ТИТУЛ

ЗАДАНИЕ

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ....3
ВВЕДЕНИЕ....4
1 Проектирование архитектуры и бизнес-логики....6
2 Проектирование базы данных и структур данных....10
2.1 Разработка даталогической схемы БД....10
2.2 Описание структур данных для описания заданий и ответов....12
3 Проектирование микросервисов....15
3.1 Микросервис взаимодействия с БД....15
3.2 Синтезатор....19
3.3 Преобразователь формата временных диаграмм и генератор wavedrom-диаграмм....20
3.4 Микросервис анализа статистики....25
3.5 Анализатор решений....26
ЗАКЛЮЧЕНИЕ....28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ....29
ПРИЛОЖЕНИЕ X – диаграммы последовательности проверки кода и редактирования задания, device.v, tb.v, *.vcd, ...30

ВВЕДЕНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД – база данных

СУБД — система управления базами данных

CRUD — create read update delete

VCD — value change dump

Временная диаграмма — ...

Wavedrom — это бесплатный онлайн-движок с открытым исходным кодом для рендеринга цифровых временных диаграмм.

Http-запрос — ...

JSON — ...

Модель C4 — ...

1 Проектирование архитектуры и бизнес-логики

Разработанная подсистема используется веб-приложением образовательного портала для управления учебными материалами, проверки пользовательских ответов на задания и работы со статистикой решения заданий.

Поскольку, информация о пользователях используется как в разработанной подсистеме, так и в других компонентах программного обеспечения образовательного портала, БД используется совместно.

Обобщенная архитектура разработанной подсистемы показана с помощью контекст-диаграммы в нотации C4 на рисунке 1.

Рисунок 1 — обобщенная архитектура разработанной подсистемы

На основе функциональных требований, предъявляемых к разработанной подсистеме, была создана диаграмма вариантов использования, представленная на рисунке 2.

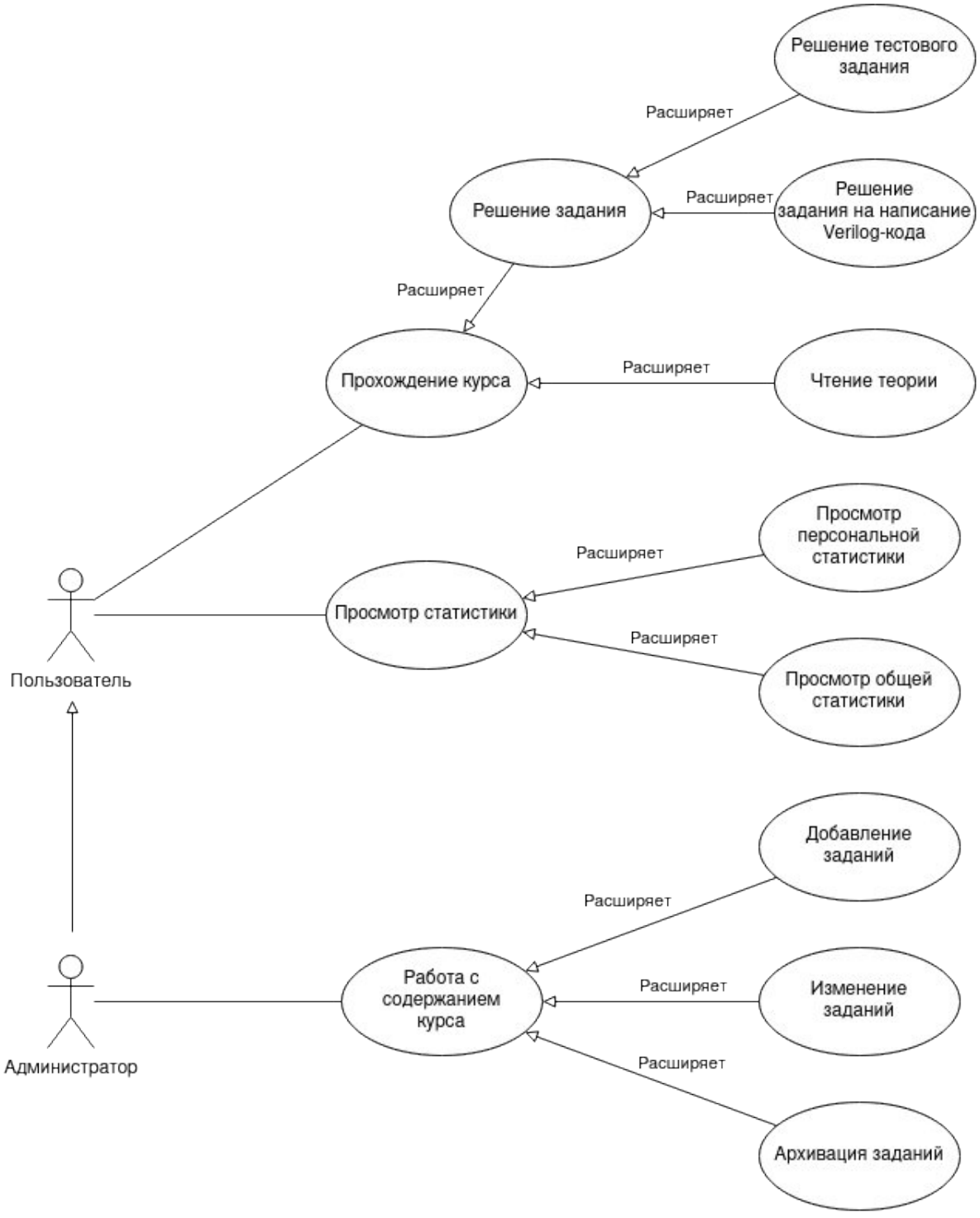


Рисунок 2 — диаграмма вариантов использования подсистемы

На представленной диаграмме демонстрируется, что в системе было выделено две роли: пользователь (учащийся) и администратор (редактирует содержание образовательных материалов). При этом администратор одновременно может являться и учащимся.

Требуемые варианты использования было решено реализовать с помощью следующего набора компонентов:

- микросервис взаимодействия с БД — реализует CRUD-операции над данными в БД;
- анализатор — выполняет проверку и анализ пользовательских решений;
- синтезатор (симулятор, компилятор) — выполняет синтез устройств из Verilog-кода и симулирует их работу;
- преобразователи форматов временных диаграмм — преобразуют временные диаграммы в удобные для хранения и обработки форматы;
- микросервис анализа статистики;
- компонент бизнес-логики — реализует бизнес-логику подсистемы, связывает остальные микросервисы.

Детализированная архитектура разработанной подсистемы показана на контейнер-диаграмме (нотация C4) на рисунке 3.

Наиболее сложные варианты использования, иллюстрирующие взаимодействие показанных компонентов — «работа с содержанием курса» и «решение задания на написание Verilog-кода». Диаграммы последовательности действий для этих вариантов использования приведены в приложении А.

Рисунок 3 — детализированная архитектура разработанной подсистемы

2 Проектирование базы данных и структур данных

2.1 Разработка даталогической схемы БД

В результате анализа предметной области удалось выделить описанные ниже сущности.

Сущность «Задание» — содержит информацию о порядковом номере задания, его условиях, правильном ответе, цене в баллах и т.п.;

Сущность «Пользователь» — позволяет идентифицировать пользователя по ID, узнать, обладает ли пользователь правами администратора и узнать его псевдоним (т.н. «никнейм»). Кроме того, эта сущность может нести в себе дополнительную информацию, необходимую веб-приложению образовательного портала.

Сущность «Попытка решения» — содержит информацию, об успешности и времени каждой попытки решения задания каким-либо пользователем.

Для реализации базы данных была выбрана реляционная СУБД MySql, для ускорения работы SQL-запросов, анализирующих статистику прохождения заданий или выдающих другую агрегированную информацию по курсу, было решено разделить сущность «Задание» на «Брифинг задания» и «Данные задания», а так же выделить отдельную сущность «Тип задания».

Для кратких текстовых полей, таких, как «Название задания» используется тип var, а для длинных — TEXT. Логические занчения сохраняются в tinyint(1).

Полученная даталогическая схема БД в нотации Мартина изображена на рисунке 4.

Ниже представлено подробное описание приведенных таблиц и их полей.

Таблица Users (пользователи):

- id — первичный ключ;
- nickname — псевдоним;
- is_admin — признак администратора.

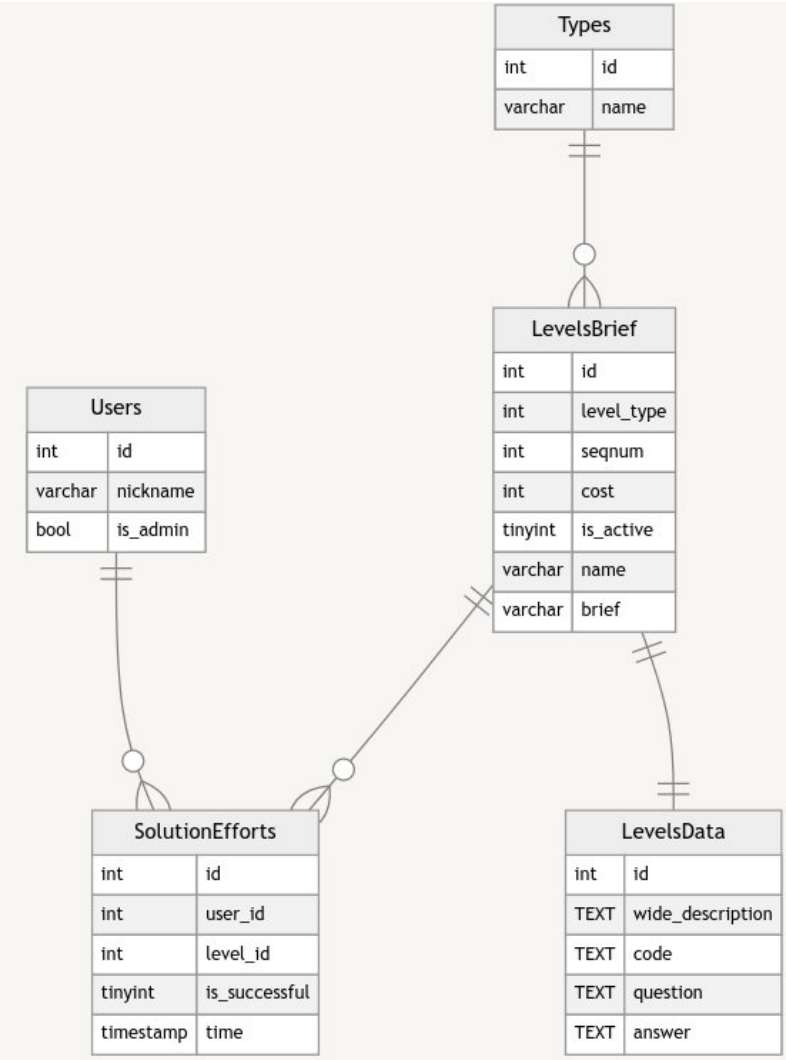


Рисунок 4 — даталогическая схема БД

Таблица LevelsBrief (краткая информация о заданиях):

- id — первичный ключ;
- level_type — тип задания;
- seqnum — порядковый номер задания в списке (может повторяться у «заархивированных» заданий);
- cost — количество баллов, начисляемых за решение задания;
- is_active — признак активности задания (если is_active = 0, задание считается «заархивированным»);
- name — название задания;
- brief — краткое описание задания.

Таблица LevelsData (подробная информация о заданиях):

- id — первичный ключ, совпадает с id задания в LevelsBrief;
- wide_description — развернутое описание задания;
- code — листинг исходного кода на Verilog, который может быть приложен к заданию;
- question — закодированные условия задания;
- answer — закодированный ответ на задание.

Таблица Types (типы заданий):

- id — первичный ключ;
- name — название типа задания.

Таблица SolutionEfforts (попытки решения заданий):

- id — первичный ключ;
- user_id — id пользователя;
- level_id — id задания;
- is_succesful — признак успешного прохождения задания;
- time — дата и время прохождения задания.

2.2 Описание структур данных для описания заданий и ответов

Так, как в разрабатываемой подсистеме используются задания различных типов, которые необходимо проверять автоматически, было решено хранить информацию об условиях и ответах на каждое задание в закодированном виде в одном поле БД (это позволило использоваться реляционную модель, позволяющую, например, удобным образом анализировать статистику выполнения заданий).

Разработанная подсистема поддерживает работу с тремя типами заданий:

- тесты с выбором одного варианта ответа;
- тесты с множественным выбором;
- задания на описание устройства на Verilog.

Все задания и ответы сохраняются в нотации JSON.

Условия задания с выбором одного ответа содержат заголовок задания (caption) и массив ответов (answers), в котором каждый ответ имеет поля с текстом варианта ответа (text) и подсказкой, которая будет показана пользователю, если ответ неверен (hint).

Формат описания задания с выбором одного ответа приведен на рисунке 5.

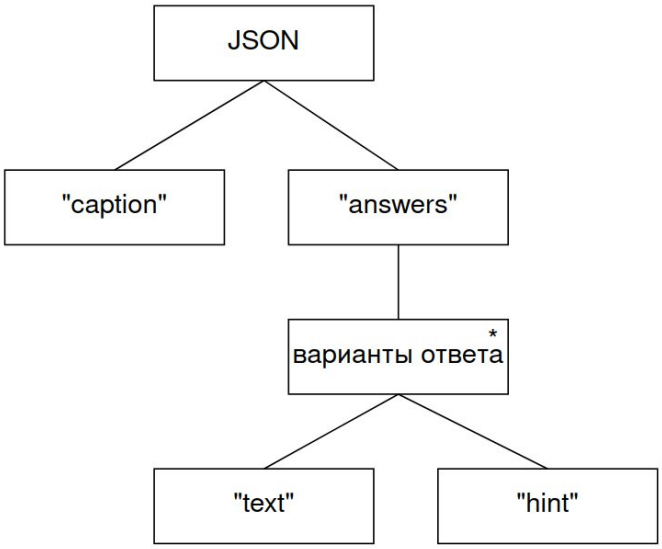


Рисунок 5 — формат описания задания с выбором одного ответа

Пример описания задания с одним ответом приведен в листинге 1.

Листинг 1 — пример описания задания с одним ответом

```
// условия задания с выбором одного варианта ответа

{
 "caption": "Основная функция сумматора",
 "answers": [
  {"text": "Умножение", "hint": "Название говорит само за себя"},
  {"text": "Вычитание", "hint": "Перечитай главу"},
  {"text": "Сложение", "hint": "Все верно"}
 ]
}

// ответ на задание с выбором одного варианта ответа

{"correct_answer_id":2}
```

Условия задания с выбором нескольких вариантов ответа хранятся в аналогичном формате, но в них отсутствует поле hint.

Примеры записи условия для заданий с множественным выбором приведен в листинге 2.

Листинг 2 — пример описания задания с множественным выбором

```
// условия задания с выбором нескольких вариантов ответа

{
  "caption": "Типы переменных в verilog",
  "answers": [
    "reg",
    "wire",
    "mem"
  ]
}

// условия ответ на задание с выбором нескольких вариантов ответа

{"correct_answers": [true, true, false]}
```

В случае задания на описание устройства с помощью языка Verilog, в поле LevelsData.question заносится код теста устройства на языке Verilog (т.н. «testbench», см. приложение Б), а в поле LevelsData.answer — описание временной диаграммы корректно описанного устройства в формате wavedrom (см. раздел «Генератор wavedrom-диаграмм»).

3 Проектирование микросервисов

3.1 Микросервис взаимодействия с БД

Для реализации CRUD-операций с данным, хранящимися в БД, был реализована микросервис взаимодействия с БД.

Логика работы с каждой из таблиц базы данных инкапсулирована в отдельный класс, каждый из таких классов работает с БД через класс соединения с БД, который в свою очередь использует драйвер СУБД MySQL (рисунок 6).

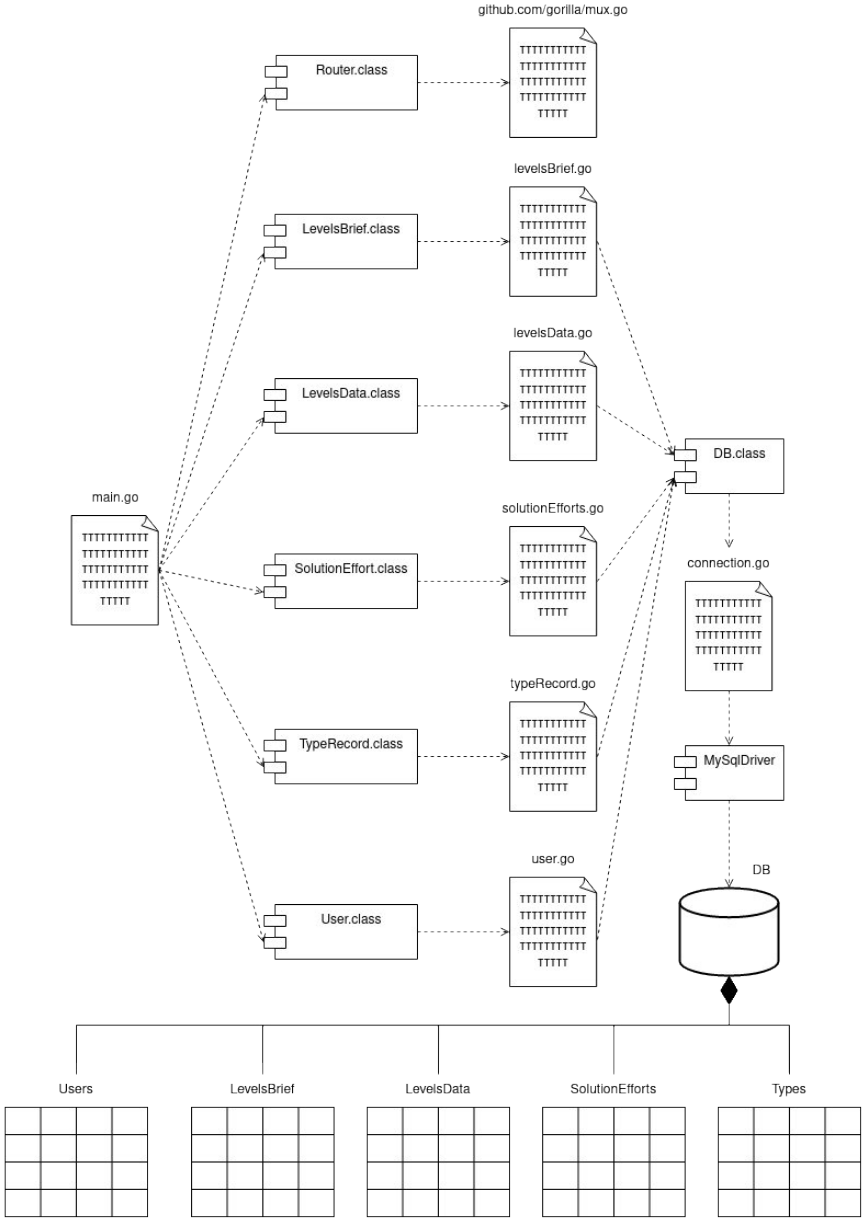


Рисунок 6 — диаграмма компоновки микросервиса взаимодействия с БД

На рисунке 7 (сделать подпись!) представлена диаграмма классов описываемого микросервис.

Классы LevelsBrief, LevelsData, SolutionEffort, TypeRecord и User реализуют взаимодействие с БД и воплощают в себе сущности предметной области.

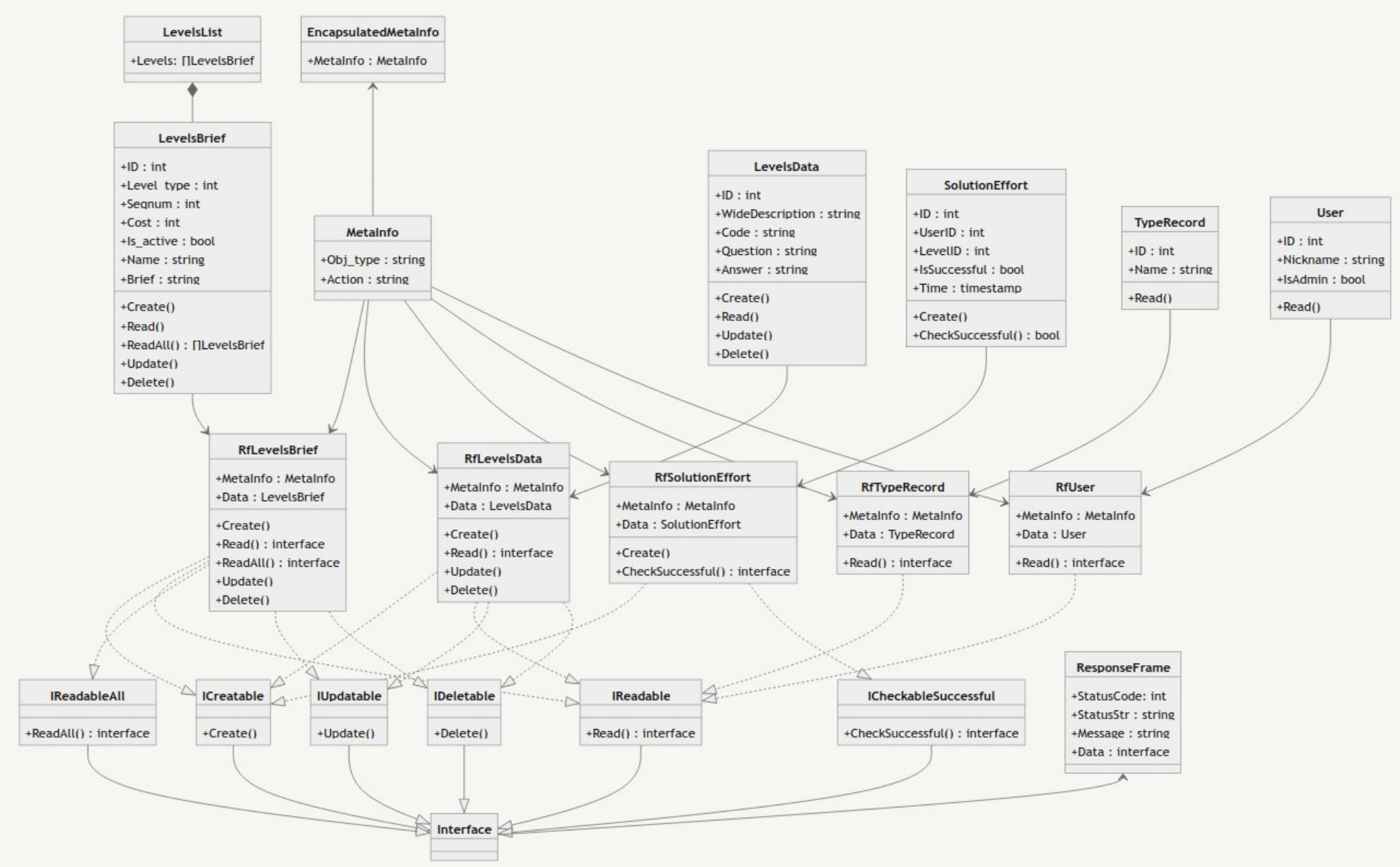
Класс MetaInfo содержит поля ObjType (сущность, надо которой выполняется операция) и Action (тип операции).

Классы с префиксом «Rf» (сокращение от «Request Frame») позволяют разобрать входные сообщения, разделив метаинформацию и данные о сущности предметной области.

Интрефейсы IReadable, IUpdatable и т.п. позволяют взаимодействовать с любым типом сущностей по одному и тому же алгоритму.

Использование типа interface для ResponseFrame.Data (данные ответного сообщения) так же позволяет записывать в это поле данные об объекте любого класса.

Примечание: любой класс в Golang является реализацией interface, однако не все отношения реализации показаны на диаграмме классов с целью ее упрощения.


LevelsList
+Levels: []LevelsBrief
EncapsulatedMetaInfo
+MetaInfo : MetaInfo
LevelsBrief
+ID : int
+Level_type : int
+Seqnum : int
+Cost : int
+Is_active : bool
+Name : string
+Brief : string
+Create()
+Read()
+ReadAll() : []LevelsBrief
+Update()
+Delete()
MetaInfo
+Obj_type : string
+Action : string
LevelsData
+ID : int
+WideDescription : string
+Code : string
+Question : string
+Answer : string
+Create()
+Read()
+Update()
+Delete()
SolutionEffort
+ID : int
+UserID : int
+LevelID : int
+IsSuccessful : bool
+Time : timestamp
+Create()
+CheckSuccessful() : bool
TypeRecord
+ID : int
+Name : string
+Read()
User
+ID : int
+Nickname : string
+IsAdmin : bool
+Read()
RfLevelsBrief
+MetaInfo : MetaInfo
+Data : LevelsBrief
+Create()
+Read() : interface
+ReadAll() : interface
+Update()
+Delete()
RfLevelsData
+MetaInfo : MetaInfo
+Data : LevelsData
+Create()
+Read() : interface
+Update()
+Delete()
RfSolutionEffort
+MetaInfo : MetaInfo
+Data : SolutionEffort
+Create()
+CheckSuccessful() : interface
RfTypeRecord
+MetaInfo : MetaInfo
+Data : TypeRecord
+Read() : interface
RfUser
+MetaInfo : MetaInfo
+Data : User
+Read() : interface
IReadableAll
+ReadAll() : interface
ICreatable
+Create()
IUpdatable
+Update()
IDeletable
+Delete()
IReadable
+Read() : interface
ICheckableSuccessful
+CheckSuccessful() : interface
ResponseFrame
+StatusCode: int
+StatusStr : string
+Message : string
+Data : interface
Interface

Фрагмент программного кода, иллюстрирующий работу с интерфейсами приведен в листинге 3.

Листинг 3 — фрагмент программного кода микросервиса взаимодействия с БД.

```
var reqFrame EncapsulatedMetaInfo
// преобразование тела HTTP-запроса в объект класса EncapsulatedMetaInfo:
err = json.Unmarshal(reqBody, &reqFrame)
if err != nil {
   /* обработка ошибок */
}

var data interface{} // создание объекта базового класса
// выбор класса, к которому произойдет обращение
if reqFrame.MetaInfo.ObjType == "levels_brief" {
   data = &RfLevelsBrief{} // присвоение ссылки на пустой экземпляр класса
} else if reqFrame.MetaInfo.ObjType == "levels_data" {
   data = &RfLevelsData{}
} else if
   /* ... */
} else {
   panic("Unknown Obj Type")
}

// преобразование тела HTTP-запроса в объект выбранного класса
err = json.Unmarshal(reqBody, data)
if err != nil {
   /* обработка ошибок */
}
if reqFrame.MetaInfo.Action == "create" {
   data.(ICreatable).Create() // обращение к методу класса через интерфейс
} else if reqFrame.MetaInfo.Action == "read" {
   // обращение к методу класса через интерфейс и запись данных в поле типа interface
   response.Data = data.(IReadable).Read()
} else if
   /* ... */
} else {
   panic("Unknown Action")
}
```

3.2 Синтезатор

Для симуляции (получения временных диаграмм работы) и синтеза (получения списка электрических соединений) устройств, описанных на языке Verilog используют специальное программное обеспечение, которое может называться «симулятором», «синтезатором» или «компилятором» (последний термин менее точен, но более интуитивно понятен).

Одним из таких синтезаторов является IcarusVerilog. Достоинствами данного программного решения являются:

- малый размер исполняемого файла;
- наличие консольного режима работы (удобно вызывать из программного кода через библиотеки для работы с операционной системой);
- распространение по свободной лицензии (GNU GPL).

В силу перечисленных выше свойств, IcarusVerilog был выбран в качестве синтезатора, используемого в данной работе.

Выбранный синтезатор был использован в составе микросервиса, осуществляющего управление файлами и взаимодействие с сетью.

Диаграмма компоновки микросервиса показана на рисунке 8.

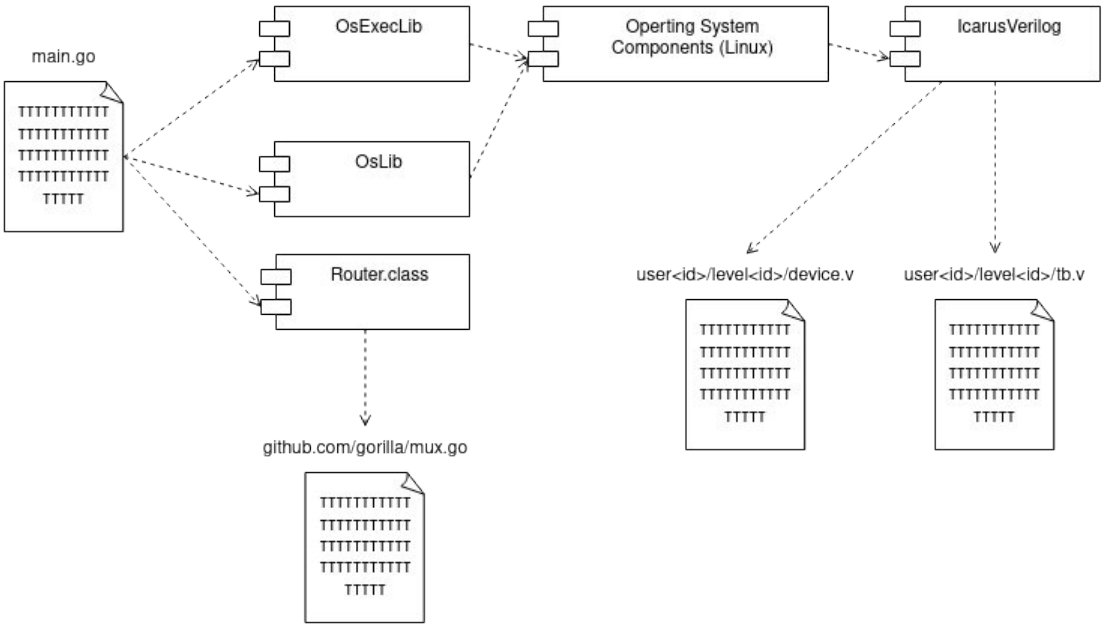


Рисунок 8 — диаграмма компоновки микросервиса «Синтезатор»

Обработка каждого задания осуществляется в несколько этапов, за каждый из которых отвечает свой программный компонент:

- получение http-запроса на симуляцию устройства (класс Router);
- сохранение полученных исходных кодов устрйоства и теста в файловой системе (OsLib, наличие user_id и level_id позволяет значительно снизить риск коллизии файлов);
- получение временной диаграммы работы устройства (в формате *.vcd) с помощью IcarusVerilog;
- отправка http-ответа, содержащего код временной диаграммы.

3.3 Преобразователь формата временных диаграмм и генератор wavedrom-диаграмм

Изначально, микросервис «Синтезатор» в ходе тестирования работы устройства формирует временную диаграмму в формате *.vcd (Приложение В). Данный формат крайне неудобен, как для анализа в сравнении с эталонной временной диаграммой, так и для генерации графического представления временной диаграммы в рамках веб-приложения.

Для преобразования временных диаграмм к более удобному для дальнейшей обработки формату был реализован микросервис «Преобразователь формата временных диаграмм». Его исходный код написан на Python с применением библиотеки PyDigitalWaveTools. Данная библиотека преобразует временную диаграмму в формате *.vcd в формат JSON-PyDigitalWaveTools согласно алгоритму, заложенному автором библиотеки. Диаграмма Джексона, описывающая этот формат представлена на рисунке 9.

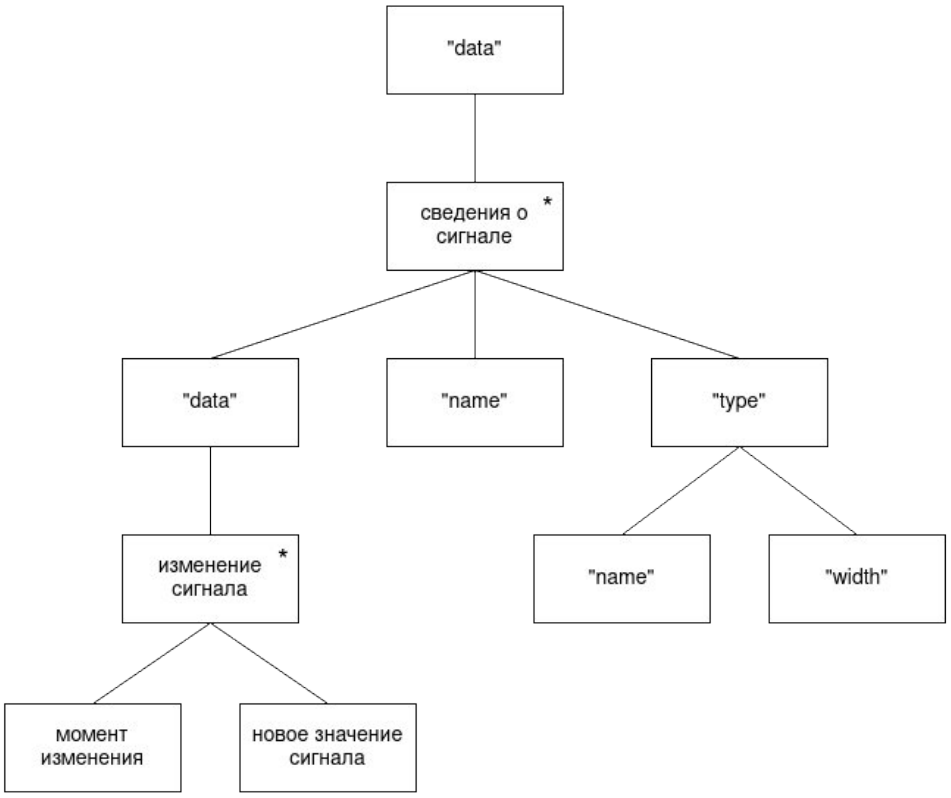


Рисунок 9 — формат временных диаграмм в PyDigitalWaveTools

В нем поле data.name — имя сигнала, data.type.name — название типа сигнала (комбинационный или регистровый), data.type.width — разрядность сигнала. «Момент изменения» — количество элементарных отрезков времени (их размер определяется в момент написания теста для устройства) от начала отсчет до изменения сигнала.

Пример описания сигнала в этом формате приведен в листинге 4.

Листинг 4 — пример описания временной диаграммы в PyDigitalWaveTools

```
{
  "data": [
    {
      "data": [
        [0, "b0" ],
        [100, "b1"]
      ],
      "name": "Sum",
      "type": {
```

```
            "name": "wire",
            "width": 4
          }
        },
        /* ... */
    ]
}
```

Формат PyDigitalWaveTools намного более удобен для сравнения с эталонной временной диаграммой (в том же формате) и анализа несоответствий, однако алгоритм визуализации для этого формата пришлось бы реализовать самостоятельно.

Вместо этого было решено реализовать еще один преобразователь формата («Генератор wavedrom-диаграмм»), который преобразовал бы временные диаграммы из формата PyDigitalWaveTools в формат движка Wavedrom. Данный движок позволяет визуализировать временные диаграммы посредством http-запроса, содержащего описание сигнала, к специальному интернет-сервису.

Описание формата для движка Wavedrom в нотации Джексона приведено на Рисунке 10.

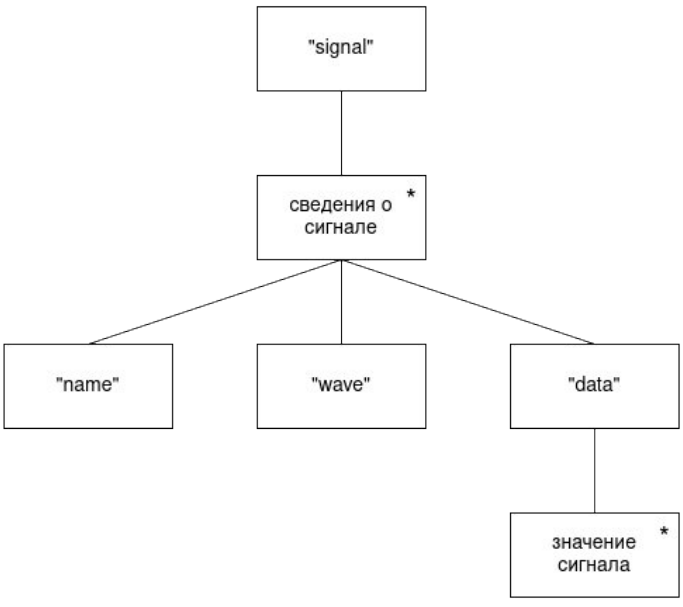


Рисунок 10 — формат временных диаграмм для движка Wavedrom

Поля структуры имеют значение, описанное ниже:

- signal — массив всех сигналов временной диаграммы;
- name — имя сигнала;
- wave — форма сигнала (для каждого такта может иметь значения: «0», «1», «x», «z», «.» — сохранить предыдущее, «|» — разрыв, «=» — обратиться к очередному элементу «data»);
- data — массив, содержащий строковые значения сигнала (можно, например, отобразить большое число для многоразрядной шины).

Пример описания временной диаграммы в формате движка Wavedrom приведен в листинге 5.

Листинг 5 — описание временной диаграммы в формате движка Wavedrom

```
{signal: [
  {name: 'clk', wave: 'p.....|...'},
  {name: 'dat', wave: 'x.345x|=.x', data: ['0x16', '0xAA', '0x07', '0x11']},
  {name: 'req', wave: '0.1..0|1.0'},
  {},
  {name: 'ack', wave: 'z.....|01.'}
]}
```

Визуализация данной временной диаграммы приведена на рисунке 11.

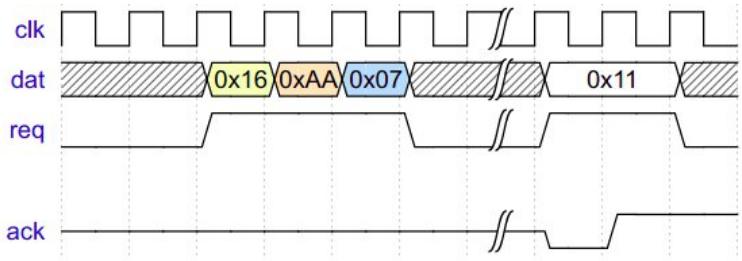


Рисунок 11 — визуализация временной диаграммы

Основной функцией «Генератора wavedrom-диаграмм» является функция parseValues, программный код которой приведен в листинге 6.

Листинг 6 — программный код функции parseValues

```
func (vcd_frame VCD_Struct) parseValues(end_scale int, width_scale int) (map[string]string, map[string][]string) {
   // значения сигналов, e.g.: {"a": ["0x10","0x35","0xA1"], "b": ["0x03","0x0F"]}
   var parsedData = map[string][]string{}
   // форма сигналов, e.g.: {"a": "1...0.....1..", "b": "0....=....=.."}
   var parsedWaves = map[string]string{}

   // отсортированные моменты изменения всех сигналов
   timings := vcd_frame.getSortedTimings()
   tick_amount := findGCD(timings) // НОД момента изменения сигнала
   end_time := timings[len(timings)-1] + tick_amount*end_scale

   for i := 0; i < end_time/tick_amount; i++ { // проход по всем моментам дискретизации
      for _, single_signal := range vcd_frame.Signal { // проход по всем сигналам диаграммы
         fl_change := false // признак изменения сигнала в этом моменте времени
         name := single_signal.Name
         fl_single_wire := true // признак однобитного сигнала
         if single_signal.Type.Width > 1 {
            name += "[0:" + strconv.Itoa(single_signal.Type.Width-1) + "]"
            fl_single_wire = false
         }
         for _, data_value := range single_signal.Data { // проход по всем точкам дискретизации
            // изменился ли сигнал в рассматриваемой точки дискретизации?
            if int(math.Round(data_value[0].(float64))) == i*tick_amount {
               if fl_single_wire {
                  parsedWaves[name] += data_value[1].(string)
               } else {
                  parsedData[name] = append(parsedData[name], data_value[1].(string))
                  parsedWaves[name] += "="
               }
               fl_change = true
               break
            }
         }
         if !fl_change {
            parsedWaves[name] += "."
         }
         // масштабирование ширины сигнала на диаграмме
```

```
            for j := 1; j < (vcd_frame.getMaxValueWidth()*width_scale)/2; j++ {
                parsedWaves[name] += "."
            }
        }
    }
    return parsedWaves, parsedData
}
```

3.4 Микросервис анализа статистики

Для работы со статистикой прохождения заданий было решено реализовать отдельный микросервис, структура которого была бы аналогична структуре микросервиса взаимодействия с БД (диаграмма компоновки представлена на рисунке 12).

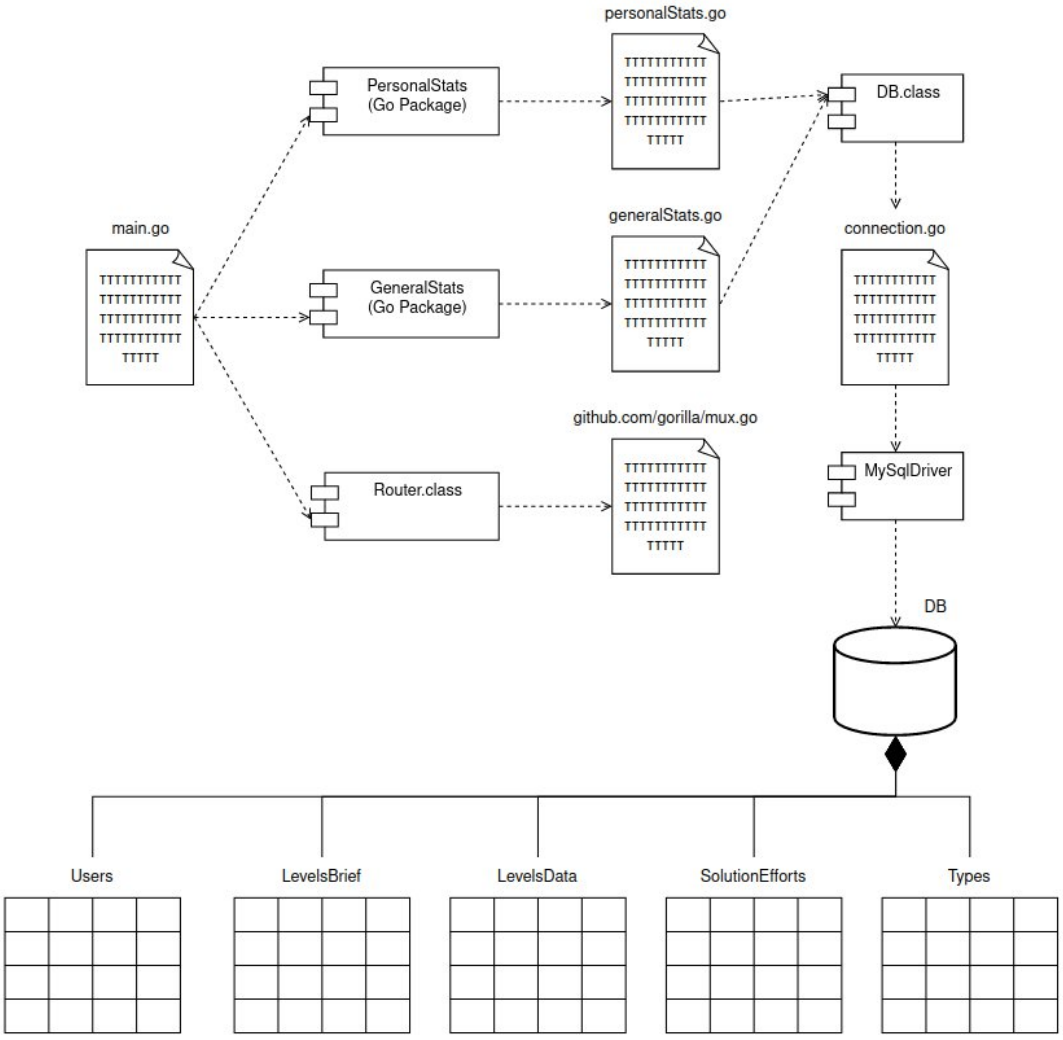


Рисунок 12 — диаграмма компоновки микросервиса анализа статистики

Микросервис реализует, два вида запросов — запросы, получающие персональную статистику пользователя (требуется user_id), и запросы, получающие обобщенную статистику для всех пользователей.

Полученные данные возвращаются в формате JSON.

Запросы, получающие персональную статистику:

- прогресс по курсу — количество полученных баллов и решенных заданий, статус прохождения курса (<80% баллов от максимума — «not_passed», 80-90% — «passed», >90% — «awesome»);
- статус прохождения каждого задания (информация об уровне из таблицы LevelsBrief и признак «is_solved» для каждого задания);
- среднее число неправильных попыток на задание;
- общее число попыток и правильных решений за последний месяц;
- Дата решения первого и последнего решенного задания.

Запросы, получающие обобщенную статистику:

- количество верных решений для каждого задания;
- среднее число ошибок в каждом задании;
- распределение количества пройденных заданий в зависимости от числа их решений (абсцисса — количество решений n, ордината — число заданий которы решили n раз);
- среднее число предоставленных пользователями решений по месяцам;
- топ 10 активных (по количеству решений) пользователей за последний месяц.

3.5 Анализатор решений

Анализатор решений представляет собой микросервис, задачей которого является проверка пользовательских ответов, а также предоставление подсказок в случае допущения пользователем ошибки.

Для универсализации алгоритма обработки решений используется полиморфизм (рисунок 13).

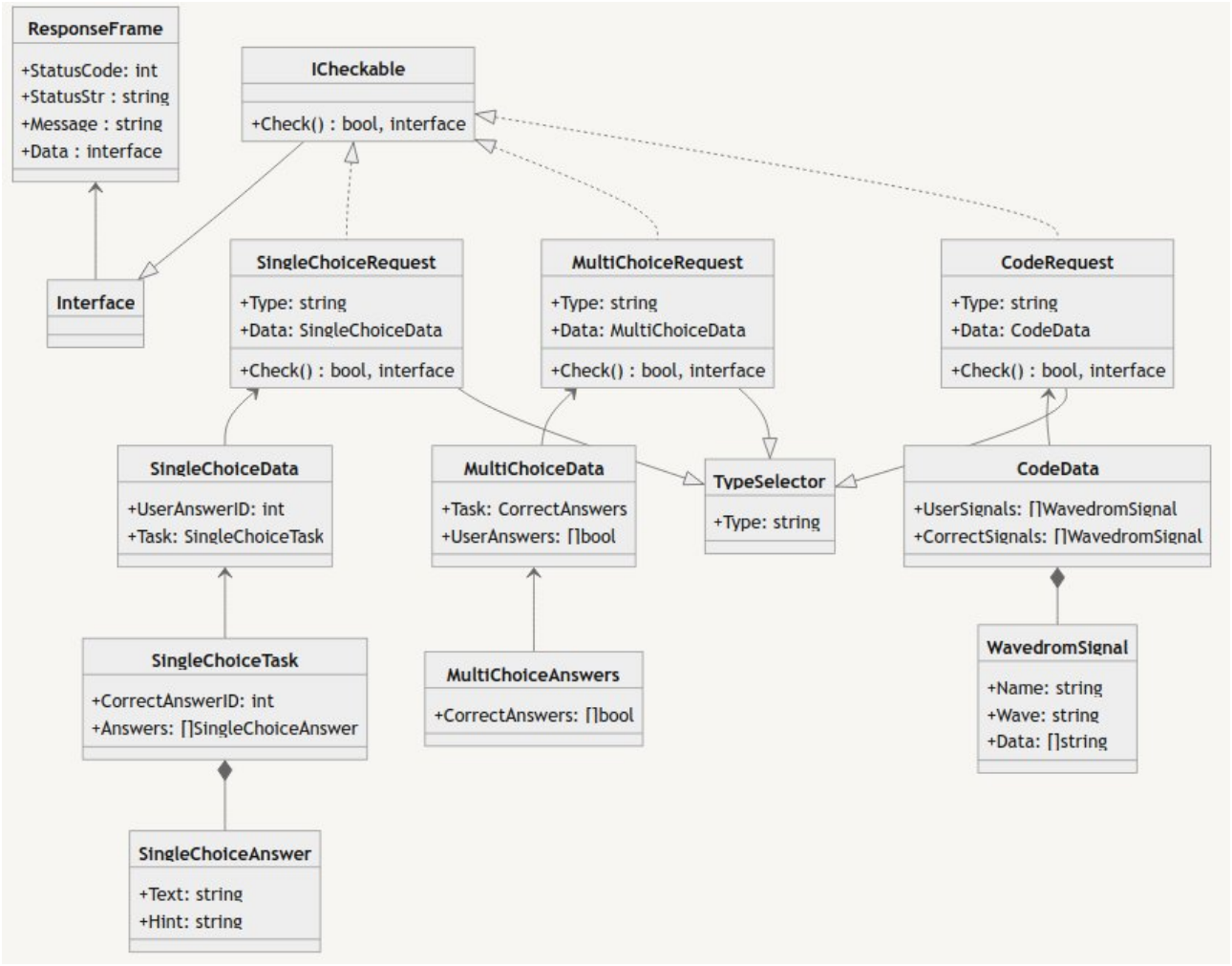


Рисунок 13 — диаграмма классов анализатора решений

Анализатор способен работать с тремя типами заданий:

- тест с выбором одного варианта ответа — анализатор сопровождает неправильный ответ текстовой подсказкой;
- тест с выбором нескольких вариантов ответа — анализатор сопровождает неправильный ответ информацией о наличии/отстутствии ложноположительных/ложноотрицательных вариантов;
- задание на описание устройства на языке Verilog — анализатор приводит список отсутствующих выходных сигналов и список сигналов, поведение которых не соответствует ожидаемому.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ Х – диаграммы последовательности проверки кода и редактирования задания, device.v, tb.v, *.vcd, ...

Приложение А – диаграммы последовательности

Приложение Б — device.v, tb.v

Приложение В — *.vcd