Main.go:

```go
package main

import (
    "log"
    "net/http"

    "idz1_opt/internal/server"
)

func main() {
    router := server.NewRouter()
    log.Println("Сервер запущен на http://localhost:8080")
    log.Println("Static files served from:", "static")
    log.Fatal(http.ListenAndServe(":8080", router))
}
```

Dichotomy.go

```go
package optimizer

import "errors"

// Iter — одна итерация метода дихотомии
type Iter struct {
    K     int     `json:"k"`
    A     float64 `json:"a"`
    B     float64 `json:"b"`
    XMid  float64 `json:"xmid"`
    FXMid float64 `json:"fxmid"`
    Len   float64 `json:"len"`
}
```

```go
// ErrStopped — специальная ошибка для принудительной остановки
var ErrStopped = errors.New("dichotomy: stopped by callback")

// Dichotomy — реализация метода дихотомии
// onIter вызывается после каждой итерации; если вернёт ErrStopped — алгоритм прерывается.
func Dichotomy(
    f Func,
    a, b, eps, delta float64,
    maxIter int,
    onIter func(Iter) error,
) (Iter, error) {
    var last Iter

    for k := 1; k <= maxIter && (b-a)/2 > eps; k++ {
        x1 := (a + b - delta) / 2
        x2 := (a + b + delta) / 2

        fx1, err1 := f.Eval(x1)
        fx2, err2 := f.Eval(x2)
        if err1 != nil || err2 != nil {
            if err1 != nil {
                return last, err1
            }
            return last, err2
        }

        if fx1 <= fx2 {
            b = x2
        } else {
            a = x1
        }

        mid := (a + b) / 2
```

```
        fmid, _ := f.Eval(mid)

        last = Iter{
            K:    k,
            A:    a,
            B:    b,
            XMid:  mid,
            FXMid: fmid,
            Len:   b - a,
        }

        if onIter != nil {
            if err := onIter(last); err != nil {
                if errors.Is(err, ErrStopped) {
                    return last, ErrStopped
                }
                return last, err
            }
        }
    }

    return last, nil
}
```

Function.go

```
package optimizer

import (
    "fmt"
    "math"
    "strconv"
    "strings"
```

```
    "github.com/Knetic/govaluate"
)

// Func — интерфейс для абстрактной функции f(x)
type Func interface {
    Eval(x float64) (float64, error)
}

// evalFunc — реализация Func на основе govaluate
type evalFunc struct {
    expr   *govaluate.EvaluableExpression
    params map[string]interface{}
}

// NewEvalFunc создаёт вычислимую функцию по строке f(x)
func NewEvalFunc(expr string) (Func, error) {
    funcs := map[string]govaluate.ExpressionFunction{
        "sin": func(args ...interface{}) (interface{}, error) { return math.Sin(toFloat(args[0])), nil },
        "cos": func(args ...interface{}) (interface{}, error) { return math.Cos(toFloat(args[0])), nil },
        "tan": func(args ...interface{}) (interface{}, error) { return math.Tan(toFloat(args[0])), nil },
        "exp": func(args ...interface{}) (interface{}, error) { return math.Exp(toFloat(args[0])), nil },
        "log": func(args ...interface{}) (interface{}, error) { return math.Log(toFloat(args[0])), nil },
        "sqrt": func(args ...interface{}) (interface{}, error) {
            return math.Sqrt(toFloat(args[0])), nil
        },
        "abs": func(args ...interface{}) (interface{}, error) {
            return math.Abs(toFloat(args[0])), nil
        },
        "pow": func(args ...interface{}) (interface{}, error) {
            return math.Pow(toFloat(args[0]), toFloat(args[1])), nil
        },
    }
```

```
    // нормализуем запятые в десятичной записи
    expr = strings.ReplaceAll(expr, ",", ".")

    parsed, err := govaluate.NewEvaluableExpressionWithFunctions(expr, funcs)
    if err != nil {
        return nil, err
    }

    return &evalFunc{
        expr:   parsed,
        params: map[string]interface{}{"x": 0.0},
    }, nil
}

func (f *evalFunc) Eval(x float64) (float64, error) {
    f.params["x"] = x
    v, err := f.expr.Evaluate(f.params)
    if err != nil {
        return math.NaN(), err
    }

    switch t := v.(type) {
    case float64:
        return t, nil
    case int:
        return float64(t), nil
    case int64:
        return float64(t), nil
    case string:
        parsed, err := strconv.ParseFloat(t, 64)
        if err != nil {
            return math.NaN(), err
        }
        return parsed, nil
```

```
    default:
        return math.NaN(), fmt.Errorf("выражение не вернуло число: %T", v)
    }
}

func toFloat(v interface{}) float64 {
    switch t := v.(type) {
    case float64:
        return t
    case int:
        return float64(t)
    case int64:
        return float64(t)
    case string:
        f, _ := strconv.ParseFloat(t, 64)
        return f
    default:
        return math.NaN()
    }
}
```

Handlers.go

```
package server

import (
    "context"
    "encoding/csv"
    "encoding/json"
    "fmt"
    "idz1_opt/internal/optimizer"
    "idz1_opt/internal/sse"
```

```
    "math"
    "net/http"
    "strconv"
    "time"

    "github.com/google/uuid"
)

// StartRun запускает новый процесс минимизации
func StartRun(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "только POST", http.StatusMethodNotAllowed)
        return
    }

    var p RunParams
    if err := json.NewDecoder(r.Body).Decode(&p); err != nil {
        http.Error(w, "ошибка JSON: "+err.Error(), http.StatusBadRequest)
        return
    }

    if p.MaxIter <= 0 {
        p.MaxIter = 200
    }
    if p.Eps <= 0 {
        p.Eps = 1e-5
    }
    if p.Delta <= 0 {
        p.Delta = p.Eps / 2
    }
    if !(p.A < p.B) {
        http.Error(w, "требуется a < b", http.StatusBadRequest)
        return
    }
```

```
f, err := optimizer.NewEvalFunc(p.Func)
if err != nil {
    http.Error(w, "ошибка в выражении функции: "+err.Error(), http.StatusBadRequest)
    return
}

// предварительно считаем значения функции для графика
const n = 400
xs := make([]float64, n)
ys := make([]float64, n)
h := (p.B - p.A) / float64(n-1)
for i := 0; i < n; i++ {
    x := p.A + float64(i)*h
    y, err := f.Eval(x)
    if err != nil || math.IsNaN(y) || math.IsInf(y, 0) {
        y = math.NaN()
    }
    xs[i], ys[i] = x, y
}

id := uuid.NewString()
ctx, cancel := context.WithCancel(context.Background())
rs := &RunState{
    ID:        id,
    Params:    p,
    CreatedAt: time.Now(),
    Cancel:    cancel,
}
saveRun(rs)

// асинхронный запуск оптимизации
go func() {
    // стартовое событие
```

```
    startMsg, _ := json.Marshal(map[string]any{
        "type": "start",
        "id":   id,
    })
    sse.Publish(id, string(startMsg))

    onIter := func(it optimizer.Iter) error {
        select {
        case <-ctx.Done():
            return optimizer.ErrStopped
        default:
        }

        rs.LastIter = it
        rs.Iters = append(rs.Iters, it)

        payload := map[string]any{
            "type": "iter",
            "iter": it,
        }
        msg, _ := json.Marshal(payload)
        sse.Publish(id, string(msg))
        return nil
    }

    last, err := optimizer.Dichotomy(
        f,
        p.A, p.B,
        p.Eps, p.Delta,
        p.MaxIter,
        onIter,
    )

    if err != nil {
```

```
        if err == optimizer.ErrStopped || err == context.Canceled {
            stopMsg, _ := json.Marshal(map[string]any{
                "type": "stopped",
            })
            sse.Publish(id, string(stopMsg))
            return
        }

        rs.Err = "ошибка при вычислении: " + err.Error()
        errMsg, _ := json.Marshal(map[string]any{
            "type": "error",
            "err":  rs.Err,
        })
        sse.Publish(id, string(errMsg))
        return
    }

    rs.Done = true
    rs.LastIter = last

    doneMsg, _ := json.Marshal(map[string]any{
        "type": "done",
        "x":    last.XMid,
        "fx":   last.FXMid,
    })
    sse.Publish(id, string(doneMsg))
}()

resp := map[string]any{
    "id": id,
    "xs": xs,
    "ys": ys,
}
w.Header().Set("Content-Type", "application/json")
```

```
    _ = json.NewEncoder(w).Encode(resp)
}

// StopRun — прерывание процесса минимизации
func StopRun(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "только POST", http.StatusMethodNotAllowed)
        return
    }
    id := r.URL.Query().Get("id")
    if id == "" {
        http.Error(w, "требуется id", http.StatusBadRequest)
        return
    }

    rs := getRun(id)
    if rs == nil {
        http.Error(w, "неизвестный id", http.StatusNotFound)
        return
    }

    if rs.Cancel != nil {
        rs.Cancel()
    }

    w.WriteHeader(http.StatusNoContent)
}

// ExportCSV — экспорт итераций в CSV
func ExportCSV(w http.ResponseWriter, r *http.Request) {
    id := r.URL.Query().Get("id")
    if id == "" {
        http.Error(w, "требуется id", http.StatusBadRequest)
        return
```

```
    }

    rs := getRun(id)
    if rs == nil {
        http.Error(w, "неизвестный id", http.StatusNotFound)
        return
    }

    w.Header().Set("Content-Type", "text/csv; charset=utf-8")
    w.Header().Set("Content-Disposition", "attachment; filename=iterations_"+id+".csv")

    cw := csv.NewWriter(w)
    defer cw.Flush()

    _ = cw.Write([]string{"k", "a", "b", "mid", "f(mid)", "b-a"})

    for _, it := range rs.Iters {
        _ = cw.Write([]string{
            strconv.Itoa(it.K),
            fmtFloat(it.A),
            fmtFloat(it.B),
            fmtFloat(it.XMid),
            fmtFloat(it.FXMid),
            fmtFloat(it.Len),
        })
    }
}

func fmtFloat(v float64) string {
    return strconv.FormatFloat(v, 'g', 16, 64)
}

// Stream — SSE-стрим итераций
func Stream(w http.ResponseWriter, r *http.Request) {
```

```
    id := r.URL.Query().Get("id")
    if id == "" {
        http.Error(w, "требуется id", http.StatusBadRequest)
        return
    }

    w.Header().Set("Content-Type", "text/event-stream")
    w.Header().Set("Cache-Control", "no-cache")
    w.Header().Set("Connection", "keep-alive")

    flusher, ok := w.(http.Flusher)
    if !ok {
        http.Error(w, "streaming unsupported", http.StatusInternalServerError)
        return
    }

    ch, cancel := sse.Subscribe(id)
    defer cancel()

    ctx := r.Context()

    for {
        select {
        case <-ctx.Done():
            return
        case msg := <-ch:
            fmt.Fprintf(w, "event: msg\n")
            fmt.Fprintf(w, "data: %s\n\n", msg)
            flusher.Flush()
        }
    }
}
```

Router.go

```
package server

import "net/http"

func NewRouter() http.Handler {
    mux := http.NewServeMux()

    // API эндпоинты
    mux.HandleFunc("/start", StartRun)
    mux.HandleFunc("/stop", StopRun)
    mux.HandleFunc("/stream", Stream)
    mux.HandleFunc("/export", ExportCSV)

    // статика
    // fs := http.FileServer(http.Dir("static"))
    // mux.Handle("/", fs)     // index.html по умолчанию
    // mux.Handle("/help", fs) // help.html

    mux.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        http.ServeFile(w, r, "static/index.html")
    })
    mux.HandleFunc("/help", func(w http.ResponseWriter, r *http.Request) {
        http.ServeFile(w, r, "static/help.html")
    })

    return mux
}
```

State.go

```
package server
```

```go
import (
    "context"
    "sync"
    "time"

    "idz1_opt/internal/optimizer"
)

// параметры запуска метода
type RunParams struct {
    Func    string  `json:"func"`
    A       float64 `json:"a"`
    B       float64 `json:"b"`
    Eps     float64 `json:"eps"`
    Delta   float64 `json:"delta"`
    MaxIter int     `json:"maxIter"`
}

// состояние одного запуска
type RunState struct {
    ID        string
    Params    RunParams
    CreatedAt time.Time

    LastIter optimizer.Iter
    Iters    []optimizer.Iter

    Err    string
    Done   bool
    Cancel context.CancelFunc
}

var (
    runsMu sync.Mutex
```

```
    runs   = map[string]*RunState{}
)

func saveRun(rs *RunState) {
    runsMu.Lock()
    defer runsMu.Unlock()
    runs[rs.ID] = rs
}

func getRun(id string) *RunState {
    runsMu.Lock()
    defer runsMu.Unlock()
    return runs[id]
}
```

Sse.go

```
package sse

import "sync"

// простой hub для SSE по runID

var (
    mu    sync.Mutex
    conns = map[string][]chan string{}
)

// Subscribe подписывает клиента на id, возвращает канал и функцию-unsubscribe
func Subscribe(id string) (chan string, func()) {
    ch := make(chan string, 16)
```

```
    mu.Lock()
    conns[id] = append(conns[id], ch)
    mu.Unlock()

    cancel := func() {
        mu.Lock()
        defer mu.Unlock()
        list := conns[id]
        for i, c := range list {
            if c == ch {
                conns[id] = append(list[:i], list[i+1:]...)
                break
            }
        }
    }

    return ch, cancel
}

// Publish отсылает сообщение всем подписчикам runID
func Publish(id, msg string) {
    mu.Lock()
    list := append([]chan string(nil), conns[id]...)
    mu.Unlock()

    for _, ch := range list {
        select {
        case ch <- msg:
        default:
            // игнорируем, если канал забит
        }
    }
}
```