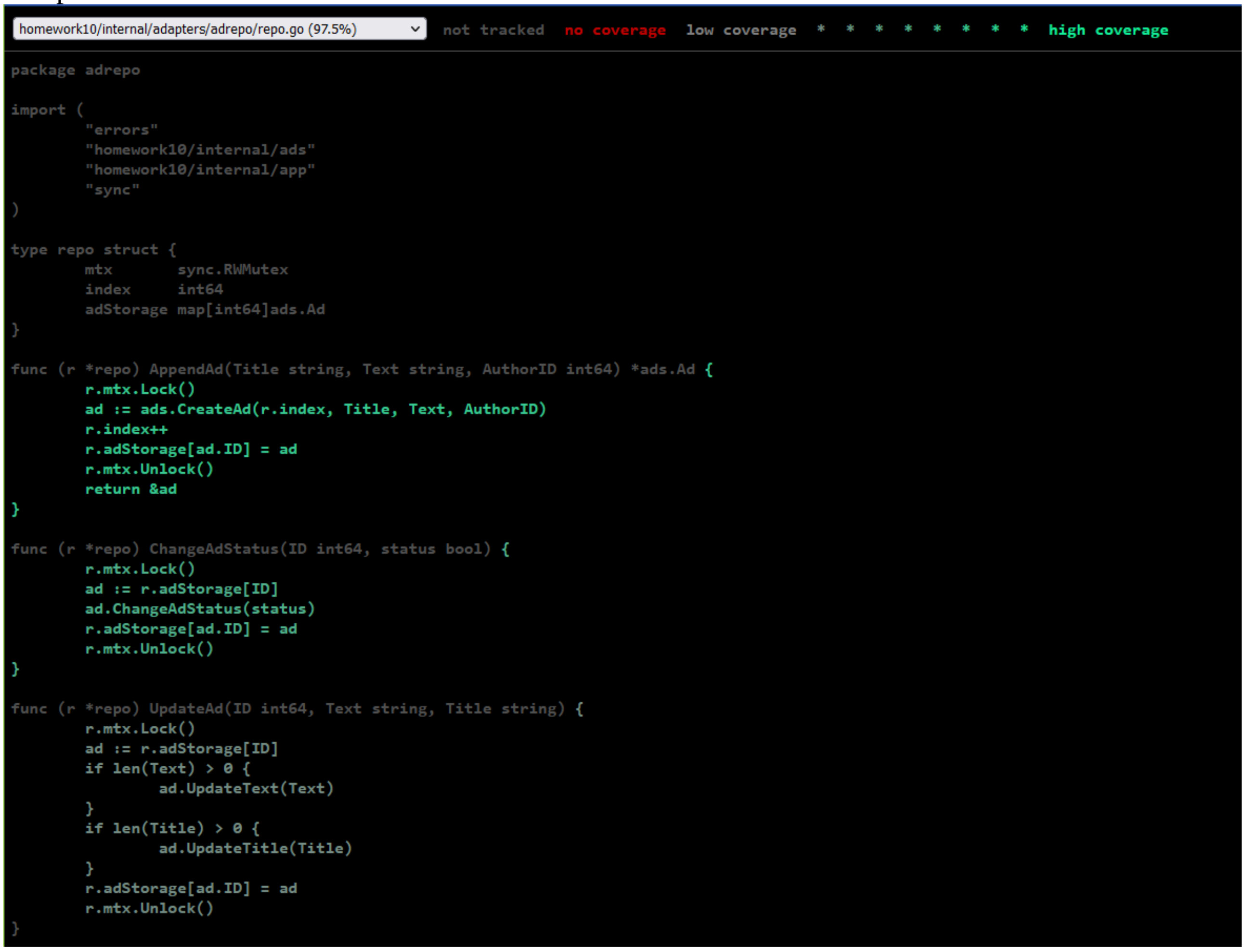
go test -v -race -coverpkg=./... ./…

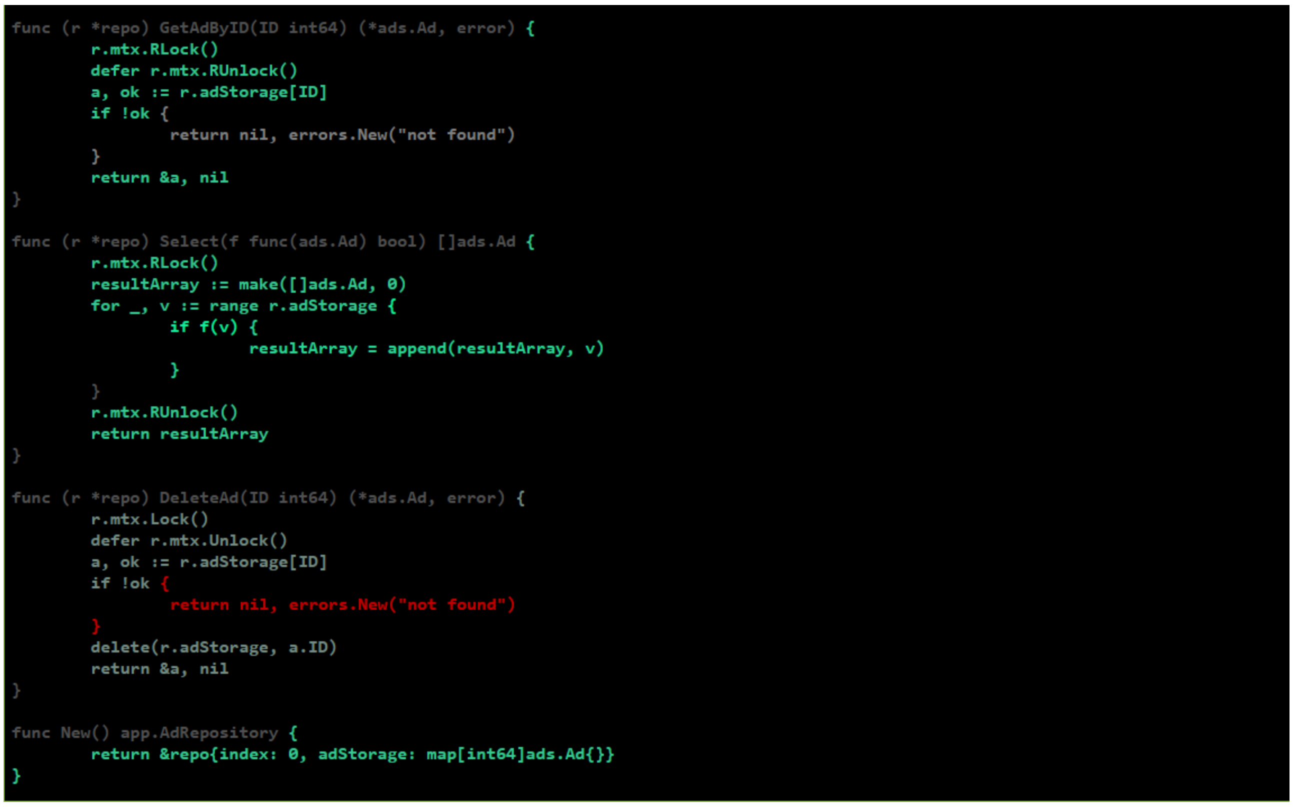
Output

```
...
PASS
        homework10/internal/adapters/adrepo      coverage: 97.5% of statements in ./...
        homework10/internal/adapters/userrepo    coverage: 96.4% of statements in ./...
        homework10/internal/ads                  coverage: 100.0% of statements in ./...
        homework10/internal/app                  coverage: 89.4% of statements in ./...
        homework10/internal/ports                coverage: 86.8% of statements in ./...
        homework10/internal/ports/grpc           coverage: 53.0% of statements in ./...
        homework10/internal/ports/httpgin        coverage: 71.7% of statements in ./...
        homework10/internal/tests                coverage: 81.2% of statements in ./...
        homework10/internal/tests/mocks          coverage: 92.6% of statements in ./...
        homework10/internal/users                coverage: 100.0% of statements in ./...
ok      homework10/internal/tests       1.723s   coverage: 97.5% of statements in ./…
```

adrepo

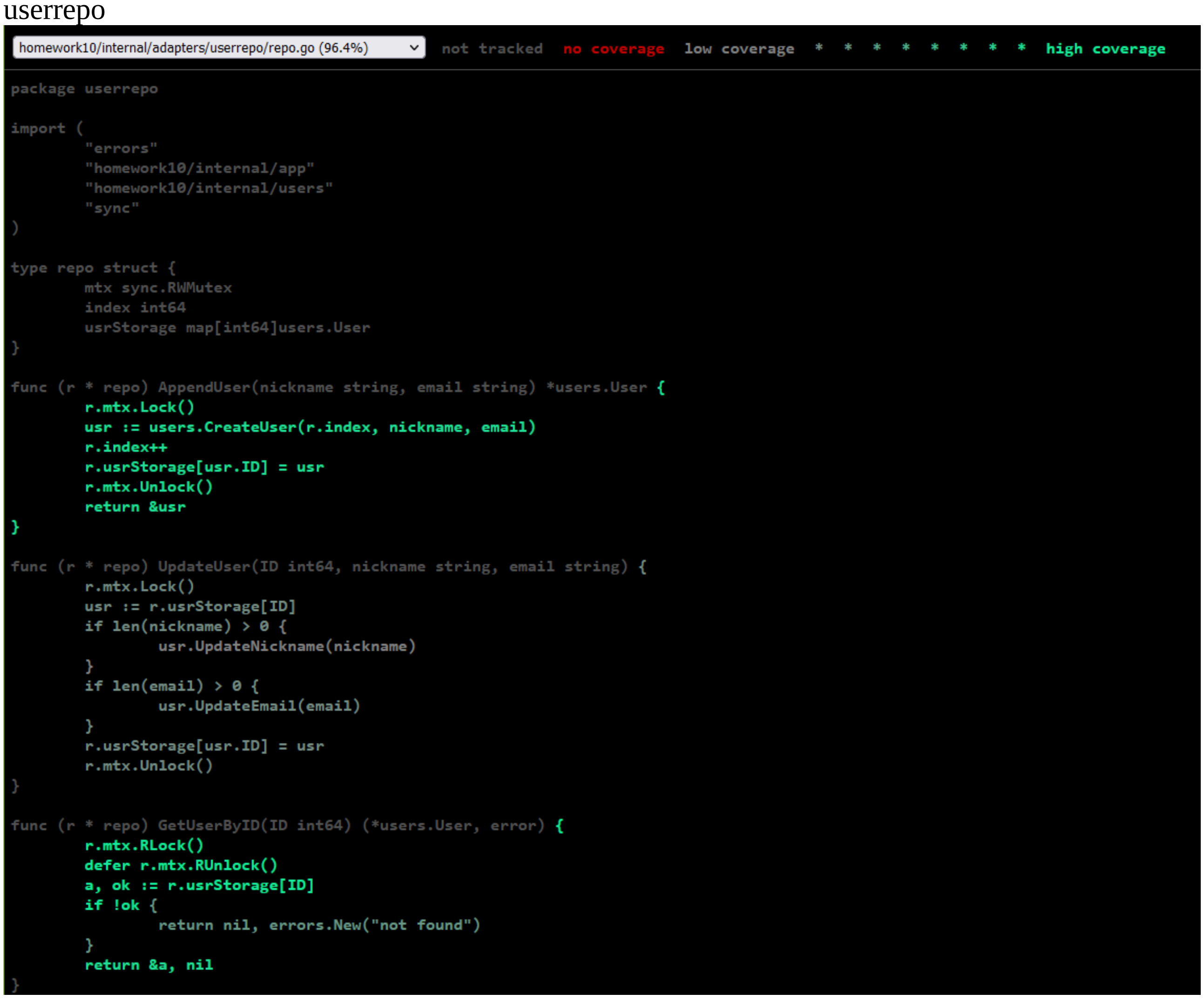


```go
package adrepo

import (
	"errors"
	"homework10/internal/ads"
	"homework10/internal/app"
	"sync"
)

type repo struct {
	mtx       sync.RWMutex
	index     int64
	adStorage map[int64]ads.Ad
}

func (r *repo) AppendAd(Title string, Text string, AuthorID int64) *ads.Ad {
	r.mtx.Lock()
	ad := ads.CreateAd(r.index, Title, Text, AuthorID)
	r.index++
	r.adStorage[ad.ID] = ad
	r.mtx.Unlock()
	return &ad
}

func (r *repo) ChangeAdStatus(ID int64, status bool) {
	r.mtx.Lock()
	ad := r.adStorage[ID]
	ad.ChangeAdStatus(status)
	r.adStorage[ad.ID] = ad
	r.mtx.Unlock()
}

func (r *repo) UpdateAd(ID int64, Text string, Title string) {
	r.mtx.Lock()
	ad := r.adStorage[ID]
	if len(Text) > 0 {
		ad.UpdateText(Text)
	}
	if len(Title) > 0 {
		ad.UpdateTitle(Title)
	}
	r.adStorage[ad.ID] = ad
	r.mtx.Unlock()
}
```



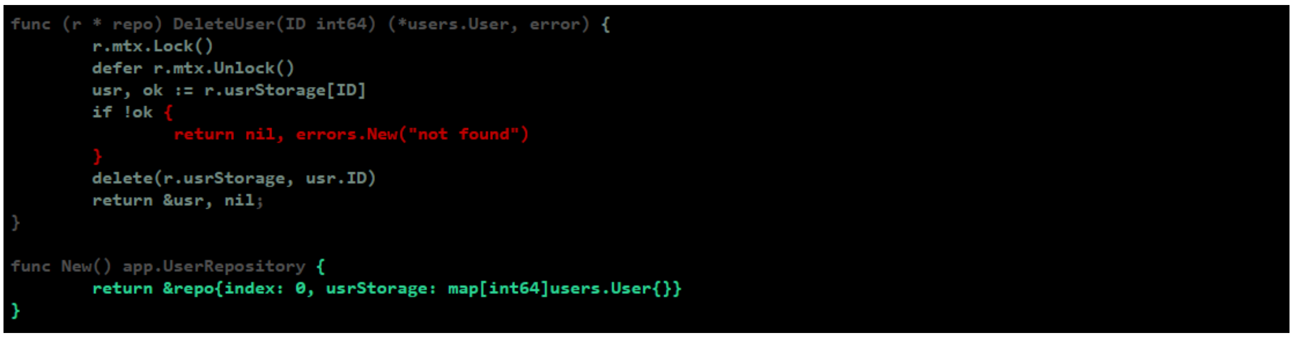
```go
func (r *repo) GetAdByID(ID int64) (*ads.Ad, error) {
	r.mtx.RLock()
	defer r.mtx.RUnlock()
	a, ok := r.adStorage[ID]
	if !ok {
		return nil, errors.New("not found")
	}
	return &a, nil
}

func (r *repo) Select(f func(ads.Ad) bool) []ads.Ad {
	r.mtx.RLock()
	resultArray := make([]ads.Ad, 0)
	for _, v := range r.adStorage {
		if f(v) {
			resultArray = append(resultArray, v)
		}
	}
	r.mtx.RUnlock()
	return resultArray
}

func (r *repo) DeleteAd(ID int64) (*ads.Ad, error) {
	r.mtx.Lock()
	defer r.mtx.Unlock()
	a, ok := r.adStorage[ID]
	if !ok {
		return nil, errors.New("not found")
	}
	delete(r.adStorage, a.ID)
	return &a, nil
}

func New() app.AdRepository {
	return &repo{index: 0, adStorage: map[int64]ads.Ad{}}
}
```

userrepo



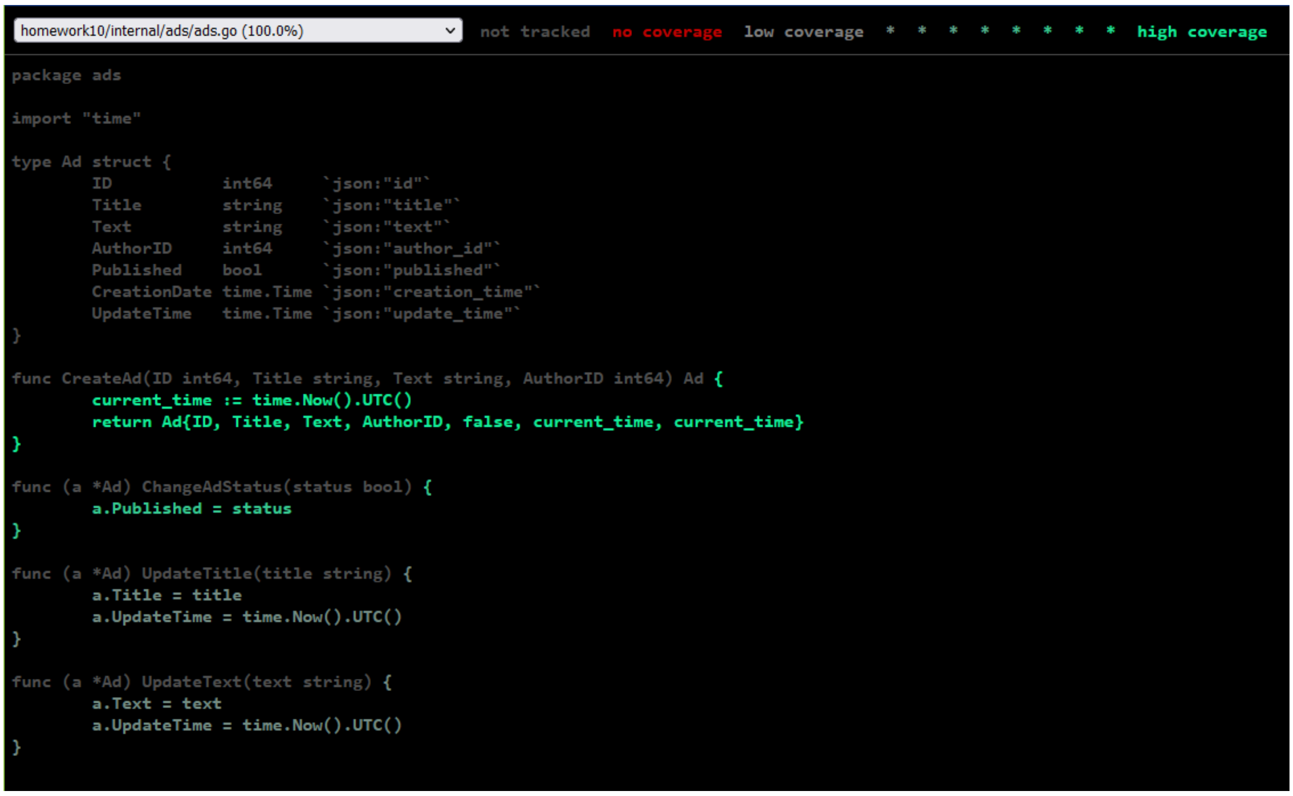
homework10/internal/adapters/userrepo/repo.go (96.4%)
not tracked no coverage low coverage * * * * * * * * high coverage

```go
package userrepo

import (
	"errors"
	"homework10/internal/app"
	"homework10/internal/users"
	"sync"
)

type repo struct {
	mtx sync.RWMutex
	index int64
	usrStorage map[int64]users.User
}

func (r * repo) AppendUser(nickname string, email string) *users.User {
	r.mtx.Lock()
	usr := users.CreateUser(r.index, nickname, email)
	r.index++
	r.usrStorage[usr.ID] = usr
	r.mtx.Unlock()
	return &usr
}

func (r * repo) UpdateUser(ID int64, nickname string, email string) {
	r.mtx.Lock()
	usr := r.usrStorage[ID]
	if len(nickname) > 0 {
		usr.UpdateNickname(nickname)
	}
	if len(email) > 0 {
		usr.UpdateEmail(email)
	}
	r.usrStorage[usr.ID] = usr
	r.mtx.Unlock()
}

func (r * repo) GetUserByID(ID int64) (*users.User, error) {
	r.mtx.RLock()
	defer r.mtx.RUnlock()
	a, ok := r.usrStorage[ID]
	if !ok {
		return nil, errors.New("not found")
	}
	return &a, nil
}
```



```go
func (r * repo) DeleteUser(ID int64) (*users.User, error) {
	r.mtx.Lock()
	defer r.mtx.Unlock()
	usr, ok := r.usrStorage[ID]
	if !ok {
		return nil, errors.New("not found")
	}
	delete(r.usrStorage, usr.ID)
	return &usr, nil;
}

func New() app.UserRepository {
	return &repo{index: 0, usrStorage: map[int64]users.User{}}
}
```

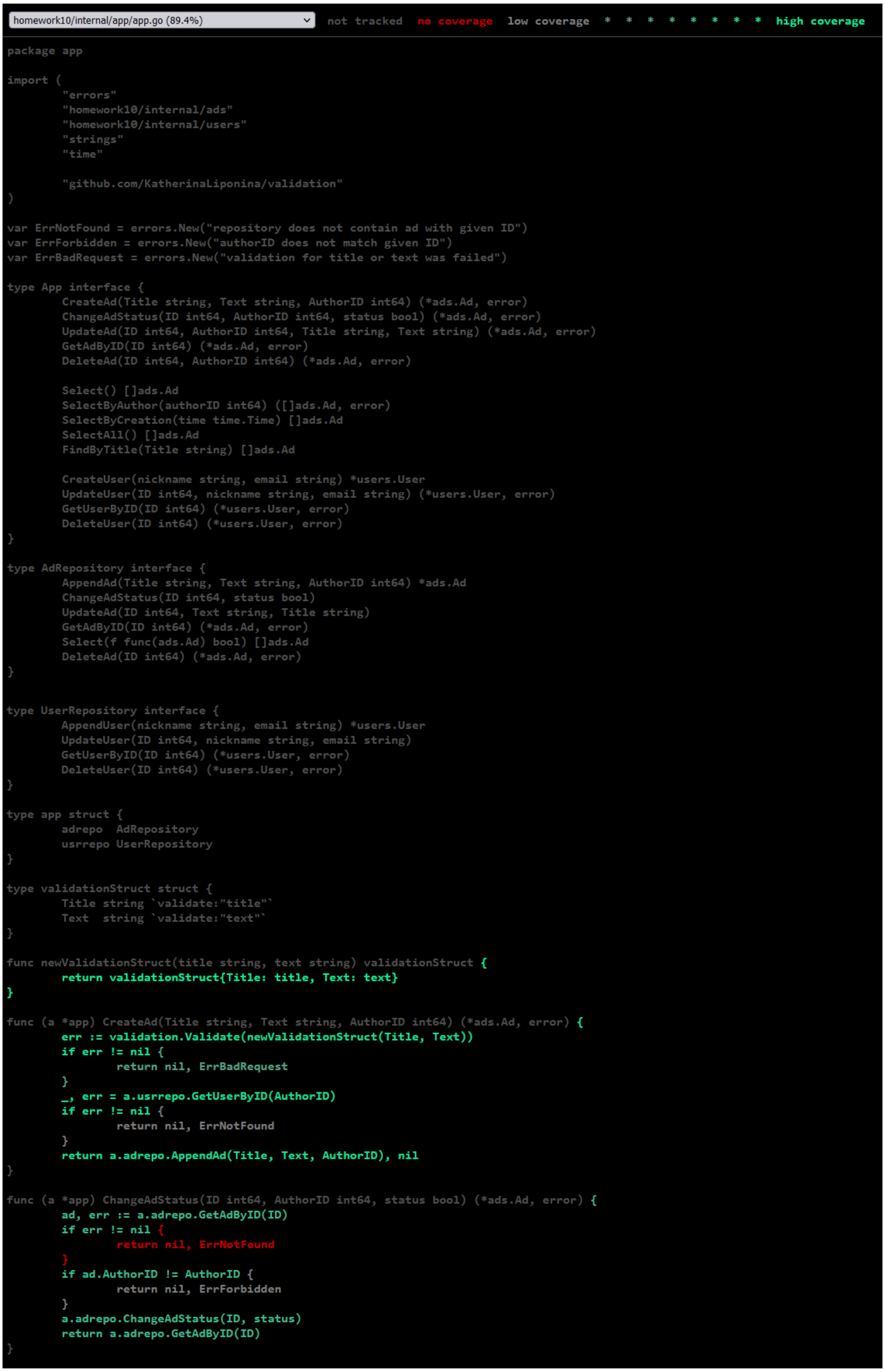
ads



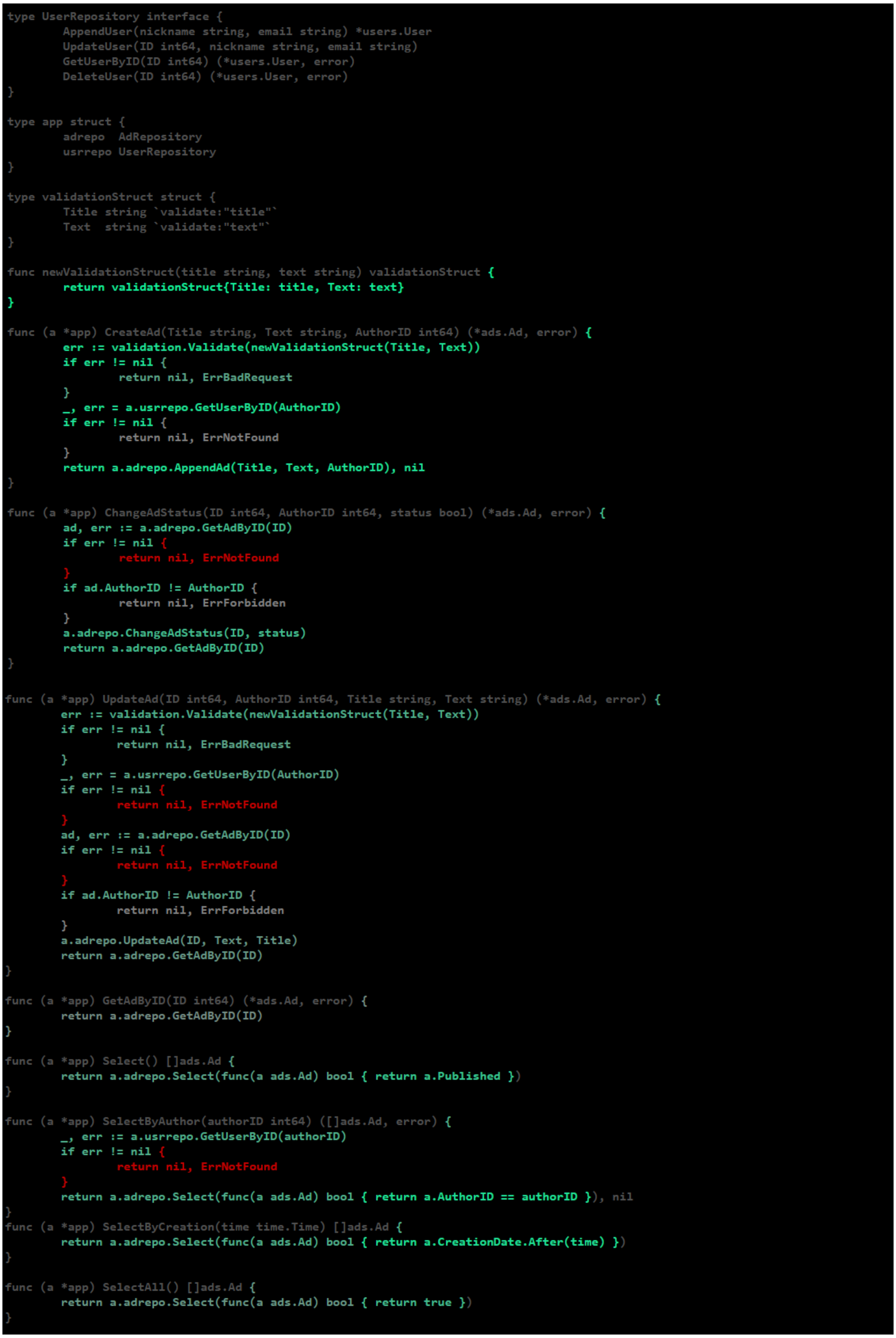
```
homework10/internal/ads/ads.go (100.0%)   not tracked  no coverage  low coverage  *  *  *  *  *  *  *  *  high coverage
```

```go
package ads

import "time"

type Ad struct {
	ID           int64     `json:"id"`
	Title        string    `json:"title"`
	Text         string    `json:"text"`
	AuthorID     int64     `json:"author_id"`
	Published    bool      `json:"published"`
	CreationDate time.Time `json:"creation_time"`
	UpdateTime   time.Time `json:"update_time"`
}

func CreateAd(ID int64, Title string, Text string, AuthorID int64) Ad {
	current_time := time.Now().UTC()
	return Ad{ID, Title, Text, AuthorID, false, current_time, current_time}
}

func (a *Ad) ChangeAdStatus(status bool) {
	a.Published = status
}

func (a *Ad) UpdateTitle(title string) {
	a.Title = title
	a.UpdateTime = time.Now().UTC()
}

func (a *Ad) UpdateText(text string) {
	a.Text = text
	a.UpdateTime = time.Now().UTC()
}
```

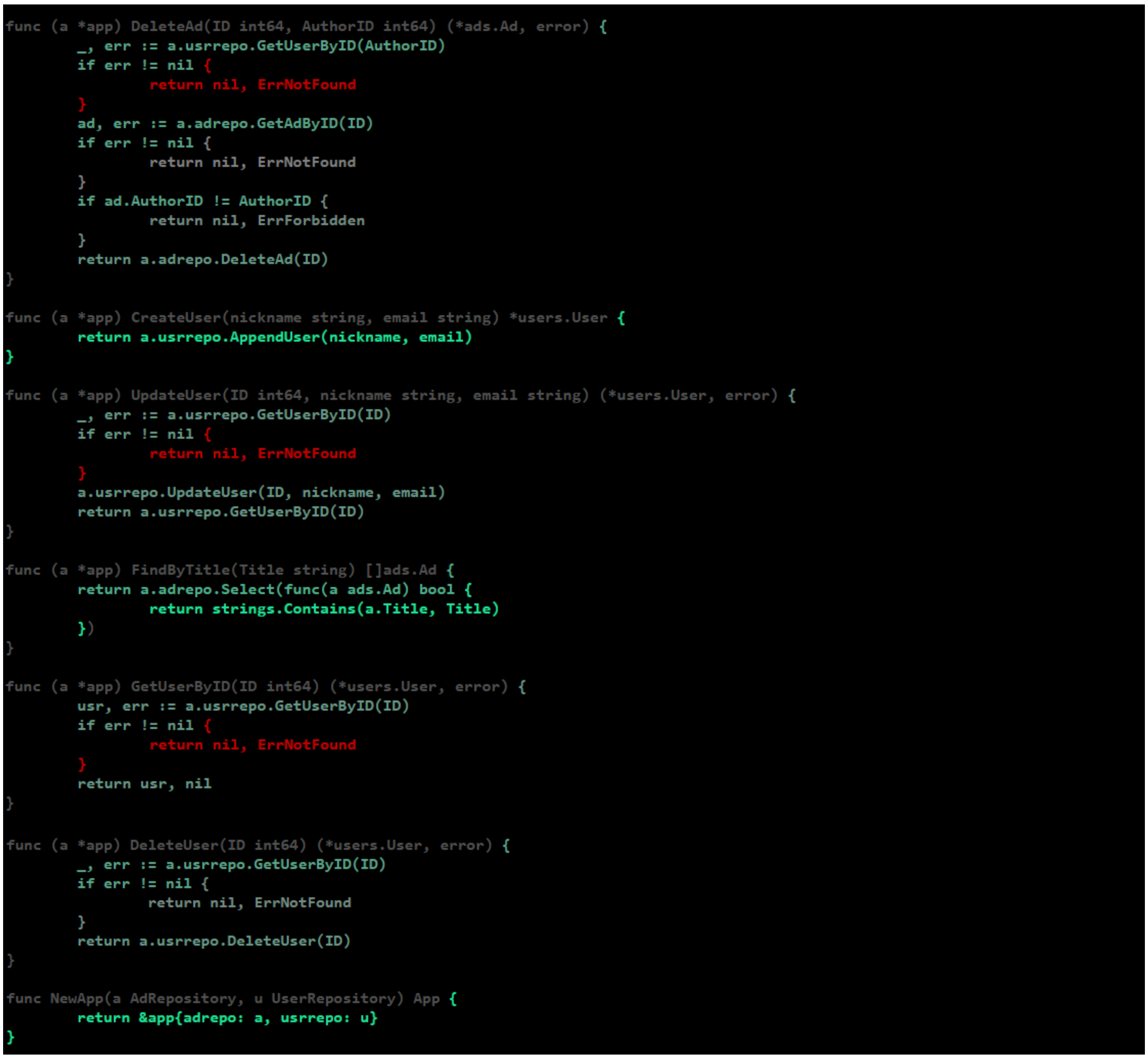
app



```go
package app

import (
	"errors"
	"homework10/internal/ads"
	"homework10/internal/users"
	"strings"
	"time"

	"github.com/KatherinaLiponina/validation"
)

var ErrNotFound = errors.New("repository does not contain ad with given ID")
var ErrForbidden = errors.New("authorID does not match given ID")
var ErrBadRequest = errors.New("validation for title or text was failed")

type App interface {
	CreateAd(Title string, Text string, AuthorID int64) (*ads.Ad, error)
	ChangeAdStatus(ID int64, AuthorID int64, status bool) (*ads.Ad, error)
	UpdateAd(ID int64, AuthorID int64, Title string, Text string) (*ads.Ad, error)
	GetAdByID(ID int64) (*ads.Ad, error)
	DeleteAd(ID int64, AuthorID int64) (*ads.Ad, error)

	Select() []ads.Ad
	SelectByAuthor(authorID int64) ([]ads.Ad, error)
	SelectByCreation(time time.Time) []ads.Ad
	SelectAll() []ads.Ad
	FindByTitle(Title string) []ads.Ad

	CreateUser(nickname string, email string) *users.User
	UpdateUser(ID int64, nickname string, email string) (*users.User, error)
	GetUserByID(ID int64) (*users.User, error)
	DeleteUser(ID int64) (*users.User, error)
}

type AdRepository interface {
	AppendAd(Title string, Text string, AuthorID int64) *ads.Ad
	ChangeAdStatus(ID int64, status bool)
	UpdateAd(ID int64, Text string, Title string)
	GetAdByID(ID int64) (*ads.Ad, error)
	Select(f func(ads.Ad) bool) []ads.Ad
	DeleteAd(ID int64) (*ads.Ad, error)
}


type UserRepository interface {
	AppendUser(nickname string, email string) *users.User
	UpdateUser(ID int64, nickname string, email string)
	GetUserByID(ID int64) (*users.User, error)
	DeleteUser(ID int64) (*users.User, error)
}

type app struct {
	adrepo  AdRepository
	usrrepo UserRepository
}

type validationStruct struct {
	Title string `validate:"title"`
	Text  string `validate:"text"`
}

func newValidationStruct(title string, text string) validationStruct {
	return validationStruct{Title: title, Text: text}
}

func (a *app) CreateAd(Title string, Text string, AuthorID int64) (*ads.Ad, error) {
	err := validation.Validate(newValidationStruct(Title, Text))
	if err != nil {
		return nil, ErrBadRequest
	}
	_, err = a.usrrepo.GetUserByID(AuthorID)
	if err != nil {
		return nil, ErrNotFound
	}
	return a.adrepo.AppendAd(Title, Text, AuthorID), nil
}

func (a *app) ChangeAdStatus(ID int64, AuthorID int64, status bool) (*ads.Ad, error) {
	ad, err := a.adrepo.GetAdByID(ID)
	if err != nil {
		return nil, ErrNotFound
	}
	if ad.AuthorID != AuthorID {
		return nil, ErrForbidden
	}
	a.adrepo.ChangeAdStatus(ID, status)
	return a.adrepo.GetAdByID(ID)
}
```

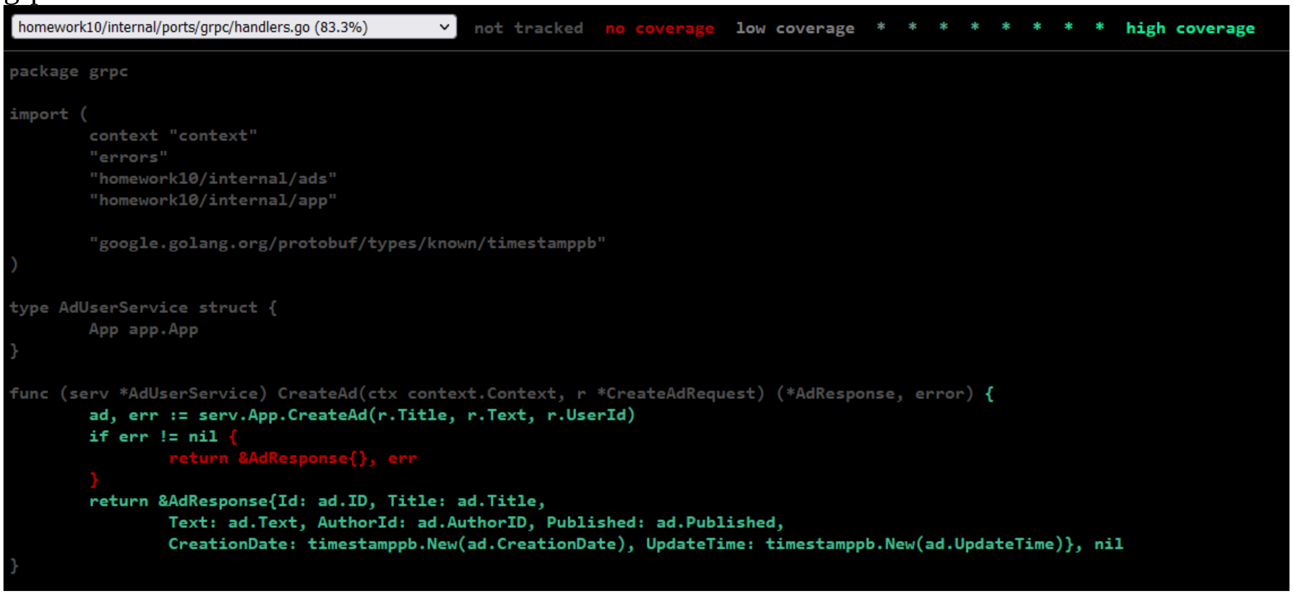


```go
type UserRepository interface {
	AppendUser(nickname string, email string) *users.User
	UpdateUser(ID int64, nickname string, email string)
	GetUserByID(ID int64) (*users.User, error)
	DeleteUser(ID int64) (*users.User, error)
}

type app struct {
	adrepo  AdRepository
	usrrepo UserRepository
}

type validationStruct struct {
	Title string `validate:"title"`
	Text  string `validate:"text"`
}

func newValidationStruct(title string, text string) validationStruct {
	return validationStruct{Title: title, Text: text}
}

func (a *app) CreateAd(Title string, Text string, AuthorID int64) (*ads.Ad, error) {
	err := validation.Validate(newValidationStruct(Title, Text))
	if err != nil {
		return nil, ErrBadRequest
	}
	_, err = a.usrrepo.GetUserByID(AuthorID)
	if err != nil {
		return nil, ErrNotFound
	}
	return a.adrepo.AppendAd(Title, Text, AuthorID), nil
}

func (a *app) ChangeAdStatus(ID int64, AuthorID int64, status bool) (*ads.Ad, error) {
	ad, err := a.adrepo.GetAdByID(ID)
	if err != nil {
		return nil, ErrNotFound
	}
	if ad.AuthorID != AuthorID {
		return nil, ErrForbidden
	}
	a.adrepo.ChangeAdStatus(ID, status)
	return a.adrepo.GetAdByID(ID)
}

func (a *app) UpdateAd(ID int64, AuthorID int64, Title string, Text string) (*ads.Ad, error) {
	err := validation.Validate(newValidationStruct(Title, Text))
	if err != nil {
		return nil, ErrBadRequest
	}
	_, err = a.usrrepo.GetUserByID(AuthorID)
	if err != nil {
		return nil, ErrNotFound
	}
	ad, err := a.adrepo.GetAdByID(ID)
	if err != nil {
		return nil, ErrNotFound
	}
	if ad.AuthorID != AuthorID {
		return nil, ErrForbidden
	}
	a.adrepo.UpdateAd(ID, Text, Title)
	return a.adrepo.GetAdByID(ID)
}

func (a *app) GetAdByID(ID int64) (*ads.Ad, error) {
	return a.adrepo.GetAdByID(ID)
}

func (a *app) Select() []ads.Ad {
	return a.adrepo.Select(func(a ads.Ad) bool { return a.Published })
}

func (a *app) SelectByAuthor(authorID int64) ([]ads.Ad, error) {
	_, err := a.usrrepo.GetUserByID(authorID)
	if err != nil {
		return nil, ErrNotFound
	}
	return a.adrepo.Select(func(a ads.Ad) bool { return a.AuthorID == authorID }), nil
}
func (a *app) SelectByCreation(time time.Time) []ads.Ad {
	return a.adrepo.Select(func(a ads.Ad) bool { return a.CreationDate.After(time) })
}

func (a *app) SelectAll() []ads.Ad {
	return a.adrepo.Select(func(a ads.Ad) bool { return true })
}
```



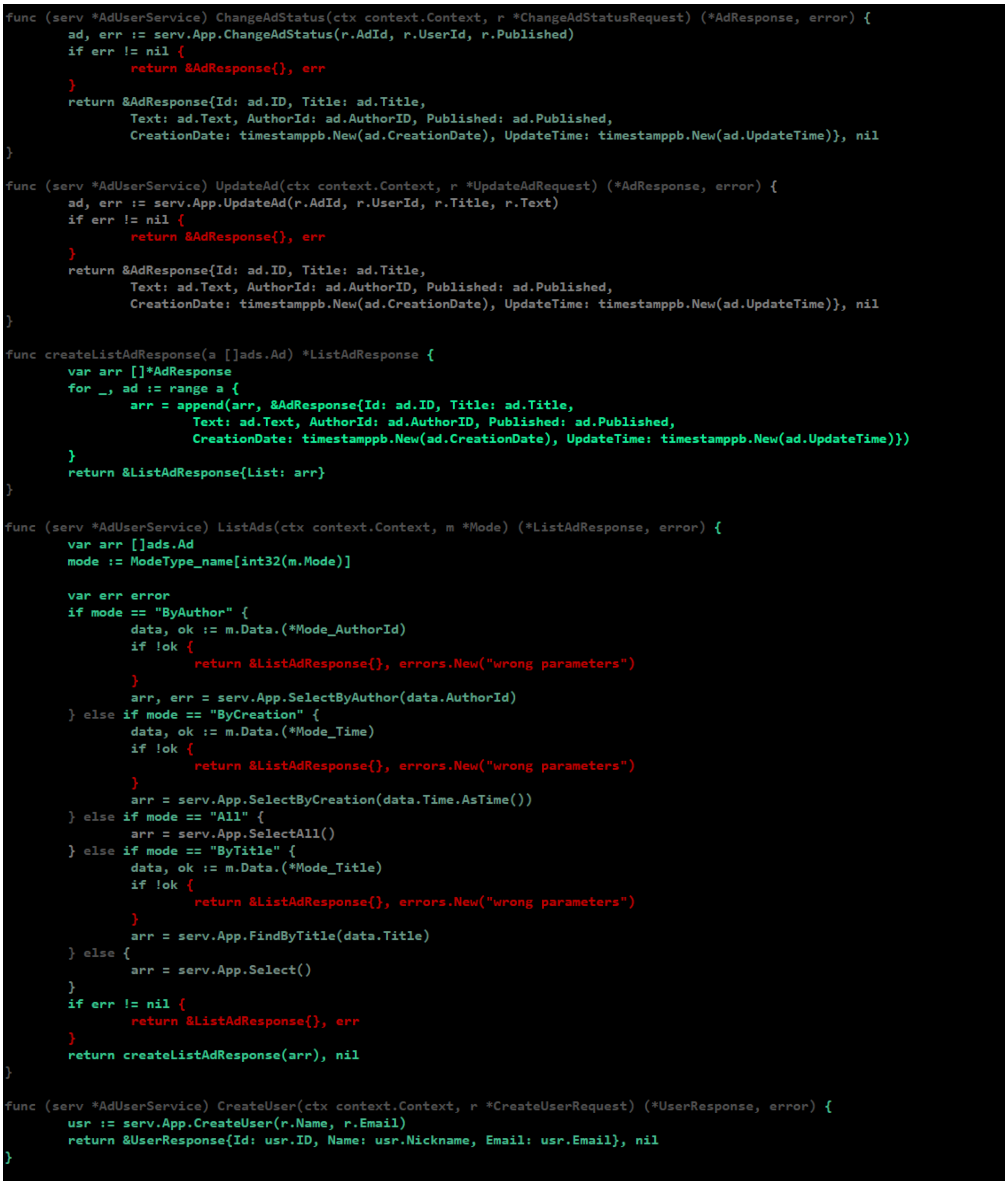
```go
func (a *app) DeleteAd(ID int64, AuthorID int64) (*ads.Ad, error) {
	_, err := a.usrrepo.GetUserByID(AuthorID)
	if err != nil {
		return nil, ErrNotFound
	}
	ad, err := a.adrepo.GetAdByID(ID)
	if err != nil {
		return nil, ErrNotFound
	}
	if ad.AuthorID != AuthorID {
		return nil, ErrForbidden
	}
	return a.adrepo.DeleteAd(ID)
}

func (a *app) CreateUser(nickname string, email string) *users.User {
	return a.usrrepo.AppendUser(nickname, email)
}

func (a *app) UpdateUser(ID int64, nickname string, email string) (*users.User, error) {
	_, err := a.usrrepo.GetUserByID(ID)
	if err != nil {
		return nil, ErrNotFound
	}
	a.usrrepo.UpdateUser(ID, nickname, email)
	return a.usrrepo.GetUserByID(ID)
}

func (a *app) FindByTitle(Title string) []ads.Ad {
	return a.adrepo.Select(func(a ads.Ad) bool {
		return strings.Contains(a.Title, Title)
	})
}

func (a *app) GetUserByID(ID int64) (*users.User, error) {
	usr, err := a.usrrepo.GetUserByID(ID)
	if err != nil {
		return nil, ErrNotFound
	}
	return usr, nil
}

func (a *app) DeleteUser(ID int64) (*users.User, error) {
	_, err := a.usrrepo.GetUserByID(ID)
	if err != nil {
		return nil, ErrNotFound
	}
	return a.usrrepo.DeleteUser(ID)
}

func NewApp(a AdRepository, u UserRepository) App {
	return &app{adrepo: a, usrrepo: u}
}
```

grpc/handler

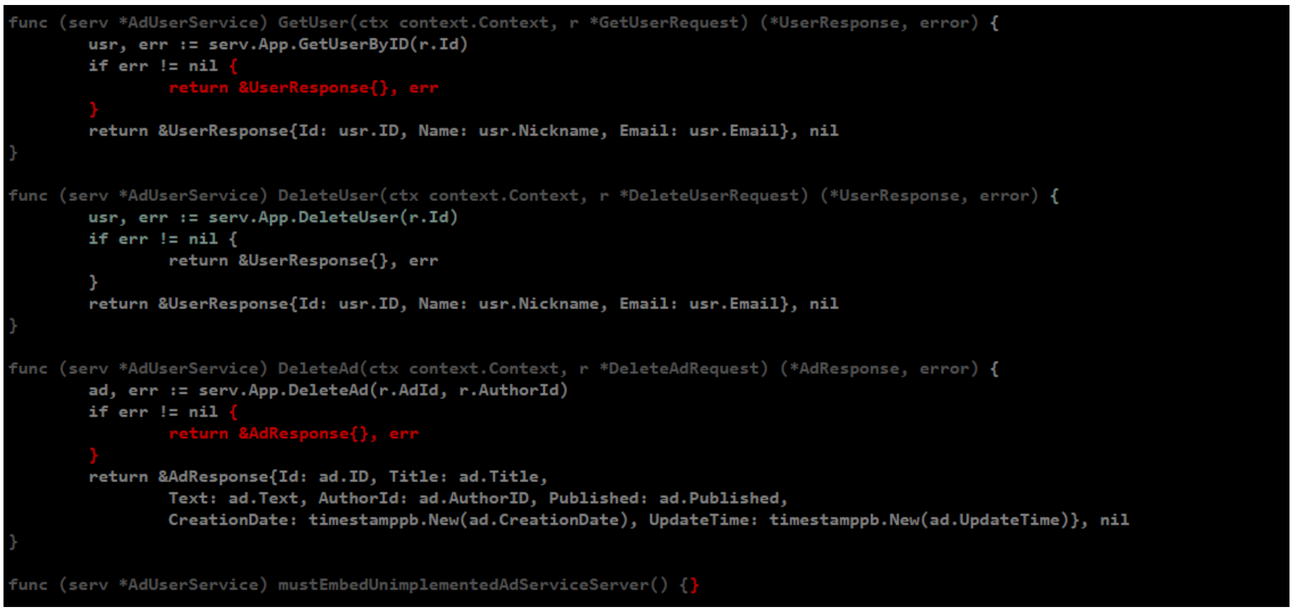


homework10/internal/ports/grpc/handlers.go (83.3%)   not tracked   no coverage   low coverage   * * * * * * * *   high coverage

```go
package grpc

import (
	context "context"
	"errors"
	"homework10/internal/ads"
	"homework10/internal/app"

	"google.golang.org/protobuf/types/known/timestamppb"
)

type AdUserService struct {
	App app.App
}

func (serv *AdUserService) CreateAd(ctx context.Context, r *CreateAdRequest) (*AdResponse, error) {
	ad, err := serv.App.CreateAd(r.Title, r.Text, r.UserId)
	if err != nil {
		return &AdResponse{}, err
	}
	return &AdResponse{Id: ad.ID, Title: ad.Title,
		Text: ad.Text, AuthorId: ad.AuthorID, Published: ad.Published,
		CreationDate: timestamppb.New(ad.CreationDate), UpdateTime: timestamppb.New(ad.UpdateTime)}, nil
}
```

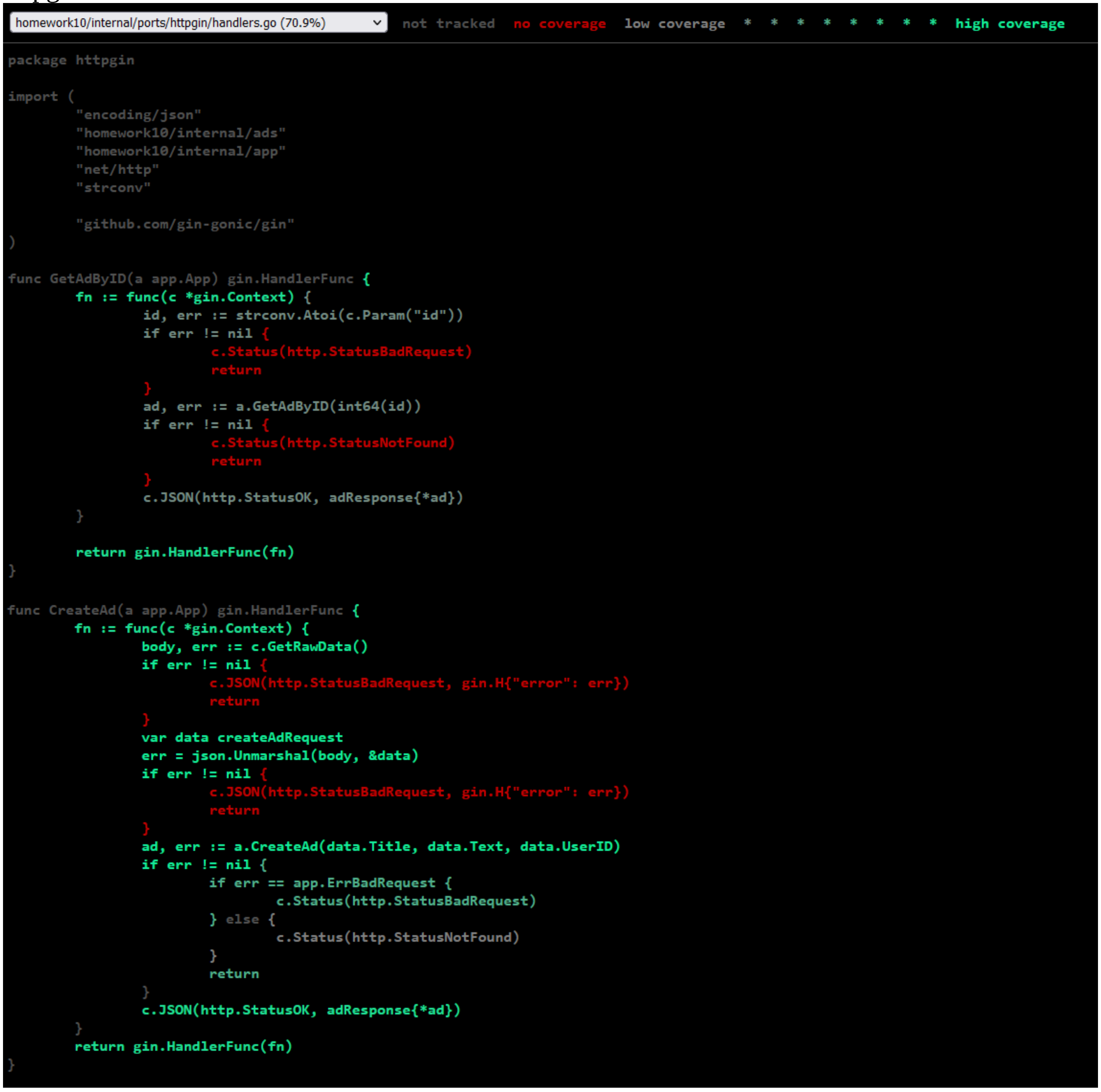


```go
func (serv *AdUserService) ChangeAdStatus(ctx context.Context, r *ChangeAdStatusRequest) (*AdResponse, error) {
	ad, err := serv.App.ChangeAdStatus(r.AdId, r.UserId, r.Published)
	if err != nil {
		return &AdResponse{}, err
	}
	return &AdResponse{Id: ad.ID, Title: ad.Title,
		Text: ad.Text, AuthorId: ad.AuthorID, Published: ad.Published,
		CreationDate: timestamppb.New(ad.CreationDate), UpdateTime: timestamppb.New(ad.UpdateTime)}, nil
}

func (serv *AdUserService) UpdateAd(ctx context.Context, r *UpdateAdRequest) (*AdResponse, error) {
	ad, err := serv.App.UpdateAd(r.AdId, r.UserId, r.Title, r.Text)
	if err != nil {
		return &AdResponse{}, err
	}
	return &AdResponse{Id: ad.ID, Title: ad.Title,
		Text: ad.Text, AuthorId: ad.AuthorID, Published: ad.Published,
		CreationDate: timestamppb.New(ad.CreationDate), UpdateTime: timestamppb.New(ad.UpdateTime)}, nil
}

func createListAdResponse(a []ads.Ad) *ListAdResponse {
	var arr []*AdResponse
	for _, ad := range a {
		arr = append(arr, &AdResponse{Id: ad.ID, Title: ad.Title,
			Text: ad.Text, AuthorId: ad.AuthorID, Published: ad.Published,
			CreationDate: timestamppb.New(ad.CreationDate), UpdateTime: timestamppb.New(ad.UpdateTime)})
	}
	return &ListAdResponse{List: arr}
}

func (serv *AdUserService) ListAds(ctx context.Context, m *Mode) (*ListAdResponse, error) {
	var arr []ads.Ad
	mode := ModeType_name[int32(m.Mode)]

	var err error
	if mode == "ByAuthor" {
		data, ok := m.Data.(*Mode_AuthorId)
		if !ok {
			return &ListAdResponse{}, errors.New("wrong parameters")
		}
		arr, err = serv.App.SelectByAuthor(data.AuthorId)
	} else if mode == "ByCreation" {
		data, ok := m.Data.(*Mode_Time)
		if !ok {
			return &ListAdResponse{}, errors.New("wrong parameters")
		}
		arr = serv.App.SelectByCreation(data.Time.AsTime())
	} else if mode == "All" {
		arr = serv.App.SelectAll()
	} else if mode == "ByTitle" {
		data, ok := m.Data.(*Mode_Title)
		if !ok {
			return &ListAdResponse{}, errors.New("wrong parameters")
		}
		arr = serv.App.FindByTitle(data.Title)
	} else {
		arr = serv.App.Select()
	}
	if err != nil {
		return &ListAdResponse{}, err
	}
	return createListAdResponse(arr), nil
}

func (serv *AdUserService) CreateUser(ctx context.Context, r *CreateUserRequest) (*UserResponse, error) {
	usr := serv.App.CreateUser(r.Name, r.Email)
	return &UserResponse{Id: usr.ID, Name: usr.Nickname, Email: usr.Email}, nil
}
```



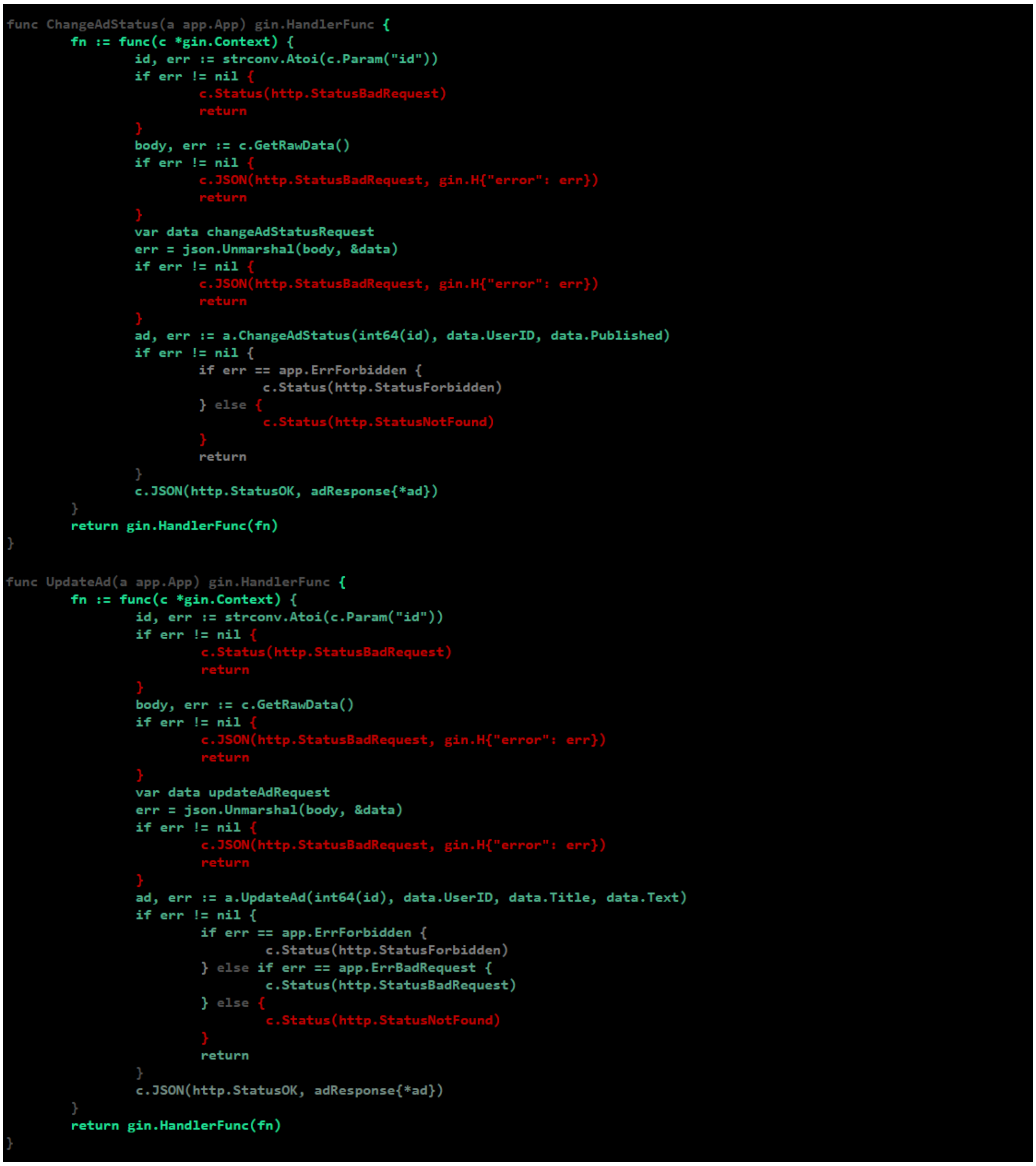
```go
func (serv *AdUserService) GetUser(ctx context.Context, r *GetUserRequest) (*UserResponse, error) {
	usr, err := serv.App.GetUserByID(r.Id)
	if err != nil {
		return &UserResponse{}, err
	}
	return &UserResponse{Id: usr.ID, Name: usr.Nickname, Email: usr.Email}, nil
}

func (serv *AdUserService) DeleteUser(ctx context.Context, r *DeleteUserRequest) (*UserResponse, error) {
	usr, err := serv.App.DeleteUser(r.Id)
	if err != nil {
		return &UserResponse{}, err
	}
	return &UserResponse{Id: usr.ID, Name: usr.Nickname, Email: usr.Email}, nil
}

func (serv *AdUserService) DeleteAd(ctx context.Context, r *DeleteAdRequest) (*AdResponse, error) {
	ad, err := serv.App.DeleteAd(r.AdId, r.AuthorId)
	if err != nil {
		return &AdResponse{}, err
	}
	return &AdResponse{Id: ad.ID, Title: ad.Title,
		Text: ad.Text, AuthorId: ad.AuthorID, Published: ad.Published,
		CreationDate: timestamppb.New(ad.CreationDate), UpdateTime: timestamppb.New(ad.UpdateTime)}, nil
}

func (serv *AdUserService) mustEmbedUnimplementedAdServiceServer() {}
```

httpgin/handlers

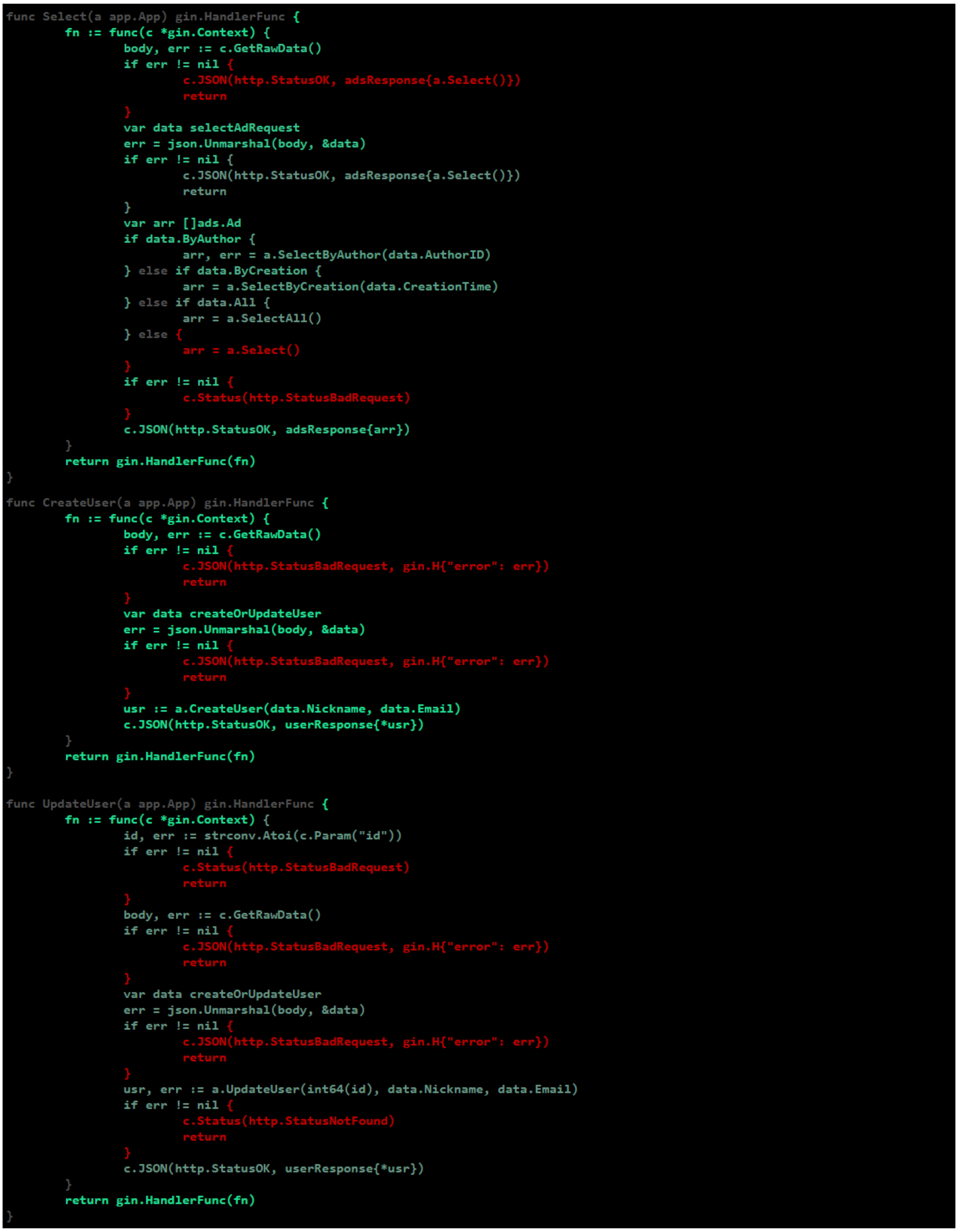


homework10/internal/ports/httpgin/handlers.go (70.9%)   not tracked   no coverage   low coverage   * * * * * * * *   high coverage

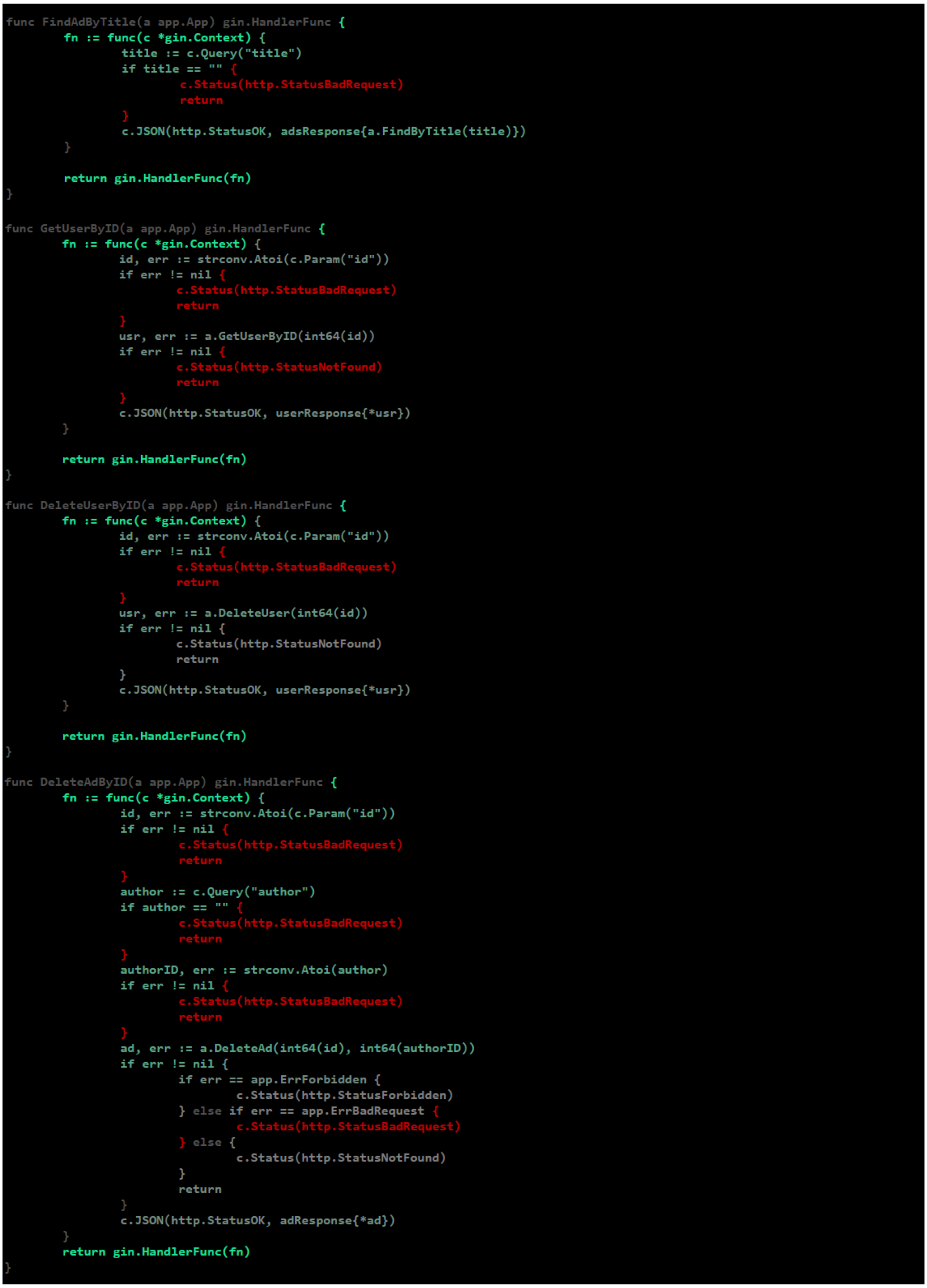
```go
package httpgin

import (
	"encoding/json"
	"homework10/internal/ads"
	"homework10/internal/app"
	"net/http"
	"strconv"

	"github.com/gin-gonic/gin"
)

func GetAdByID(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		id, err := strconv.Atoi(c.Param("id"))
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		ad, err := a.GetAdByID(int64(id))
		if err != nil {
			c.Status(http.StatusNotFound)
			return
		}
		c.JSON(http.StatusOK, adResponse{*ad})
	}

	return gin.HandlerFunc(fn)
}

func CreateAd(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		body, err := c.GetRawData()
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		var data createAdRequest
		err = json.Unmarshal(body, &data)
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		ad, err := a.CreateAd(data.Title, data.Text, data.UserID)
		if err != nil {
			if err == app.ErrBadRequest {
				c.Status(http.StatusBadRequest)
			} else {
				c.Status(http.StatusNotFound)
			}
			return
		}
		c.JSON(http.StatusOK, adResponse{*ad})
	}
	return gin.HandlerFunc(fn)
}
```



```go
func ChangeAdStatus(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		id, err := strconv.Atoi(c.Param("id"))
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		body, err := c.GetRawData()
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		var data changeAdStatusRequest
		err = json.Unmarshal(body, &data)
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		ad, err := a.ChangeAdStatus(int64(id), data.UserID, data.Published)
		if err != nil {
			if err == app.ErrForbidden {
				c.Status(http.StatusForbidden)
			} else {
				c.Status(http.StatusNotFound)
			}
			return
		}
		c.JSON(http.StatusOK, adResponse{*ad})
	}
	return gin.HandlerFunc(fn)
}

func UpdateAd(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		id, err := strconv.Atoi(c.Param("id"))
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		body, err := c.GetRawData()
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		var data updateAdRequest
		err = json.Unmarshal(body, &data)
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		ad, err := a.UpdateAd(int64(id), data.UserID, data.Title, data.Text)
		if err != nil {
			if err == app.ErrForbidden {
				c.Status(http.StatusForbidden)
			} else if err == app.ErrBadRequest {
				c.Status(http.StatusBadRequest)
			} else {
				c.Status(http.StatusNotFound)
			}
			return
		}
		c.JSON(http.StatusOK, adResponse{*ad})
	}
	return gin.HandlerFunc(fn)
}
```

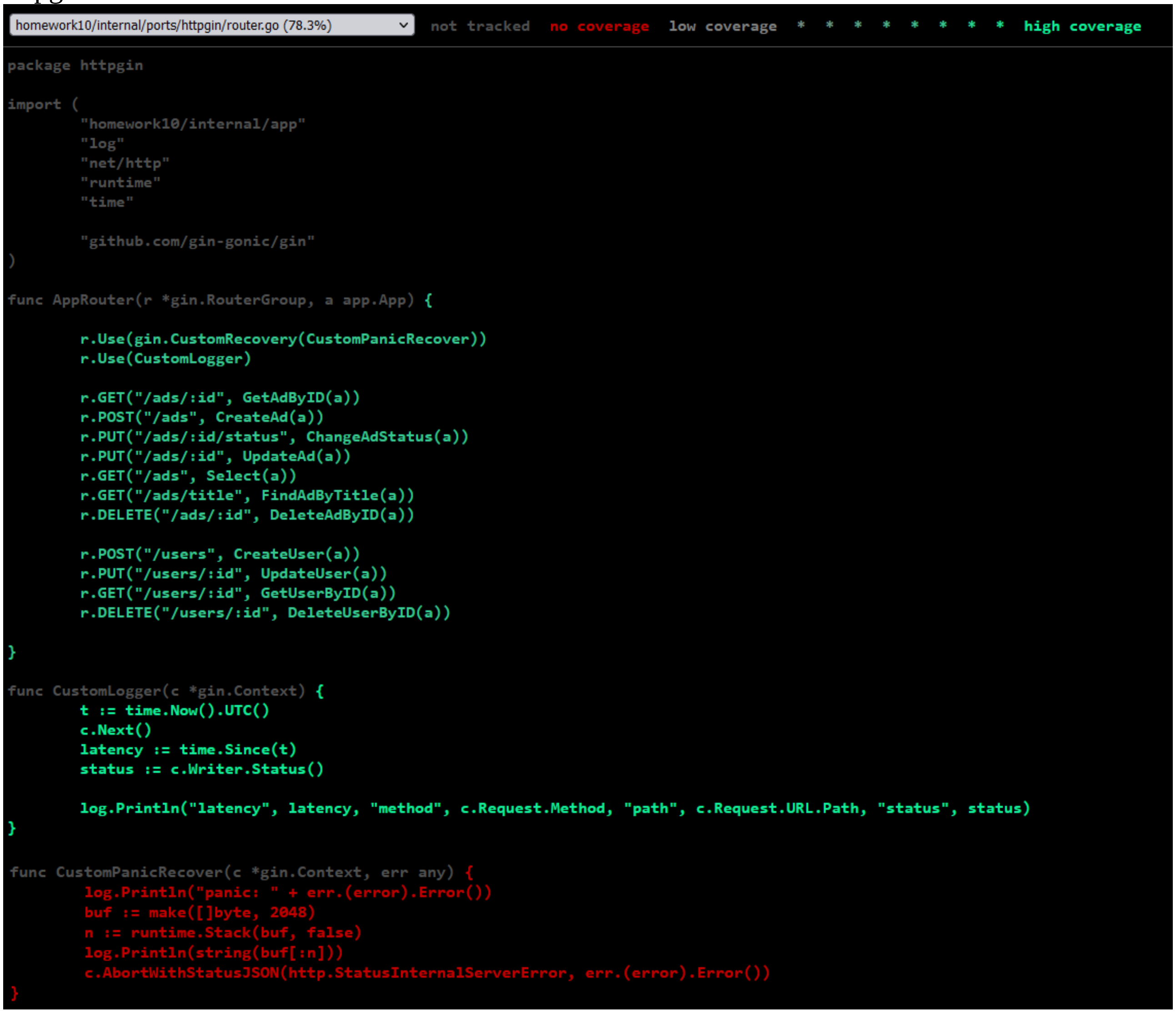


```go
func Select(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		body, err := c.GetRawData()
		if err != nil {
			c.JSON(http.StatusOK, adsResponse{a.Select()})
			return
		}
		var data selectAdRequest
		err = json.Unmarshal(body, &data)
		if err != nil {
			c.JSON(http.StatusOK, adsResponse{a.Select()})
			return
		}
		var arr []ads.Ad
		if data.ByAuthor {
			arr, err = a.SelectByAuthor(data.AuthorID)
		} else if data.ByCreation {
			arr = a.SelectByCreation(data.CreationTime)
		} else if data.All {
			arr = a.SelectAll()
		} else {
			arr = a.Select()
		}
		if err != nil {
			c.Status(http.StatusBadRequest)
		}
		c.JSON(http.StatusOK, adsResponse{arr})
	}
	return gin.HandlerFunc(fn)
}

func CreateUser(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		body, err := c.GetRawData()
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		var data createOrUpdateUser
		err = json.Unmarshal(body, &data)
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		usr := a.CreateUser(data.Nickname, data.Email)
		c.JSON(http.StatusOK, userResponse{*usr})
	}
	return gin.HandlerFunc(fn)
}

func UpdateUser(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		id, err := strconv.Atoi(c.Param("id"))
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		body, err := c.GetRawData()
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		var data createOrUpdateUser
		err = json.Unmarshal(body, &data)
		if err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": err})
			return
		}
		usr, err := a.UpdateUser(int64(id), data.Nickname, data.Email)
		if err != nil {
			c.Status(http.StatusNotFound)
			return
		}
		c.JSON(http.StatusOK, userResponse{*usr})
	}
	return gin.HandlerFunc(fn)
}
```



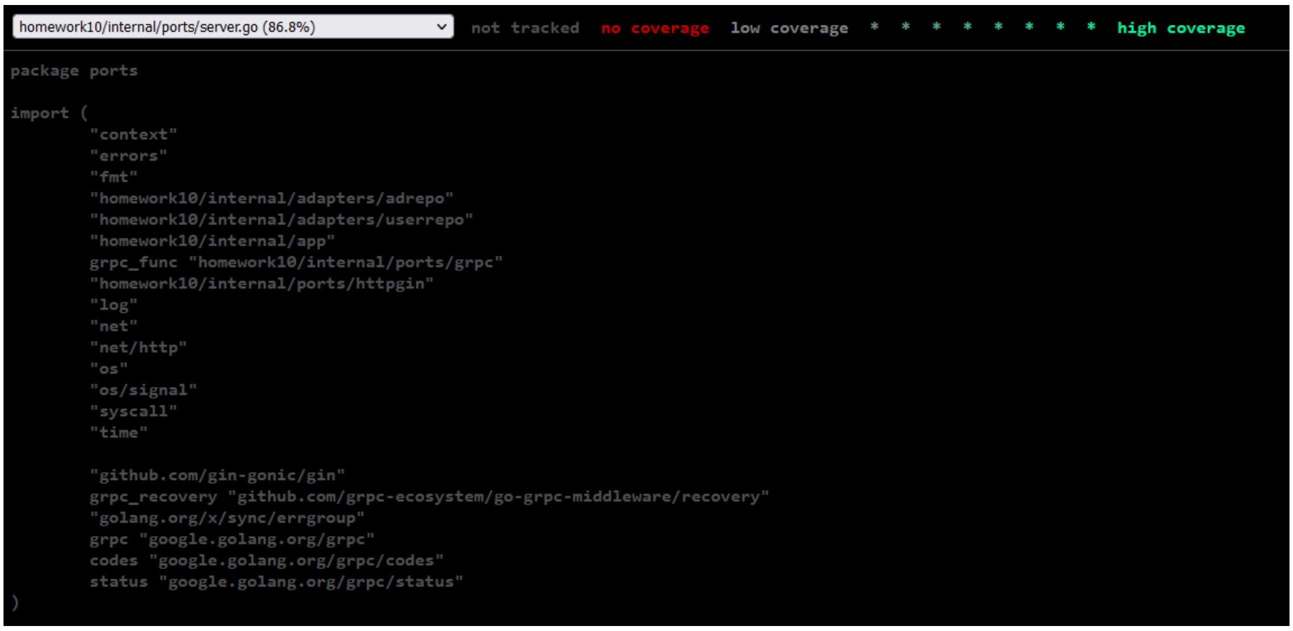
```go
func FindAdByTitle(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		title := c.Query("title")
		if title == "" {
			c.Status(http.StatusBadRequest)
			return
		}
		c.JSON(http.StatusOK, adsResponse{a.FindByTitle(title)})
	}

	return gin.HandlerFunc(fn)
}

func GetUserByID(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		id, err := strconv.Atoi(c.Param("id"))
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		usr, err := a.GetUserByID(int64(id))
		if err != nil {
			c.Status(http.StatusNotFound)
			return
		}
		c.JSON(http.StatusOK, userResponse{*usr})
	}

	return gin.HandlerFunc(fn)
}

func DeleteUserByID(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		id, err := strconv.Atoi(c.Param("id"))
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		usr, err := a.DeleteUser(int64(id))
		if err != nil {
			c.Status(http.StatusNotFound)
			return
		}
		c.JSON(http.StatusOK, userResponse{*usr})
	}

	return gin.HandlerFunc(fn)
}

func DeleteAdByID(a app.App) gin.HandlerFunc {
	fn := func(c *gin.Context) {
		id, err := strconv.Atoi(c.Param("id"))
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		author := c.Query("author")
		if author == "" {
			c.Status(http.StatusBadRequest)
			return
		}
		authorID, err := strconv.Atoi(author)
		if err != nil {
			c.Status(http.StatusBadRequest)
			return
		}
		ad, err := a.DeleteAd(int64(id), int64(authorID))
		if err != nil {
			if err == app.ErrForbidden {
				c.Status(http.StatusForbidden)
			} else if err == app.ErrBadRequest {
				c.Status(http.StatusBadRequest)
			} else {
				c.Status(http.StatusNotFound)
			}
			return
		}
		c.JSON(http.StatusOK, adResponse{*ad})
	}
	return gin.HandlerFunc(fn)
}
```

httpgin/router

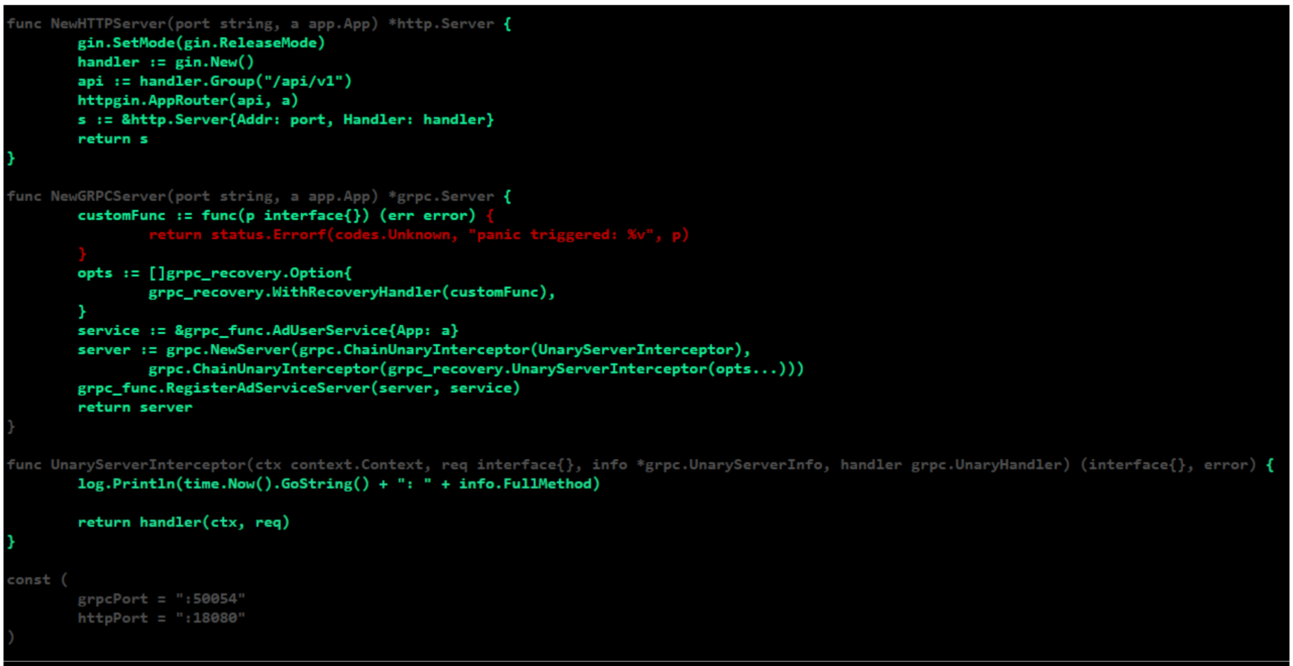

homework10/internal/ports/httpgin/router.go (78.3%)
not tracked no coverage low coverage * * * * * * * * high coverage

```go
package httpgin

import (
	"homework10/internal/app"
	"log"
	"net/http"
	"runtime"
	"time"

	"github.com/gin-gonic/gin"
)

func AppRouter(r *gin.RouterGroup, a app.App) {

	r.Use(gin.CustomRecovery(CustomPanicRecover))
	r.Use(CustomLogger)

	r.GET("/ads/:id", GetAdByID(a))
	r.POST("/ads", CreateAd(a))
	r.PUT("/ads/:id/status", ChangeAdStatus(a))
	r.PUT("/ads/:id", UpdateAd(a))
	r.GET("/ads", Select(a))
	r.GET("/ads/title", FindAdByTitle(a))
	r.DELETE("/ads/:id", DeleteAdByID(a))

	r.POST("/users", CreateUser(a))
	r.PUT("/users/:id", UpdateUser(a))
	r.GET("/users/:id", GetUserByID(a))
	r.DELETE("/users/:id", DeleteUserByID(a))

}

func CustomLogger(c *gin.Context) {
	t := time.Now().UTC()
	c.Next()
	latency := time.Since(t)
	status := c.Writer.Status()

	log.Println("latency", latency, "method", c.Request.Method, "path", c.Request.URL.Path, "status", status)
}

func CustomPanicRecover(c *gin.Context, err any) {
	log.Println("panic: " + err.(error).Error())
	buf := make([]byte, 2048)
	n := runtime.Stack(buf, false)
	log.Println(string(buf[:n]))
	c.AbortWithStatusJSON(http.StatusInternalServerError, err.(error).Error())
}
```

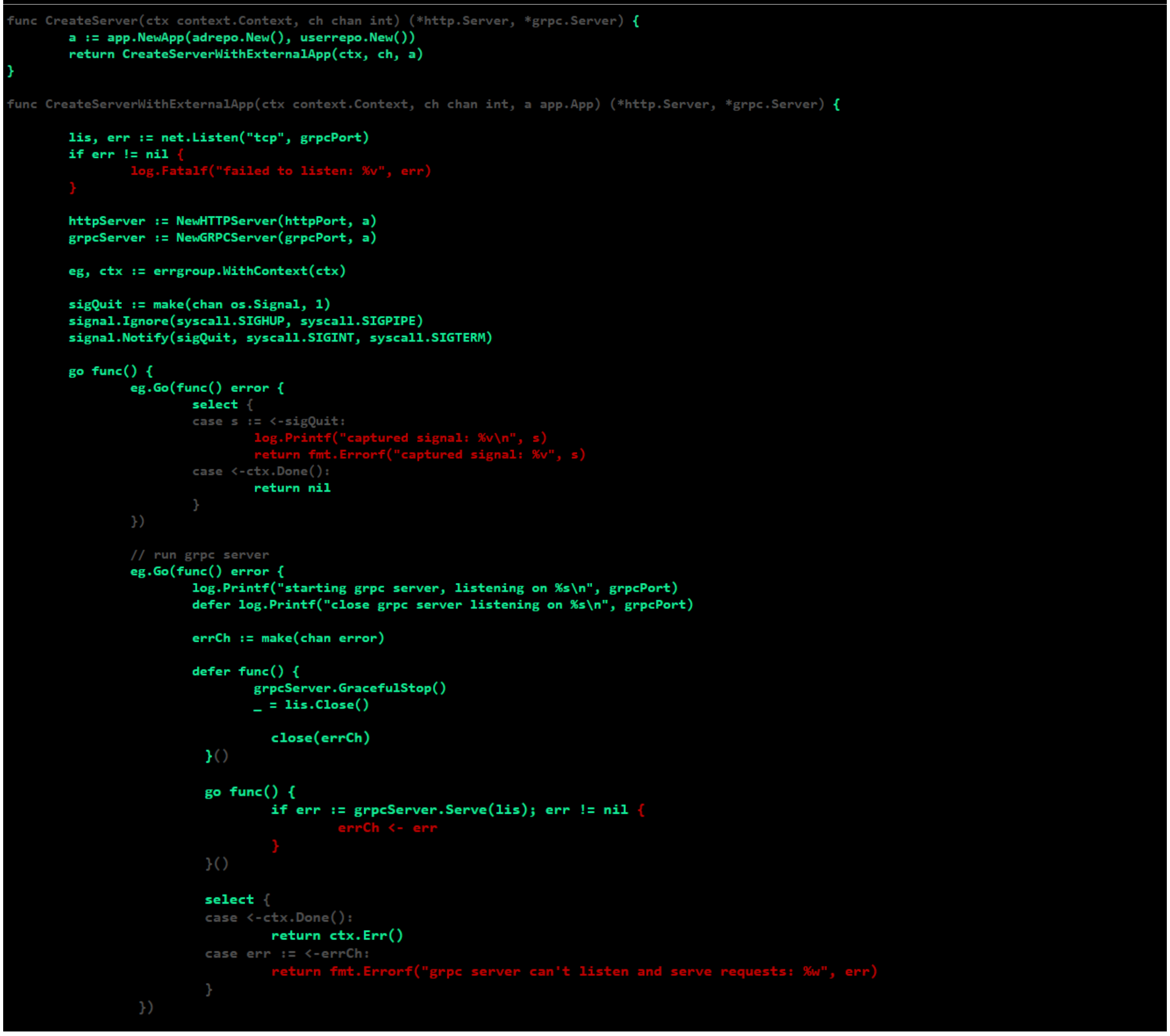
server


homework10/internal/ports/server.go (86.8%)
not tracked no coverage low coverage * * * * * * * * high coverage

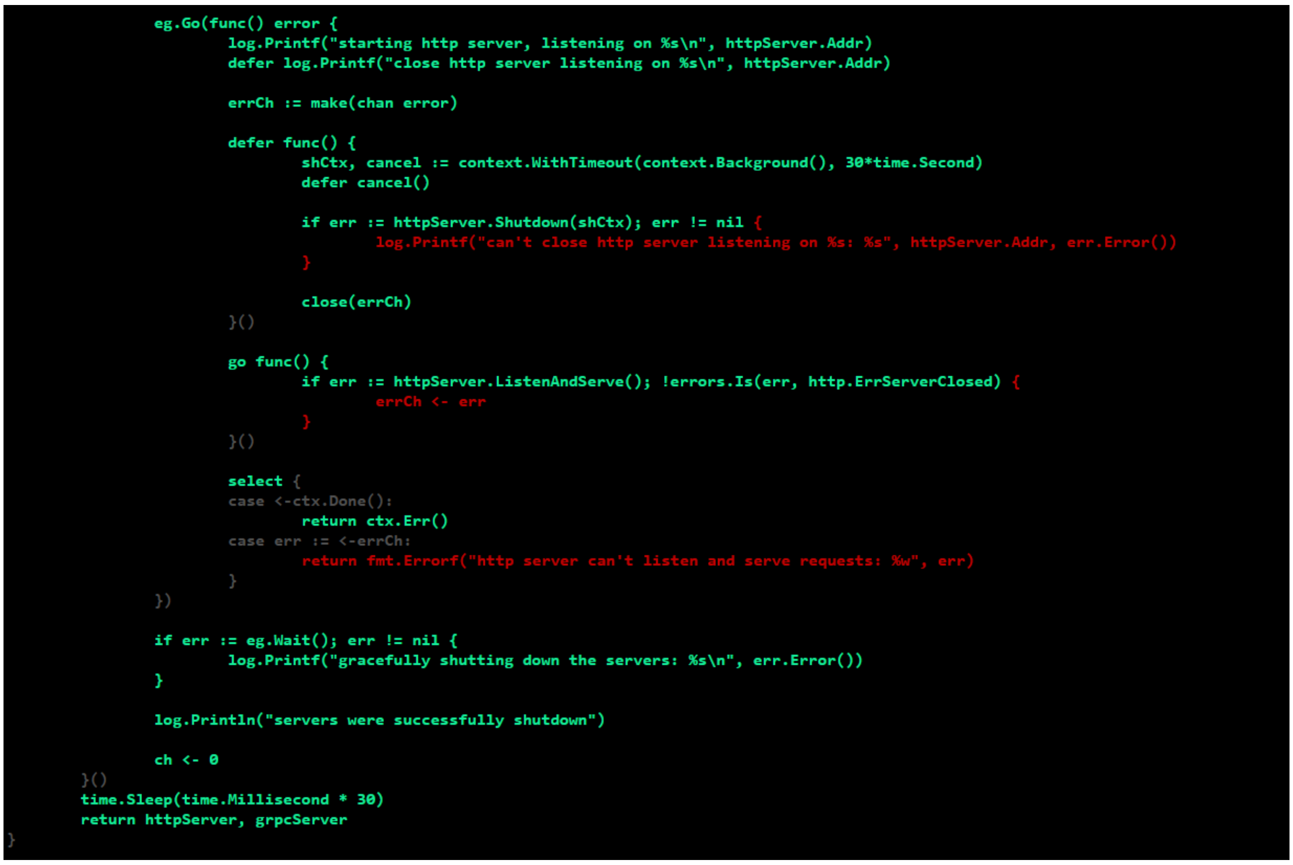
```go
package ports

import (
	"context"
	"errors"
	"fmt"
	"homework10/internal/adapters/adrepo"
	"homework10/internal/adapters/userrepo"
	"homework10/internal/app"
	grpc_func "homework10/internal/ports/grpc"
	"homework10/internal/ports/httpgin"
	"log"
	"net"
	"net/http"
	"os"
	"os/signal"
	"syscall"
	"time"

	"github.com/gin-gonic/gin"
	grpc_recovery "github.com/grpc-ecosystem/go-grpc-middleware/recovery"
	"golang.org/x/sync/errgroup"
	grpc "google.golang.org/grpc"
	codes "google.golang.org/grpc/codes"
	status "google.golang.org/grpc/status"
)
```



```go
func NewHTTPServer(port string, a app.App) *http.Server {
	gin.SetMode(gin.ReleaseMode)
	handler := gin.New()
	api := handler.Group("/api/v1")
	httpgin.AppRouter(api, a)
	s := &http.Server{Addr: port, Handler: handler}
	return s
}

func NewGRPCServer(port string, a app.App) *grpc.Server {
	customFunc := func(p interface{}) (err error) {
		return status.Errorf(codes.Unknown, "panic triggered: %v", p)
	}
	opts := []grpc_recovery.Option{
		grpc_recovery.WithRecoveryHandler(customFunc),
	}
	service := &grpc_func.AdUserService{App: a}
	server := grpc.NewServer(grpc.ChainUnaryInterceptor(UnaryServerInterceptor),
		grpc.ChainUnaryInterceptor(grpc_recovery.UnaryServerInterceptor(opts...)))
	grpc_func.RegisterAdServiceServer(server, service)
	return server
}

func UnaryServerInterceptor(ctx context.Context, req interface{}, info *grpc.UnaryServerInfo, handler grpc.UnaryHandler) (interface{}, error) {
	log.Println(time.Now().GoString() + ": " + info.FullMethod)

	return handler(ctx, req)
}

const (
	grpcPort = ":50054"
	httpPort = ":18080"
)
```

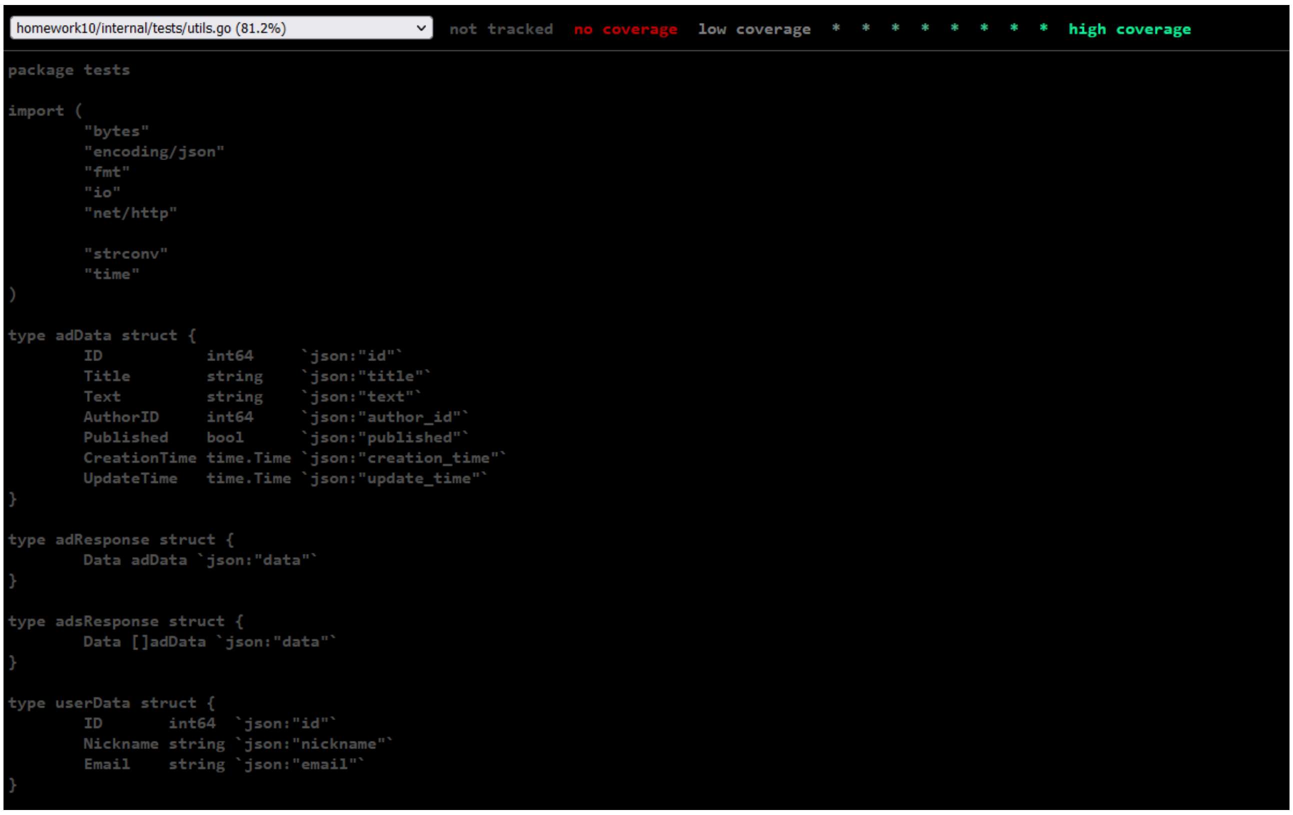


```go
func CreateServer(ctx context.Context, ch chan int) (*http.Server, *grpc.Server) {
	a := app.NewApp(adrepo.New(), userrepo.New())
	return CreateServerWithExternalApp(ctx, ch, a)
}

func CreateServerWithExternalApp(ctx context.Context, ch chan int, a app.App) (*http.Server, *grpc.Server) {

	lis, err := net.Listen("tcp", grpcPort)
	if err != nil {
		log.Fatalf("failed to listen: %v", err)
	}

	httpServer := NewHTTPServer(httpPort, a)
	grpcServer := NewGRPCServer(grpcPort, a)

	eg, ctx := errgroup.WithContext(ctx)

	sigQuit := make(chan os.Signal, 1)
	signal.Ignore(syscall.SIGHUP, syscall.SIGPIPE)
	signal.Notify(sigQuit, syscall.SIGINT, syscall.SIGTERM)

	go func() {
		eg.Go(func() error {
			select {
			case s := <-sigQuit:
				log.Printf("captured signal: %v\n", s)
				return fmt.Errorf("captured signal: %v", s)
			case <-ctx.Done():
				return nil
			}
		})

		// run grpc server
		eg.Go(func() error {
			log.Printf("starting grpc server, listening on %s\n", grpcPort)
			defer log.Printf("close grpc server listening on %s\n", grpcPort)

			errCh := make(chan error)

			defer func() {
				grpcServer.GracefulStop()
				_ = lis.Close()

				  close(errCh)
			  }()

			  go func() {
				  if err := grpcServer.Serve(lis); err != nil {
					  errCh <- err
				  }
			  }()

			  select {
			  case <-ctx.Done():
				  return ctx.Err()
			  case err := <-errCh:
				  return fmt.Errorf("grpc server can't listen and serve requests: %w", err)
			  }
		 })
```



```go
		eg.Go(func() error {
			log.Printf("starting http server, listening on %s\n", httpServer.Addr)
			defer log.Printf("close http server listening on %s\n", httpServer.Addr)

			errCh := make(chan error)

			defer func() {
				shCtx, cancel := context.WithTimeout(context.Background(), 30*time.Second)
				defer cancel()

				if err := httpServer.Shutdown(shCtx); err != nil {
					log.Printf("can't close http server listening on %s: %s", httpServer.Addr, err.Error())
				}

				close(errCh)
			}()

			go func() {
				if err := httpServer.ListenAndServe(); !errors.Is(err, http.ErrServerClosed) {
					errCh <- err
				}
			}()

			select {
			case <-ctx.Done():
				return ctx.Err()
			case err := <-errCh:
				return fmt.Errorf("http server can't listen and serve requests: %w", err)
			}
		})

		if err := eg.Wait(); err != nil {
			log.Printf("gracefully shutting down the servers: %s\n", err.Error())
		}

		log.Println("servers were successfully shutdown")

		ch <- 0
	}()
	time.Sleep(time.Millisecond * 30)
	return httpServer, grpcServer
}
```

utils



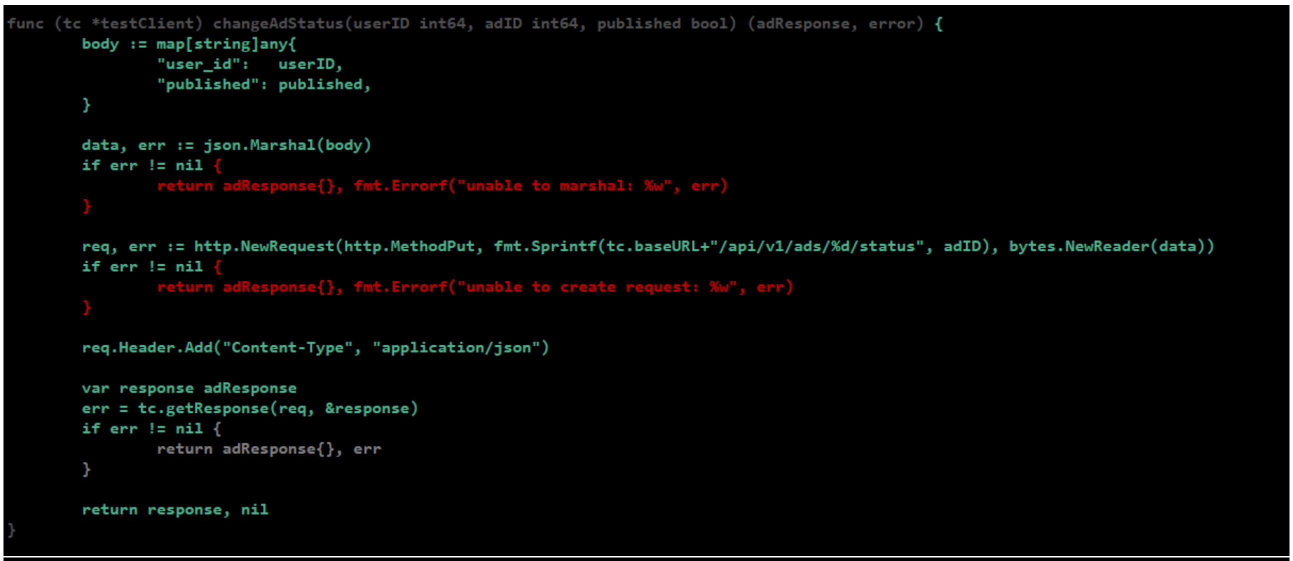
homework10/internal/tests/utils.go (81.2%)   not tracked no coverage low coverage * * * * * * * * high coverage

```go
package tests

import (
	"bytes"
	"encoding/json"
	"fmt"
	"io"
	"net/http"

	"strconv"
	"time"
)

type adData struct {
	ID           int64     `json:"id"`
	Title        string    `json:"title"`
	Text         string    `json:"text"`
	AuthorID     int64     `json:"author_id"`
	Published    bool      `json:"published"`
	CreationTime time.Time `json:"creation_time"`
	UpdateTime   time.Time `json:"update_time"`
}

type adResponse struct {
	Data adData `json:"data"`
}

type adsResponse struct {
	Data []adData `json:"data"`
}

type userData struct {
	ID       int64  `json:"id"`
	Nickname string `json:"nickname"`
	Email    string `json:"email"`
}
```

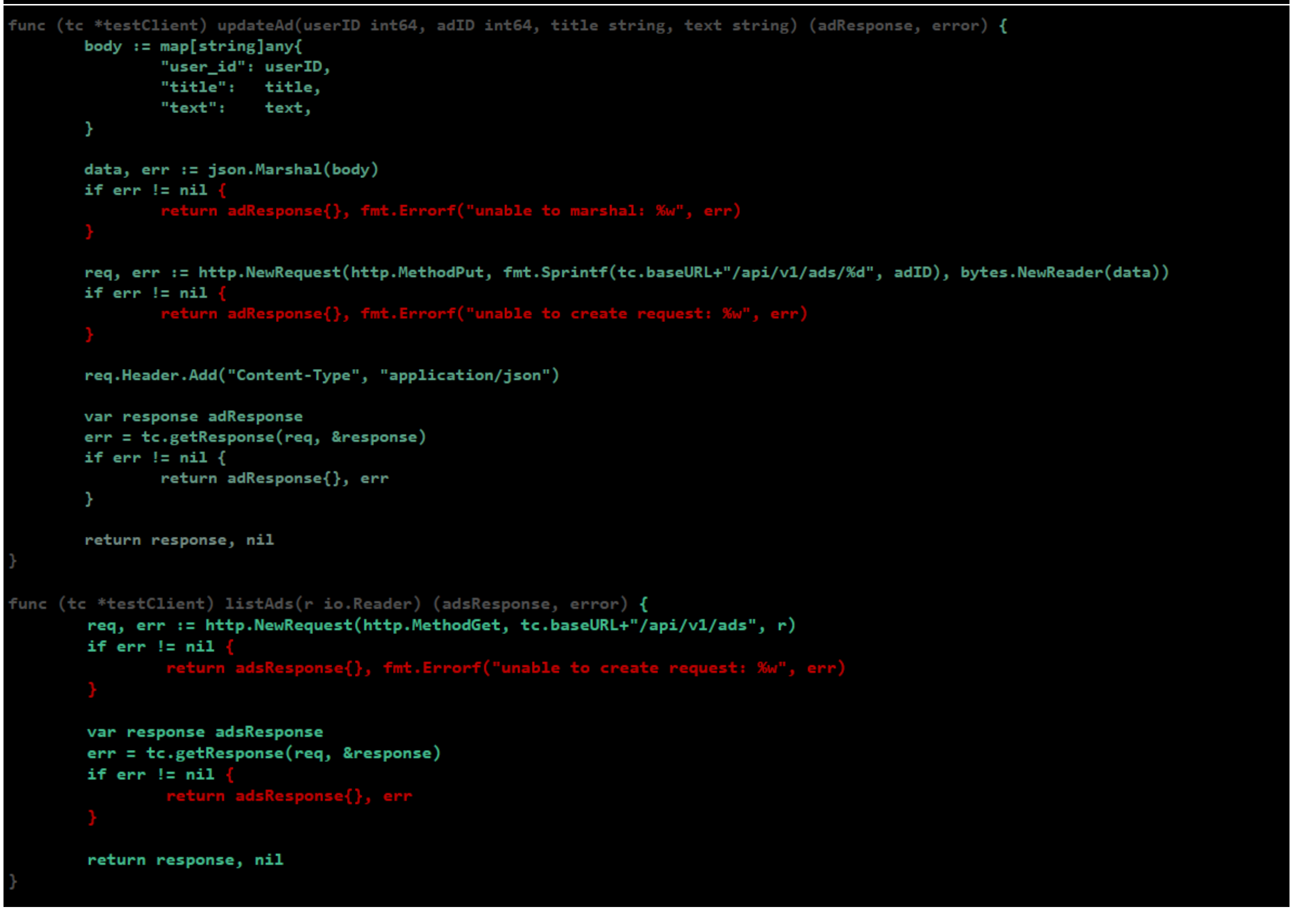
```go
type userResponse struct {
	Data userData `json:"data"`
}

var (
	ErrBadRequest = fmt.Errorf("bad request")
	ErrForbidden  = fmt.Errorf("forbidden")
	ErrNotFound   = fmt.Errorf("not found")
)

type testClient struct {
	baseURL string
}

func getTestClient(adr string) *testClient {
	return &testClient{
		baseURL: "http://localhost" + adr,
	}
}
```

```go
func (tc *testClient) getResponse(req *http.Request, out any) error {
	client := &http.Client{}
	resp, err := client.Do(req)
	if err != nil {
		return fmt.Errorf("unexpected error: %w", err)
	}

	if resp.StatusCode != http.StatusOK {
		if resp.StatusCode == http.StatusBadRequest {
			return ErrBadRequest
		}
		if resp.StatusCode == http.StatusForbidden {
			return ErrForbidden
		}
		if resp.StatusCode == http.StatusNotFound {
			return ErrNotFound
		}
		return fmt.Errorf("unexpected status code: %s", resp.Status)
	}

	respBody, err := io.ReadAll(resp.Body)
	if err != nil {
		return fmt.Errorf("unable to read response: %w", err)
	}

	err = json.Unmarshal(respBody, out)
	if err != nil {
		return fmt.Errorf("unable to unmarshal: %w", err)
	}

	return nil
}
```

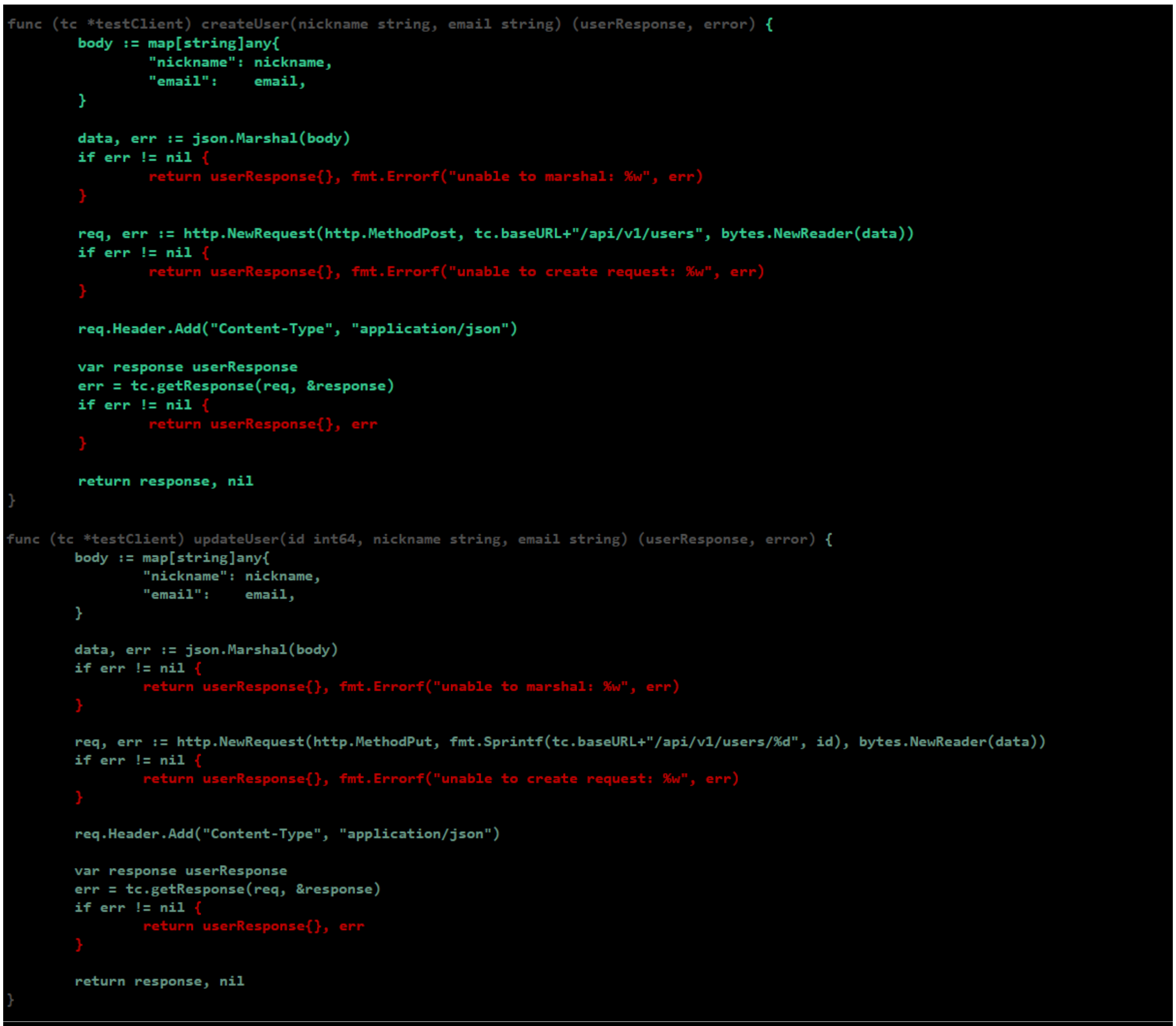
```go
func (tc *testClient) createAd(userID int64, title string, text string) (adResponse, error) {
	body := map[string]any{
		"user_id": userID,
		"title":   title,
		"text":    text,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	req, err := http.NewRequest(http.MethodPost, tc.baseURL+"/api/v1/ads", bytes.NewReader(data))
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response adResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return adResponse{}, err
	}

	return response, nil
}
```



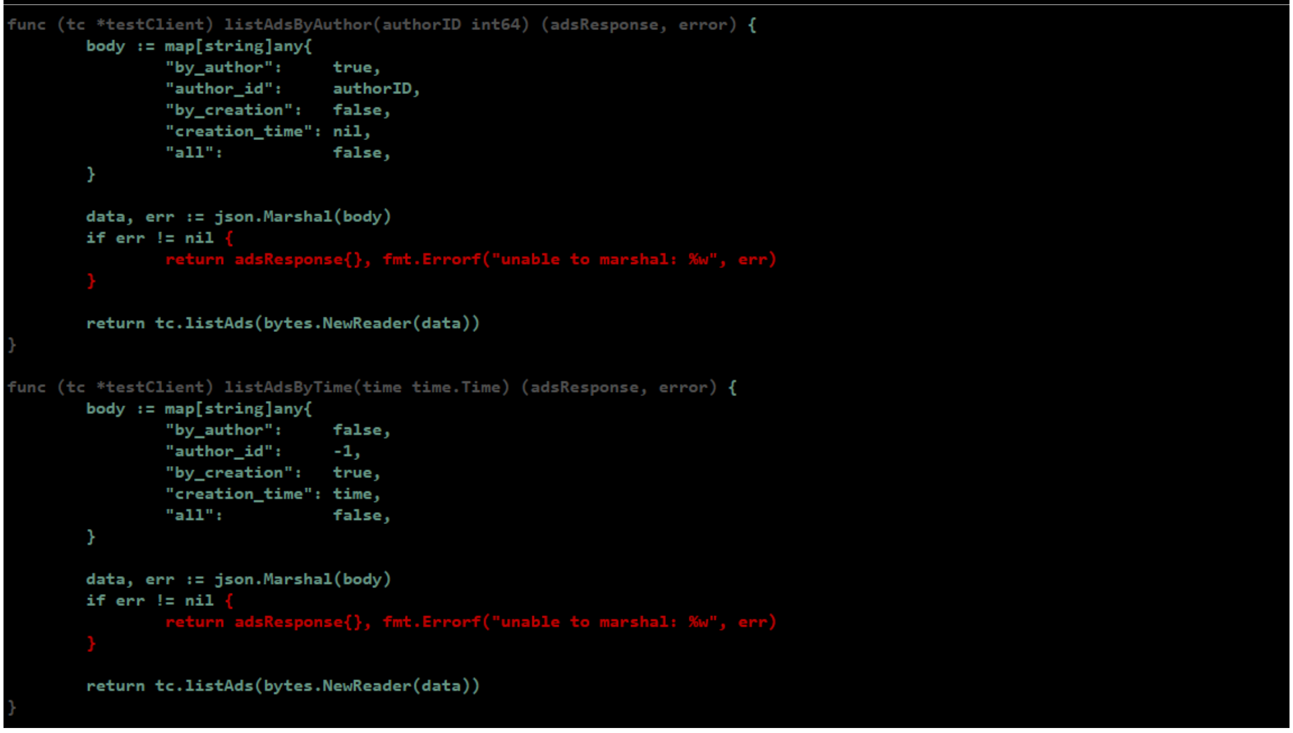
```go
func (tc *testClient) changeAdStatus(userID int64, adID int64, published bool) (adResponse, error) {
	body := map[string]any{
		"user_id":   userID,
		"published": published,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	req, err := http.NewRequest(http.MethodPut, fmt.Sprintf(tc.baseURL+"/api/v1/ads/%d/status", adID), bytes.NewReader(data))
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response adResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return adResponse{}, err
	}

	return response, nil
}
```



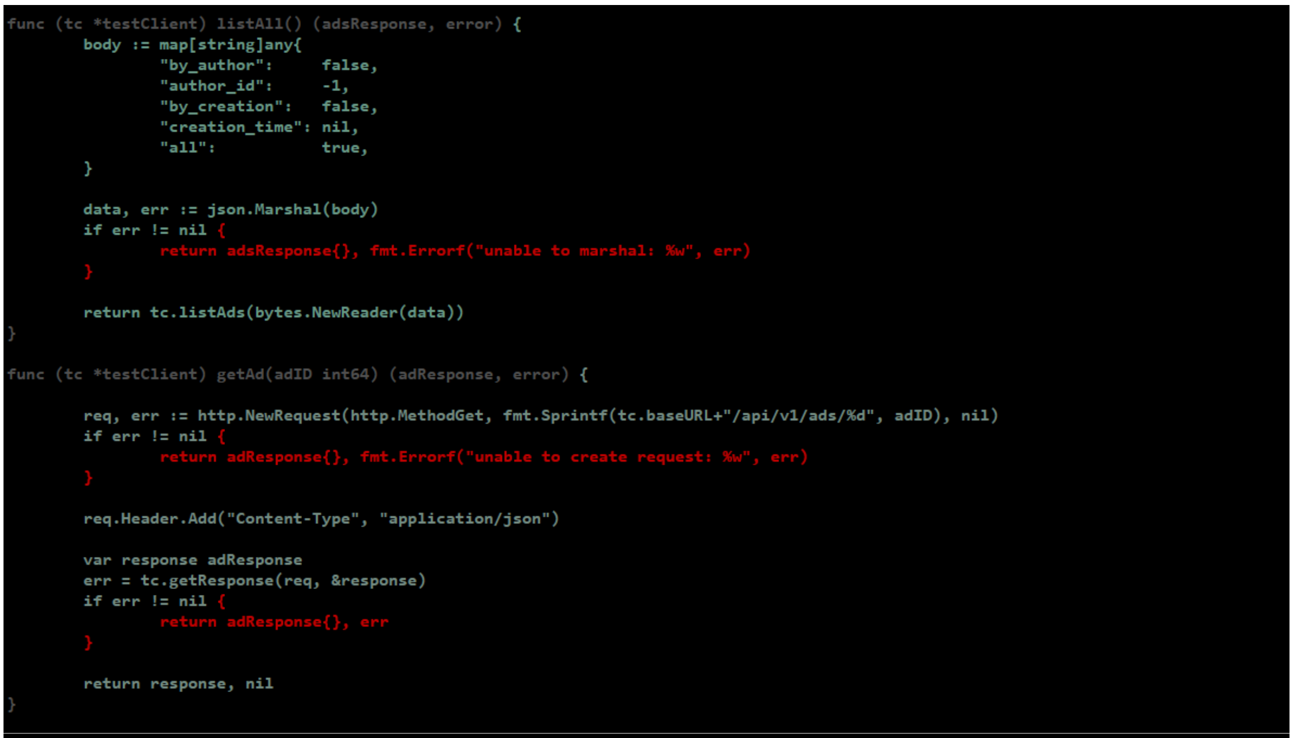
```go
func (tc *testClient) updateAd(userID int64, adID int64, title string, text string) (adResponse, error) {
	body := map[string]any{
		"user_id": userID,
		"title":   title,
		"text":    text,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	req, err := http.NewRequest(http.MethodPut, fmt.Sprintf(tc.baseURL+"/api/v1/ads/%d", adID), bytes.NewReader(data))
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response adResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return adResponse{}, err
	}

	return response, nil
}

func (tc *testClient) listAds(r io.Reader) (adsResponse, error) {
	req, err := http.NewRequest(http.MethodGet, tc.baseURL+"/api/v1/ads", r)
	if err != nil {
		return adsResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	var response adsResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return adsResponse{}, err
	}

	return response, nil
}
```



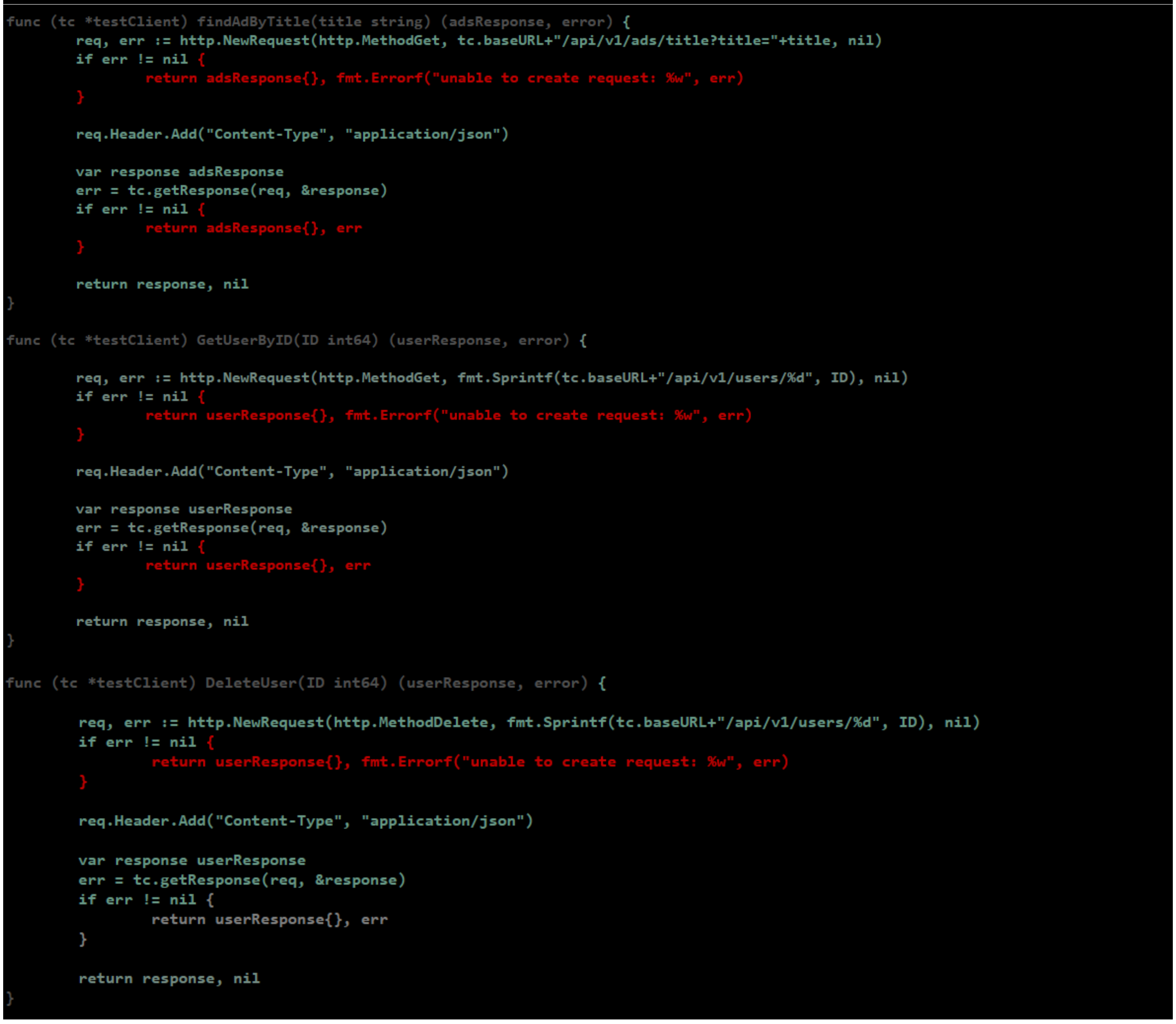
```go
func (tc *testClient) createUser(nickname string, email string) (userResponse, error) {
	body := map[string]any{
		"nickname": nickname,
		"email":    email,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return userResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	req, err := http.NewRequest(http.MethodPost, tc.baseURL+"/api/v1/users", bytes.NewReader(data))
	if err != nil {
		return userResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response userResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return userResponse{}, err
	}

	return response, nil
}

func (tc *testClient) updateUser(id int64, nickname string, email string) (userResponse, error) {
	body := map[string]any{
		"nickname": nickname,
		"email":    email,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return userResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	req, err := http.NewRequest(http.MethodPut, fmt.Sprintf(tc.baseURL+"/api/v1/users/%d", id), bytes.NewReader(data))
	if err != nil {
		return userResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response userResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return userResponse{}, err
	}

	return response, nil
}
```



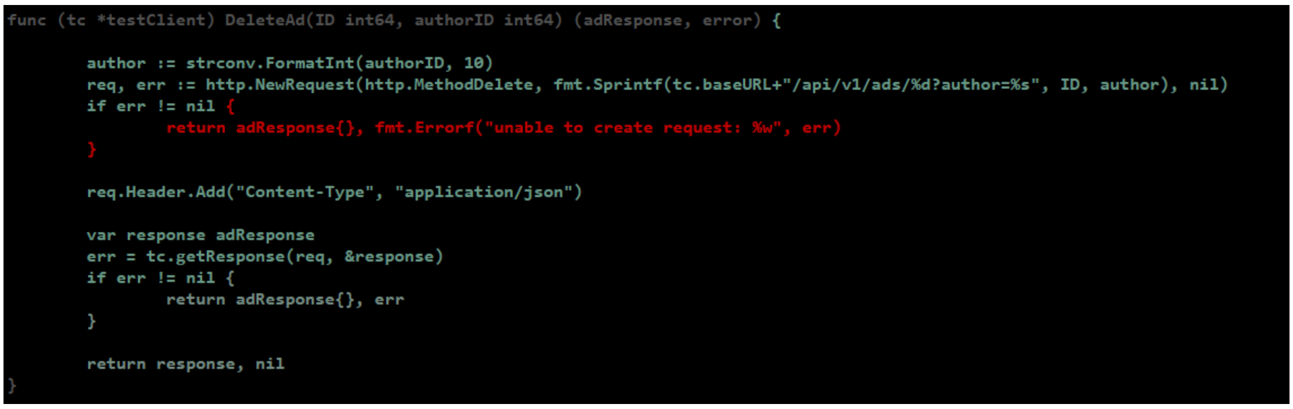
```go
func (tc *testClient) listAdsByAuthor(authorID int64) (adsResponse, error) {
	body := map[string]any{
		"by_author":     true,
		"author_id":     authorID,
		"by_creation":   false,
		"creation_time": nil,
		"all":           false,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return adsResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	return tc.listAds(bytes.NewReader(data))
}

func (tc *testClient) listAdsByTime(time time.Time) (adsResponse, error) {
	body := map[string]any{
		"by_author":     false,
		"author_id":     -1,
		"by_creation":   true,
		"creation_time": time,
		"all":           false,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return adsResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	return tc.listAds(bytes.NewReader(data))
}
```



```go
func (tc *testClient) listAll() (adsResponse, error) {
	body := map[string]any{
		"by_author":     false,
		"author_id":     -1,
		"by_creation":   false,
		"creation_time": nil,
		"all":           true,
	}

	data, err := json.Marshal(body)
	if err != nil {
		return adsResponse{}, fmt.Errorf("unable to marshal: %w", err)
	}

	return tc.listAds(bytes.NewReader(data))
}

func (tc *testClient) getAd(adID int64) (adResponse, error) {

	req, err := http.NewRequest(http.MethodGet, fmt.Sprintf(tc.baseURL+"/api/v1/ads/%d", adID), nil)
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response adResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return adResponse{}, err
	}

	return response, nil
}
```

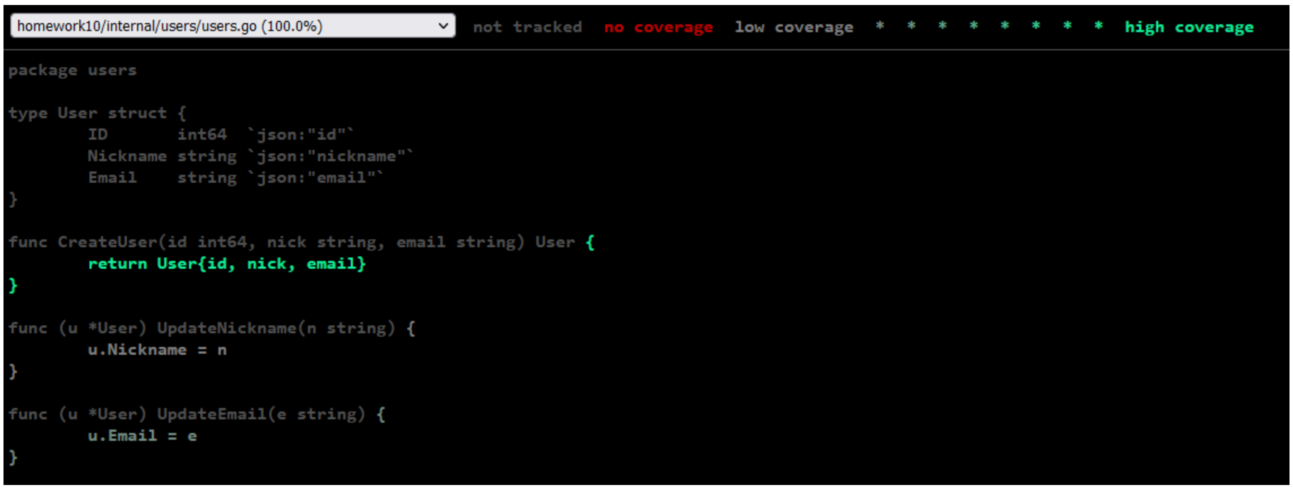


```go
func (tc *testClient) findAdByTitle(title string) (adsResponse, error) {
	req, err := http.NewRequest(http.MethodGet, tc.baseURL+"/api/v1/ads/title?title="+title, nil)
	if err != nil {
		return adsResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response adsResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return adsResponse{}, err
	}

	return response, nil
}

func (tc *testClient) GetUserByID(ID int64) (userResponse, error) {

	req, err := http.NewRequest(http.MethodGet, fmt.Sprintf(tc.baseURL+"/api/v1/users/%d", ID), nil)
	if err != nil {
		return userResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response userResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return userResponse{}, err
	}

	return response, nil
}

func (tc *testClient) DeleteUser(ID int64) (userResponse, error) {

	req, err := http.NewRequest(http.MethodDelete, fmt.Sprintf(tc.baseURL+"/api/v1/users/%d", ID), nil)
	if err != nil {
		return userResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response userResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return userResponse{}, err
	}

	return response, nil
}
```



```
func (tc *testClient) DeleteAd(ID int64, authorID int64) (adResponse, error) {

	author := strconv.FormatInt(authorID, 10)
	req, err := http.NewRequest(http.MethodDelete, fmt.Sprintf(tc.baseURL+"/api/v1/ads/%d?author=%s", ID, author), nil)
	if err != nil {
		return adResponse{}, fmt.Errorf("unable to create request: %w", err)
	}

	req.Header.Add("Content-Type", "application/json")

	var response adResponse
	err = tc.getResponse(req, &response)
	if err != nil {
		return adResponse{}, err
	}

	return response, nil
}
```

users



```
package users

type User struct {
	ID       int64  `json:"id"`
	Nickname string `json:"nickname"`
	Email    string `json:"email"`
}

func CreateUser(id int64, nick string, email string) User {
	return User{id, nick, email}
}

func (u *User) UpdateNickname(n string) {
	u.Nickname = n
}

func (u *User) UpdateEmail(e string) {
	u.Email = e
}
```