки и отладки.

В связи с тем, что высоких нагрузок на сервис не ожидается, а также, нет необходимости в двунаправленном режиме передачи данных, да и в целом использование сложных технологий в данном случае избыточно, был выбран первый вариант - использование только REST API. . Итоговая схема взаимодействия клиентского и серверного приложений представлена на рис.2.12.

Рис. 2.12. Взаимодействие пользовательского и серверного приложений

ЗАКЛЮЧЕНИЕ

В рамках работы над магистерской диссертацией были рассмотрены ключевые моменты при работе с системой нейроморфного моделирования, согласно которым были предложены пользовательские сценарии и заданы функциональные требования. При проектировании архитектуры программного комплекса, удовлетворяющего данным требованиям, были учтены лучшие практики и шаблоны разработки. По данной архитектуре был разработан программный комплекс, включающий в себя серверное приложение и графический интерфейс. Для серверного приложения была произведена интеграции с модулем нейроморфного моделирования. Также было выстроено взаимодействие между клиентским и серверным приложением. Готовый программный комплекс предполается интегрировать в платформу многомасштабного моделирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Морозов А. Ю., Ревизников Д. Л., Абгарян К. К. Вопросы реализации нейросетевых алгоритмов на мемристорных кроссбарах // Известия высших учебных заведений. Материалы электронной техники, 2019. Т. 22. № 4

[2] Морозов А.Ю., Абгарян К.К., Ревизников Д.Л. Математическое моделирование самообучающейся нейроморфной сети, основанной на наноразмерных мемристивных элементах с 1T1R-кроссбар-архитектурой // Известия высших учебных заведений. Материалы электронной техники. 2020;23(3):186-195.

[3] Абгарян К.К, Гаврилов Е.С. «Интеграционная платформа для многомасштабного моделирования нейроморфных систем», «Информатика и ее применения», том 14, выпуск 2, 2020 г. Стр. 104-110.

[4] Щербаков В.С. Разработка программного и пользовательского интерфейсов для управления моделями нейроморфных вычислительных систем // Гагаринские чтения – 2021: XLV Международная молодёжная научная конференция: Сборник тезисов докладов: М.; Московский авиационный институт (национальный исследовательский университет), 2019. – С. 476-477.

[5] Fowler, M., and P. J. Sadalage. 2012. NoSQL distilled: A brief guide to the emerging world of polyglot persistence. Addison-Wesley Professional.

[6] Newman, S. 2015. Building microservices. Sebastopol, CA: O'Reilly Media.

[7] Jemerov, D., Isakova, S. Kotlin in Action. Manning, 2017 г.

[8] Robert C. Martin, James Grenning, Simon Braun. 2018. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson Education, Inc.

[9] Kirupa Chinnathambi. Learning React. 2018, Pearson Education, Inc.

[10] Marc Garreau, Will Faurot, Redux in action. 2018, Manning Publications Co.

[11] Michael S. Mikowski, Josh C. Powell, Single Page Web Applications, 2014, Manning Publications Co.

[12] JEFF NICKOLOFF, Docker in Action., 2016, Manning Publications Co.

[13] Yang J. J., Strukov D. B., Stewart D. R. Memristive devices for computing // Nature Nanotechnology, 2013. Vol. 8, no. 1, pp. 13–24. DOI: 10.1038/nnano.2012.240.

[14] KYLE BANKER, PETER BAKKUM, SHAUN VERCH, DOUGLAS GARRETT, TIM HAWKINS, MongoDB in Action, 2016, Manning Publications Co.

[15] Leonard Richardson, Mike Amundsen, RESTful Web APIs, 2013, O'Reilly Media, Inc.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1. Скриншоты интерфейсов.

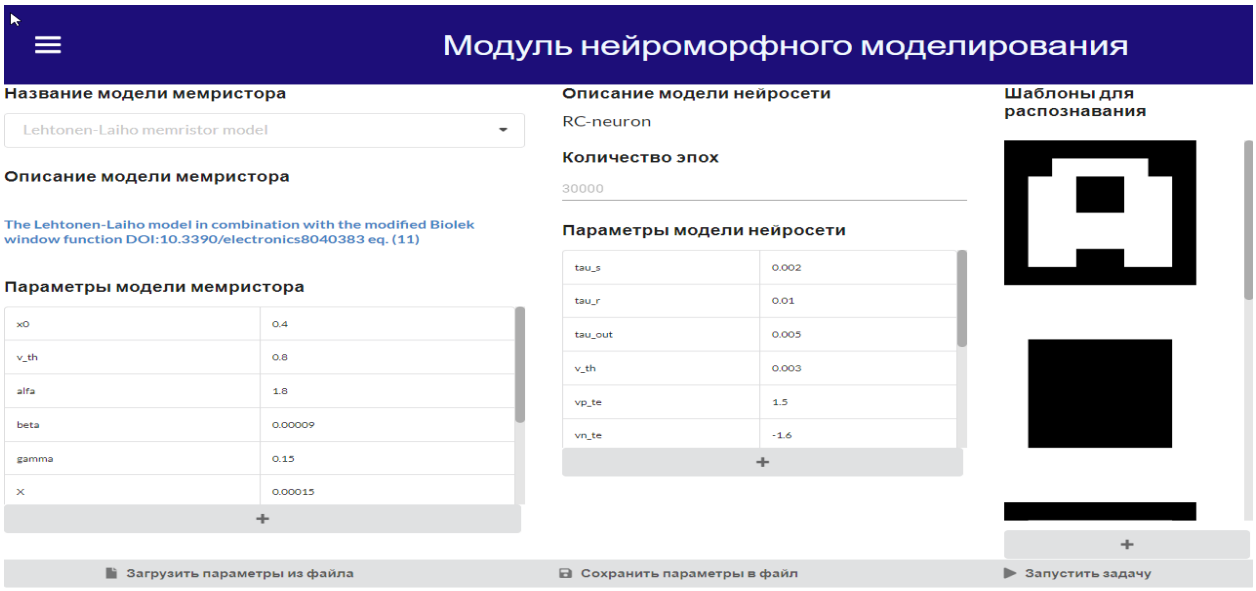


Рис. П.1.1. Основная страница веб интерфейса

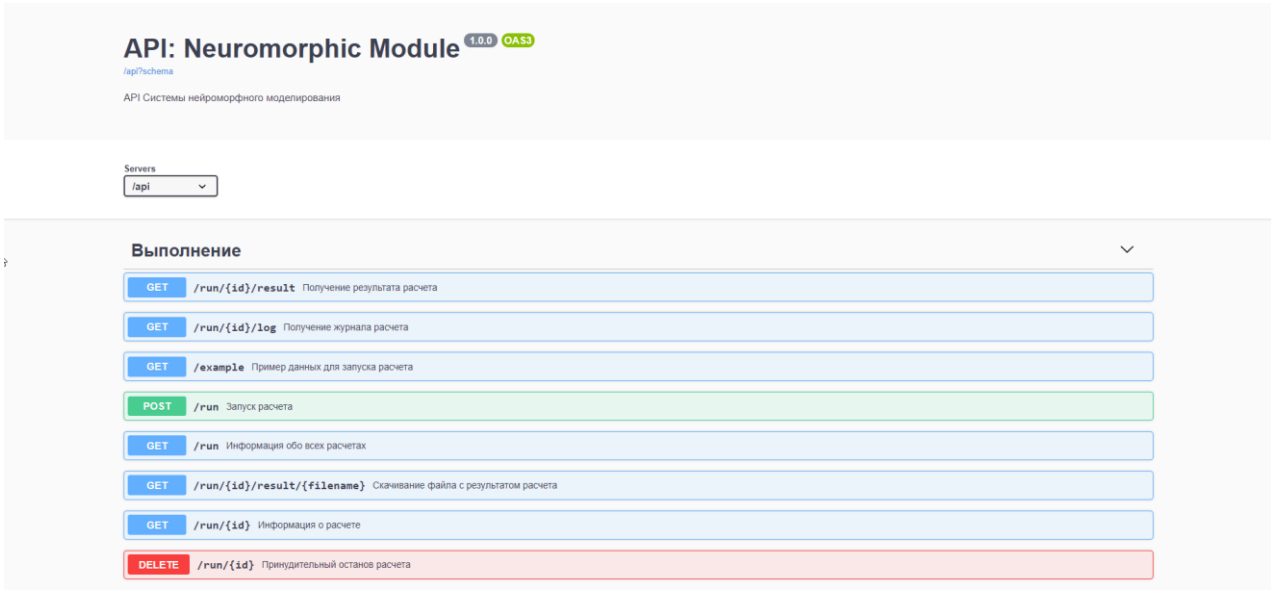


Рис. П.1.2. Swagger интерфейс.

ПРИЛОЖЕНИЕ 2. Пример записи из базы данных.


```
{
  "_id" : "60aa13d82635686502d5daa1",
  "runId" : "60aa13d82635686502d5da9f",
  "input" : {
    "@class" :
"ru.mai.crystal.modules.neuromorphic.contracts.NeuralNetworkModelingModuleInp
ut",
    "module" : "neural_network_modeling",
    "seed" : 457,
    "n_out" : 100,
    "n_epochs" : 100,
    "memristor" : {
      "id" : 1,
      "v_p" : 0.65,
      "v_n" : -0.87,
      "d" : 6.208429e-07,
      "r_off" : 2128.27,
      "r_on" : 204.9949,
      "nu_v" : 5.952302e-10,
      "x0" : 0.111025
    },
    "neuron" : {
      "tau_s" : 5e-05,
      "tau_r" : 0.003,
      "tau_out" : 0.0015,
      "v_th" : 0.009,
      "alfa" : 0.1,
      "v0_te" : 0.01,
      "vp_out" : 2.0,
      "vp_te" : 0.7,
      "vn_te" : -0.9,
      "r_int" : 200.0,
      "p_noise" : 0.27,
      "c_int" : 4.5e-05
    },
    "n_patterns" : 5,
    "w" : 8,
    "patterns" : [
      "java.util.ArrayList",
      [
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 0, 0, 1, 1, 0,
        0, 1, 1, 0, 0, 1, 1, 0,
        0, 1, 1, 1, 1, 1, 1, 0,
        0, 1, 1, 1, 1, 1, 1, 0,
        0, 1, 1, 0, 0, 1, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0
      ]
    ],
    "h" : 8
  },
  "createdTime" : ISODate("2021-05-23T08:35:36.022Z")
}
```

ПРИЛОЖЕНИЕ 3.Листинг кода для взаимодействия клиентского и серверного приложений по протоколу Rest API.

```
@JsonIgnoreProperties(ignoreUnknown = true)
class NeuralNetworkModelingModuleInput (
    @get:JsonProperty("module")
    val ModuleName: String = "neural_network_modeling",
    @get:JsonProperty("seed")
    var Seed: Int = 0,
    @get:JsonProperty("n_epochs")
    var CountEpochs: Int = 0,
    @get:JsonProperty("n_out")
    var CountOut: Int = 0,
    @get:JsonProperty("memristor")
    var Memristor: Memristor = Memristor(),
    @get:JsonProperty("neuron")
    var Neuron: Neuron = Neuron(),
    @get:JsonProperty("n_patterns")
    var CountPatterns: Int = 0,
    @get:JsonProperty("w")
    var W: Int = 0,
    @get:JsonProperty("h")
    var H: Int = 0,
    @get:JsonProperty("patterns")
    var PatternsArray: List<Int> = emptyList()
)
@JsonIgnoreProperties(ignoreUnknown = true)
@JsonRootName("memristor")
class Memristor(
    @get:JsonProperty("id")
    var Id: Int = 0,
    @get:JsonProperty("x0")
    var X0: Double = 0.0,
    @get:JsonProperty("v_p")
    var VP: Double = 0.0,
    @get:JsonProperty("v_n")
    var VN: Double = 0.0,
    @get:JsonProperty("nu_v")
    var NuV: Double = 0.0,
    @get:JsonProperty("r_on")
    var ROn: Double = 0.0,
    @get:JsonProperty("r_off")
    var RcOff: Double = 0.0,
    @get:JsonProperty("d")
    var D: Double = 0.0
)


@JsonIgnoreProperties(ignoreUnknown = true)
@JsonRootName("neuron")
class Neuron(
        @get:JsonProperty("tau_s")
        var TauS: Double = 0.0,
        @get:JsonProperty("tau_r")
        var TauR: Double = 0.0,
        @get:JsonProperty("tau_out")
        var TauOut: Double = 0.0,
        @get:JsonProperty("v_th")
        var VTh: Double = 0.0,
        @get:JsonProperty("vp_te")
        var VpTe: Double = 0.0,
```

```
@get:JsonProperty("vn_te")
var VnTe: Double = 0.0,
@get:JsonProperty("v0_te")
var V0Te: Double = 0.0,
@get:JsonProperty("vp_out")
var VpOut: Double = 0.0,
@get:JsonProperty("c_int")
var CInt: Double = 0.0,
@get:JsonProperty("r_int")
var RInt: Double = 0.0,
@get:JsonProperty("alfa")
var Alfa: Double = 0.0,
@get:JsonProperty("p_noise")
var PNoise: Double = 0.0)
```

```
type Memristor = {
    id: number;
    x0: number;
    v_th: number;
    "alfa": number;
    "beta": number;
    "gamma": number;
    X: number;
    a: number;
    b: number;
    c: number;
    n: number
}

type Neuron = {
    tau_s: number;
    tau_r: number;
    tau_out: number;
    v_th: number;
    vp_te: number;
    vn_te: number;
    v0_te: number;
    vp_out: number;
    c_int: number;
    r_int: number;
    alfa: number;
    p_noise: number;
}

type RunInput = {
    module: String;
    seed: number;
    n_epochs: number;
    n_out: number;
    memristor: Memristor;
    neuron: Neuron;
    n_patterns: number;
    w: number;
    h: number;
    patterns: number[][][]
}
```

```
install(CORS){
    method(HttpMethod.Options)
    method(HttpMethod.Post)
    method(HttpMethod.Get)
    method(HttpMethod.Put)
    method(HttpMethod.Delete)
    method(HttpMethod.Head)
    header(HttpHeaders.AccessControlAllowHeaders)
    header(HttpHeaders.ContentType)
    header(HttpHeaders.AccessControlAllowOrigin)
    header(HttpHeaders.AccessControlAllowMethods)
    allowCredentials = true
    allowNonSimpleContentTypes = true
    hosts.addAll(listOf("http://localhost:3000","http://localhost:3001",
"http://localhost:8080"))
}
```

```
{
    method: 'POST',
    mode: 'cors',
    headers: {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
        'Access-Control-Request-Headers': 'Content-Type, Authorization'
    },
    body: inputJson
}
```

ПРИЛОЖЕНИЕ 4. Примеры HTTP запросов к серверному приложению.

```
GET http://localhost:8080/api/example

HTTP/1.1 200 OK
Vary: Origin
Content-Length: 1826
Content-Type: text/plain; charset=UTF-8
Connection: keep-alive

{
  "module": "neural_network_modeling",
  "seed": 457,
  "n_epochs": 30000,
  "n_out": 100,

  "memristor": {
    "id": 1,
    "x0": 1.110250e-01,
    "v_p": 6.500000e-01,
    "v_n": -8.700000e-01,
    "nu_v": 5.952302e-10,
    "r_on": 2.049949e+02,
    "r_off": 2.128270e+03,
    "d": 6.208429e-07
  },

  "neuron": {
    "tau_s": 0.00005,
    "tau_r": 0.003,
    "tau_out": 0.0015,
    "v_th": 0.009,
    "vp_te": 0.7,
    "vn_te": -0.9,
    "v0_te": 0.01,
    "vp_out": 2.0,
    "c_int": 4.5e-05,
    "r_int": 200.0,
    "alfa": 0.1,
    "p_noise": 0.27
  },
  "n_patterns": 5,
  "w": 8,
  "h": 8,
  "patterns":
[
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 1, 1, 1, 1, 0, 0,
    0, 1, 1, 0, 0, 1, 1, 0,
    0, 1, 1, 0, 0, 1, 1, 0,
    0, 1, 1, 1, 1, 1, 1, 0,
    0, 1, 1, 1, 1, 1, 1, 0,
    0, 1, 1, 0, 0, 1, 1, 0,
    0, 0, 0, 0, 0, 0, 0, 0
  ]
}

Response code: 200 (OK); Time: 880ms; Content length: 1826 bytes

GET http://localhost:8080/api/run

HTTP/1.1 200 OK
Vary: Origin
```

```
Content-Type: application/json; charset=UTF-8
Connection: keep-alive
transfer-encoding: chunked

[
  {
    "id": "60aa0b6b75283276a3375525",
    "name": "neuromorphic module run",
    "status": "READY_TO_START"
  }
]

GET http://localhost:8080/api/run/60a9f88120ef102391a80a47

HTTP/1.1 200 OK
Vary: Origin
Content-Type: application/json; charset=UTF-8
Connection: keep-alive
transfer-encoding: chunked

{
  "id": "60a9f88120ef102391a80a47",
  "name": "neuromorphic module run",
  "status": "READY_TO_START"
}

Response code: 200 (OK); Time: 233ms; Content length: 108 bytes


POST /api/run?userId=test-user&profile=test HTTP/1.1
Host: http://localhost:8080
Content-Type: application/json
Accept: application/json


{
  "module": "neural_network_modeling",
  "seed": 457,
  "n_epochs": 30000,
  "n_out": 100,

  "memristor": {
    "id": 1,
    "x0": 1.110250e-01,
    "v_p": 6.500000e-01,
    "v_n": -8.700000e-01,
    "nu_v": 5.952302e-10,
    "r_on": 2.049949e+02,
    "r_off": 2.128270e+03,
    "d": 6.208429e-07
  },

  "neuron": {
    "tau_s": 0.00005,
    "tau_r": 0.003,
    "tau_out": 0.0015,
    "v_th": 0.009,
    "vp_te": 0.7,
    "vn_te": -0.9,
    "v0_te": 0.01,
    "vp_out": 2.0,
    "c_int": 4.5e-05,
    "r_int": 200.0,
    "alfa": 0.1,
```

```
    "p_noise": 0.27
  },
  "n_patterns": 1,
  "w": 8,
  "h": 8,
  "patterns":
  [
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 1, 1, 1, 1, 0, 0,
    0, 1, 1, 0, 0, 1, 1, 0,
    0, 1, 1, 0, 0, 1, 1, 0,
    0, 1, 1, 1, 1, 1, 1, 0,
    0, 1, 1, 1, 1, 1, 1, 0,
    0, 1, 1, 0, 0, 1, 1, 0,
    0, 0, 0, 0, 0, 0, 0, 0
  ]
}
```


```
HTTP/1.1 200 OK
Vary: Origin
Content-Type: application/json; charset=UTF-8
Connection: keep-alive
transfer-encoding: chunked
```


```
{
  "id": "60ac8c079adfb31ca15f9c27",
  "name": "neuromorphic module run",
  "status": "IN_PROGRESS"
}
```


```
DELETE http://localhost:8080/api/run/60ac7d11a0ed665eb819d35b

HTTP/1.1 200 OK
Vary: Origin
Content-Length: 7
Content-Type: text/plain; charset=UTF-8
Connection: keep-alive

stopped

Response code: 200 (OK); Time: 6588ms; Content length: 7 bytes
```