**ТИТУЛ**

**ЗАДАНИЕ**

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc1)

[1 Проектирование архитектуры и слоя бизнес-логики 6](#_Toc2)

[2 Проектирование и взаимодействие с БД 7](#_Toc3)

[2.1 Проектирование даталогической схемы БД 7](#_Toc4)

[2.2 Форматы хранения заданий и ответов 9](#_Toc5)

[2.3 Подсистема работы с содержанием курса 12](#_Toc6)

[3 Проектирование микросервисов 16](#_Toc7)

[3.Х Преобразователь формата временных диаграмм и генератор wavedrom-диаграмм 16](#_Toc8)

[3.Х Синтезатор 21](#_Toc9)

[ЗАКЛЮЧЕНИЕ 22](#_Toc10)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc11)

[ПРИЛОЖЕНИЕ Х – src.v, tb.v, \*.vcd, ... 24](#_Toc12)

# **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 — ...

# **1 Проектирование архитектуры и слоя бизнес-логики**

# **2 Проектирование и взаимодействие с БД**

## **2.1 Проектирование 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 текcтовых 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 Х.

Ниже представлено подробное описание приведенных таблиц и их полей.

Таблица Users (пользователи):

* id — первичный ключ;
* nickname — псевдоним;
* is\_admin — признак администратора.



Рисунок Х — 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 Форматы хранения 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 Х.



Рисунок Х — формат описания задания с выбором одного ответа

Пример описания задания с одним ответом приведен в листинге Х.

Листинг Х — 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 Х.

Листинг Х — 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 Х), а в поле LevelsData.answer — описание временной диаграммы корректно описанного устройства в формате wavedrom (см. раздел «Генератор wavedrom-диаграмм»).

## 2.3 Подсистема работы с содержанием курса

Для реализации CRUD-операций с данным, хранящимися в БД, была реализована отдельная подсистема работы с содержимым курса.

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 Х).

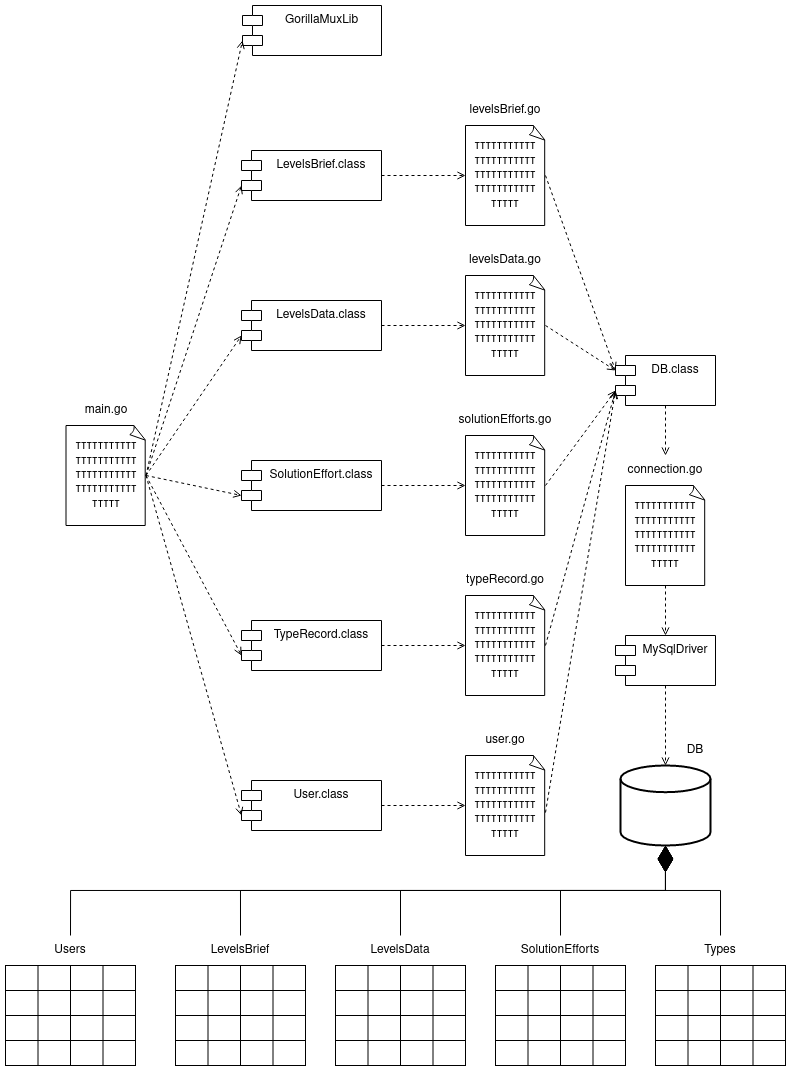


Рисунок Х — диаграмма компоновки подсистемы работы с содержанием курса

На рисунке Х (сделать подпись!) представлена диаграмма классов описываемой подсистемы.

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

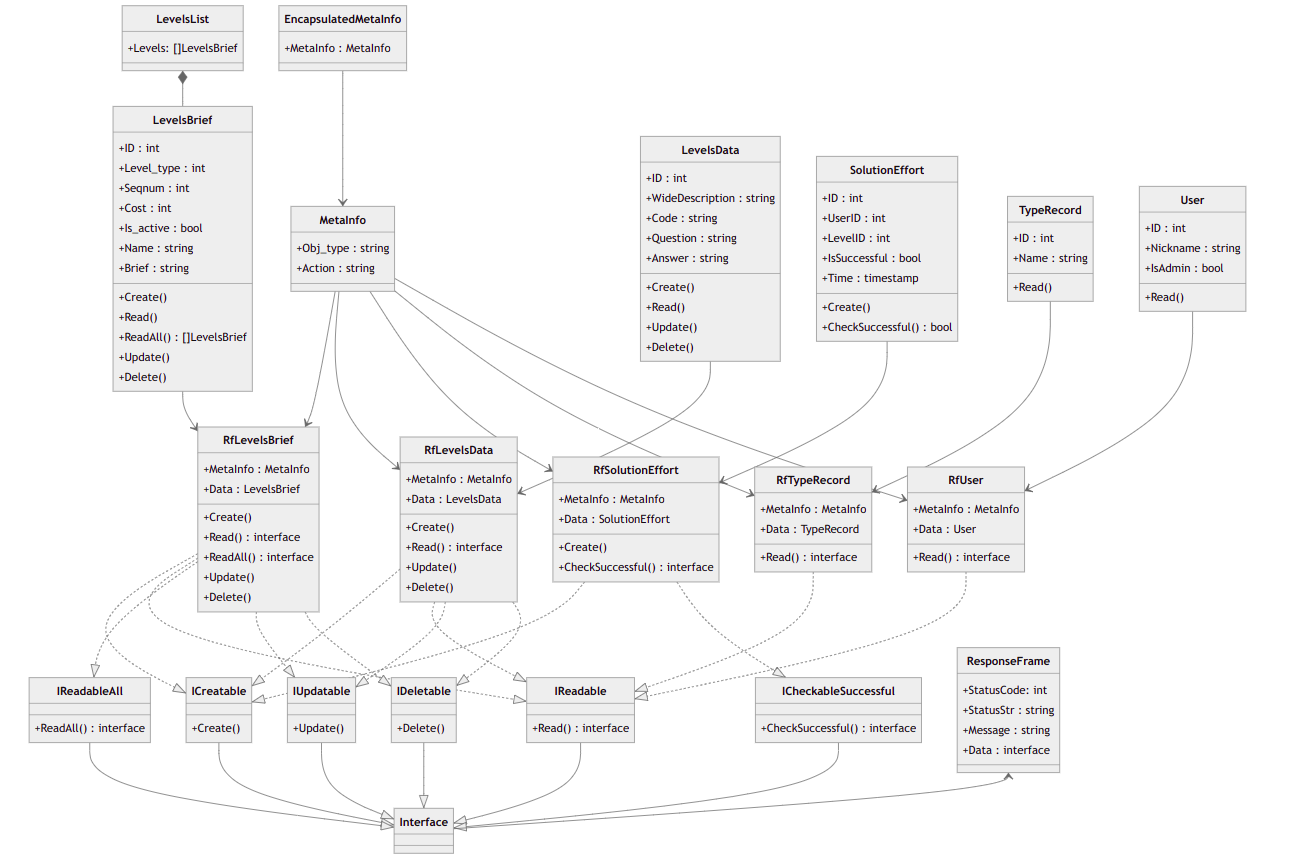
Класс MetaInfo содержит поля ObjType (сущность, надо которой выполняется операция) и Action (тип операции).

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.



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

|  |
| --- |
| 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 Проектирование микросервисов**

## 3.Х Преобразователь формата временных диаграмм и генератор wavedrom-диаграмм

Изначально, микросервис «Синтезатор» в ходе тестирования работы устройства формирует временную диаграмму в формате \*.vcd (Приложение Х). 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 Х.



Рисунок Х — 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 Х.

Листинг Х — 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 движка Wavedrom в нотации Джексона приведено на Рисунке Х.



Рисунок Х — 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 Х.

Листинг Х — 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 Х.



Рисунок Х — визуализация временной диаграммы

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

Листинг Х — 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 изменения сигнала  // 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.Х Синтезатор

# **ЗАКЛЮЧЕНИЕ**

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

# **ПРИЛОЖЕНИЕ Х – src.v, tb.v, \*.vcd, ...**