```
package main

import (
	"encoding/json"
	"fmt"
	"io/ioutil"
	"net/http"
	"os/exec"
	"runtime"

	"github.com/gorilla/mux"
)

// позволяет считать тип проверяемого задания из тела http-запроса
type TypeSelector struct {
	Type string `json:"type"`
}

// описывает тело запроса на проверку теста с одним вариантом ответа
type SingleChoiceTestRequest struct {
	Type string `json:"type"`
	Data struct {
		UserAnswerID int `json:"user_answer_id"`
		Task         struct {
			CorrectAnswerID int `json:"correct_answer_id"`
			Answers         []struct {
				Text string `json:"text"`
				Hint string `json:"hint"`
			} `json:"answers"`
		} `json:"task"`
	} `json:"data"`
}

// описывает тело запроса на проверку теста с несколькими вариантами ответа
type MultiChoiceTestRequest struct {
	Type string `json:"type"`
	Data struct {
		UserAnswers []bool `json:"user_answers"`
		Task        struct {
			CorrectAnswers []bool `json:"correct_answers"`
		} `json:"task"`
	} `json:"data"`
}
```

```
// описывает поведение цифрового сигнала во времени
type WavedromSignal struct {
     Name string   `json:"name"`
     Wave string   `json:"wave"`
     Data []string `json:"data"`
}

// описывает тело запроса на проверку задания на написание
Verilog-кода
// (сверяться будут временные диаграммы)
type CodeRequest struct {
     Type string `json:"type"`
     Data struct {
          UserSignals    []WavedromSignal `json:"user_signals"`
          CorrectSignals []WavedromSignal `json:"correct_signals"`
     } `json:"data"`
}

// === форматы ответов  ===

type SingleChoiceTestResult struct {
     Hint string `json:"hint"`
}

type MultiChoiceTestResult struct {
     FalsePositive bool `json:"false_positive"`
     FalseNegative bool `json:"false_negative"`
}

type CodeResult struct {
     MissingSignals     []string `json:"missing_signals"`
     MismatchingSignals []string `json:"mismatching_signals"`
}



// формат полного ответа
type ResponseFrame struct {
     StatusStr  string      `json:"status_str"`
     StatusCode int         `json:"status_code"`
     Message    string      `json:"message"`
     IsCorrect  bool        `json:"is_correct"`
            //  SingleChoiceTestResult,  MultiChoiceTestResult  или
CodeResult
     Data       interface{} `json:"data"`
```

```
}

// интерфейс для проверки заданий
type ICheckable interface {
     Check() (bool, interface{})
}

// метод для проверки задания с одним вариантом ответа
func (v SingleChoiceTestRequest) Check() (bool, interface{}) {
     var fl bool
     fl = (v.Data.UserAnswerID == v.Data.Task.CorrectAnswerID)
     var res SingleChoiceTestResult
     res.Hint = v.Data.Task.Answers[v.Data.UserAnswerID].Hint
     return fl, res
}

// метод для проверки задания с несколькими вариантами ответа
func (v MultiChoiceTestRequest) Check() (bool, interface{}) {
     var fl, false_positive, false_negative bool
     if len(v.Data.UserAnswers) != len(v.Data.Task.CorrectAnswers)
{
          panic("Answers arrays size mismatch")
     }

     fl = true
     false_positive = false
     false_negative = false
     for i, _ := range v.Data.UserAnswers {
          if               !v.Data.UserAnswers[i]               &&
v.Data.Task.CorrectAnswers[i] {
               fl = false
               false_negative = true
          }          else          if          v.Data.UserAnswers[i]
&& !v.Data.Task.CorrectAnswers[i] {
               fl = false
               false_positive = true
          }
     }

     var res MultiChoiceTestResult
     res.FalsePositive = false_positive
     res.FalseNegative = false_negative
     return fl, res
}

// метод для проверки задания на написание Verilog-кода (сверки
```

```
временных диаграмм)
func (v CodeRequest) Check() (bool, interface{}) {
     var fl bool
     var res CodeResult
     fl = true

             // для каждого сигнала из эталонных
     for _, signal_correct := range v.Data.CorrectSignals {
          var entry_fl bool
          entry_fl = false
          var signal_user_buf WavedromSignal
                          // найти соответствующий сигнал в врем.
диаграмме пользователя
          for _, signal_user := range v.Data.UserSignals {
               if signal_correct.Name == signal_user.Name {
                    entry_fl = true
                    signal_user_buf = signal_user
               }
          }
          if entry_fl { // если найден, то проверить соответствие
значений
               var equality_fl bool
               equality_fl = true
               if signal_correct.Wave != signal_user_buf.Wave {
                    equality_fl = false
               }
               if            len(signal_correct.Data)            !=
len(signal_user_buf.Data) {
                    equality_fl = false
               } else {
                    for i, _ := range signal_correct.Data {
                         if        signal_correct.Data[i]        !=
signal_user_buf.Data[i] {
                              equality_fl = false
                              break
                         }
                    }
               }
               if !equality_fl {
                                                    // если
несоответствуют - добавить в список несоответствий
                    res.MismatchingSignals =
append(res.MismatchingSignals,

signal_correct.Name)
                    fl = false
```

```
                }
          } else {
                                        // если сигнал не найден – добавить в список ненайденных
                res.MissingSignals = append(res.MissingSignals, signal_correct.Name)
                fl = false
          }
     }

     return fl, res
}

// обработка http-запросов
func handler(w http.ResponseWriter, req *http.Request) {
     var response ResponseFrame

             // обработка ошибок
     defer func() {
          if panicInfo := recover(); panicInfo != nil {
                var response ResponseFrame
                response.StatusStr = "error"
                response.StatusCode = 400
                response.Message = fmt.Sprintf("Top-level panic: %v", panicInfo)
                w.WriteHeader(response.StatusCode)
                json.NewEncoder(w).Encode(response)
          }
     }()

     reqBody, _ := ioutil.ReadAll(req.Body) // считать тело запроса
     var type_selector TypeSelector

     err := json.Unmarshal(reqBody, &type_selector) // определение типа задания

     if err != nil {
          defer func() {
                if r := recover(); r != nil {
                     response.StatusStr = "error"
                     response.StatusCode = 400
                     response.Message = err.Error()
                     w.WriteHeader(response.StatusCode)
                     json.NewEncoder(w).Encode(response)
                }
```

```
          }()
          panic("request-JSON parsing error")
     }

            // разьор тела запроса в зависимости от типа задания
     var data interface{}
     if type_selector.Type == "singlechoice_test" {
          data = &SingleChoiceTestRequest{}
     } else if type_selector.Type == "multichoice_test" {
          data = &MultiChoiceTestRequest{}
     } else if type_selector.Type == "program" {
          data = &CodeRequest{}
     } else {
          panic("Unknown task type")
     }

     err = json.Unmarshal(reqBody, data)

     if err != nil {
          defer func() {
               if r := recover(); r != nil {
                    response.StatusStr = "error"
                    response.StatusCode = 400
                    response.Message = err.Error()
                    w.WriteHeader(response.StatusCode)
                    json.NewEncoder(w).Encode(response)
               }
          }()
          panic("data-JSON parsing error")
     }

             // проверить задание
     response.IsCorrect, response.Data = data.(ICheckable).Check()

     response.StatusStr = "ok"
     response.StatusCode = 200
     response.Message = "checked"
     w.WriteHeader(response.StatusCode)
     json.NewEncoder(w).Encode(response)
}

// инициализация сервера
func main() {
     if runtime.GOOS == "windows" {
          fmt.Println("Can't Execute this on a windows machine")
     } else {
```

```
            fmt.Println("Server start")

            r := mux.NewRouter().StrictSlash(true)
            r.HandleFunc("/check", handler).Methods("POST")
            http.ListenAndServe(":8083", r)

            fmt.Println("Server stop")
        }
}
```