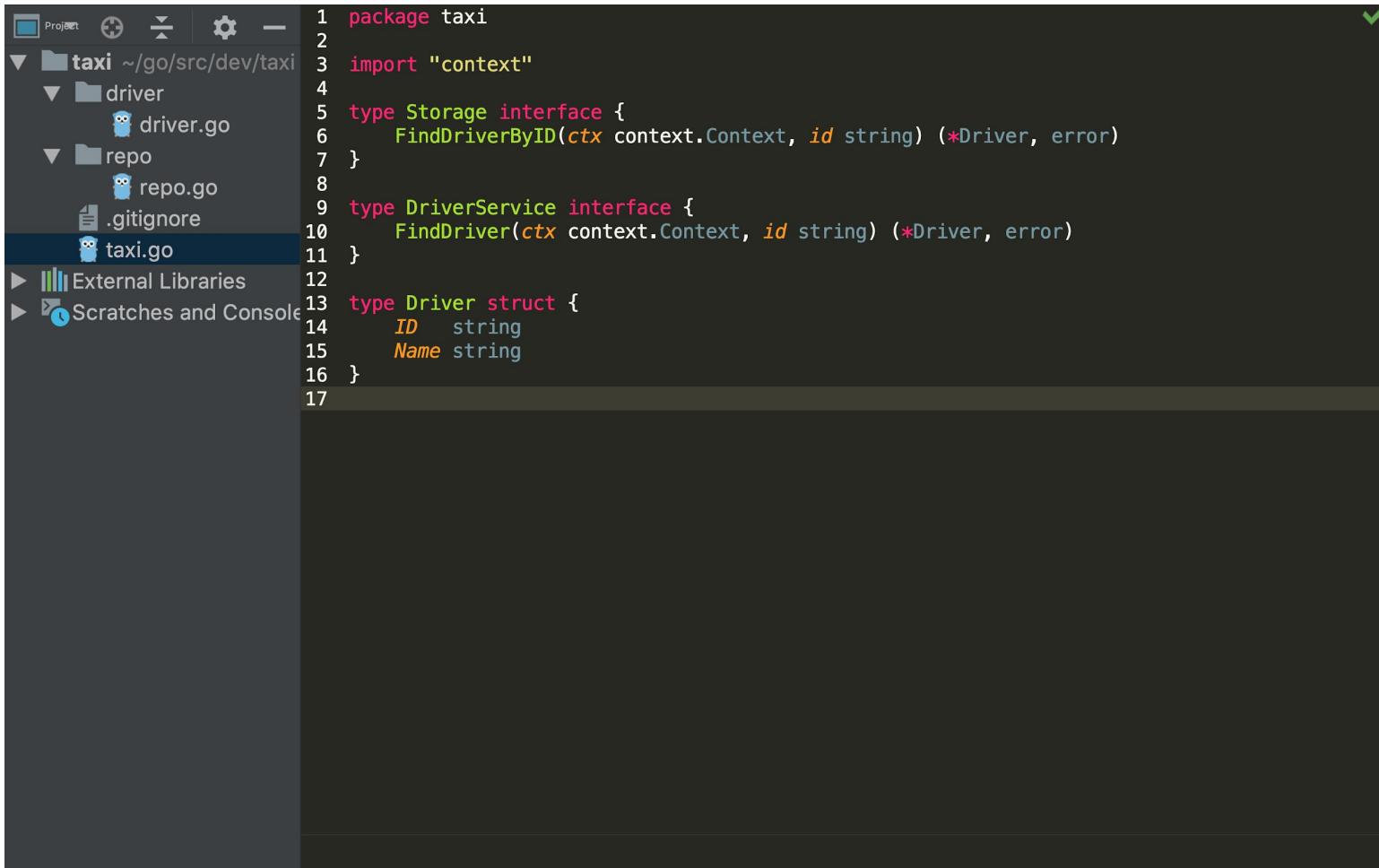


Golang generator of client to sql database based on interface

Zaur Abasmirzoev | 27.06.2019

Repository pattern



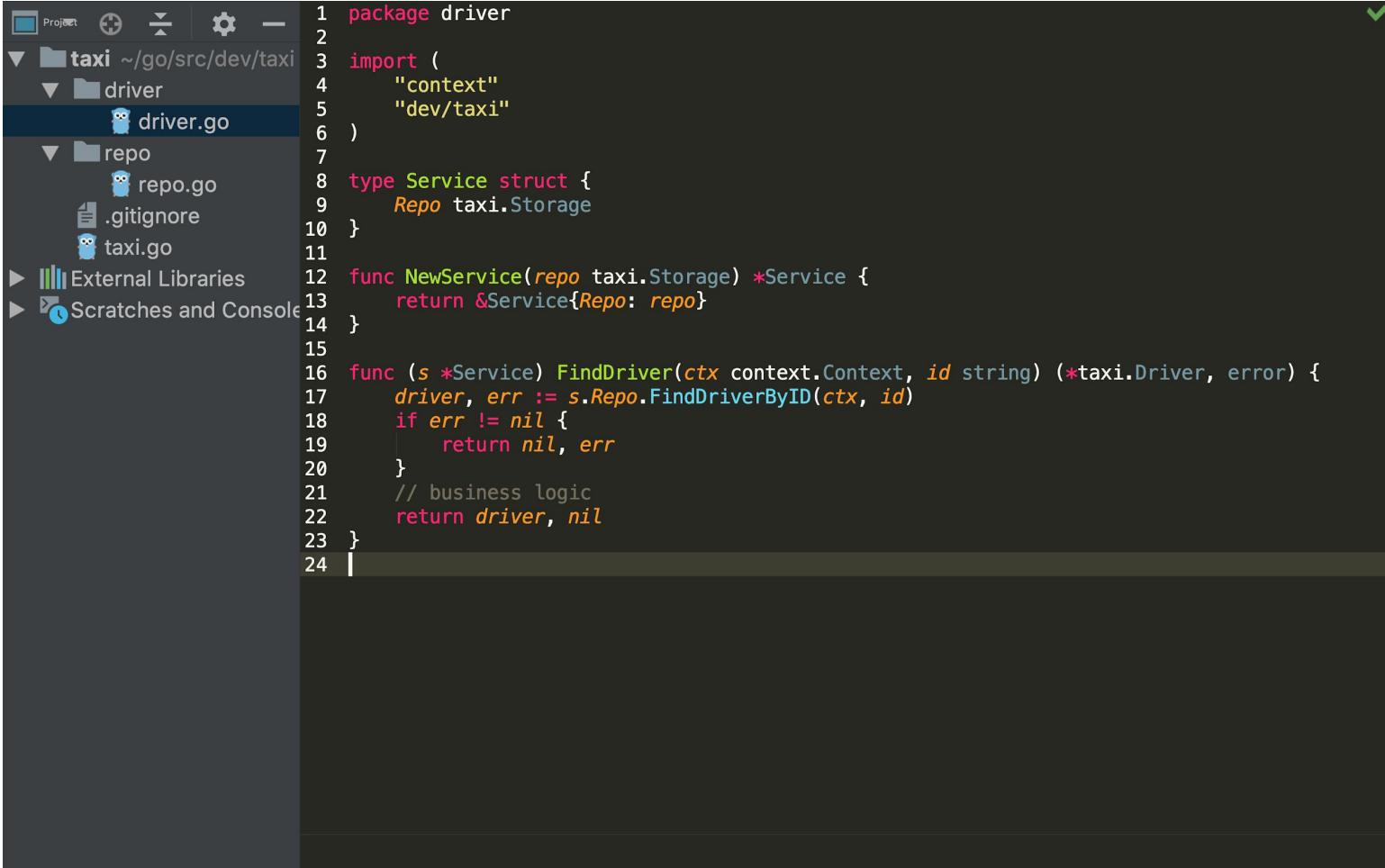
The screenshot shows a Go code editor interface with a dark theme. On the left is a project navigation sidebar containing:

- Project icon
- taxi (~/go/src/dev/taxi)
- driver (with driver.go)
- repo (with repo.go)
- .gitignore
- taxi.go (selected)
- External Libraries
- Scratches and Console

The main editor area displays the following Go code:

```
1 package taxi
2
3 import "context"
4
5 type Storage interface {
6     FindDriverByID(ctx context.Context, id string) (*Driver, error)
7 }
8
9 type DriverService interface {
10    FindDriver(ctx context.Context, id string) (*Driver, error)
11 }
12
13 type Driver struct {
14     ID   string
15     Name string
16 }
17
```

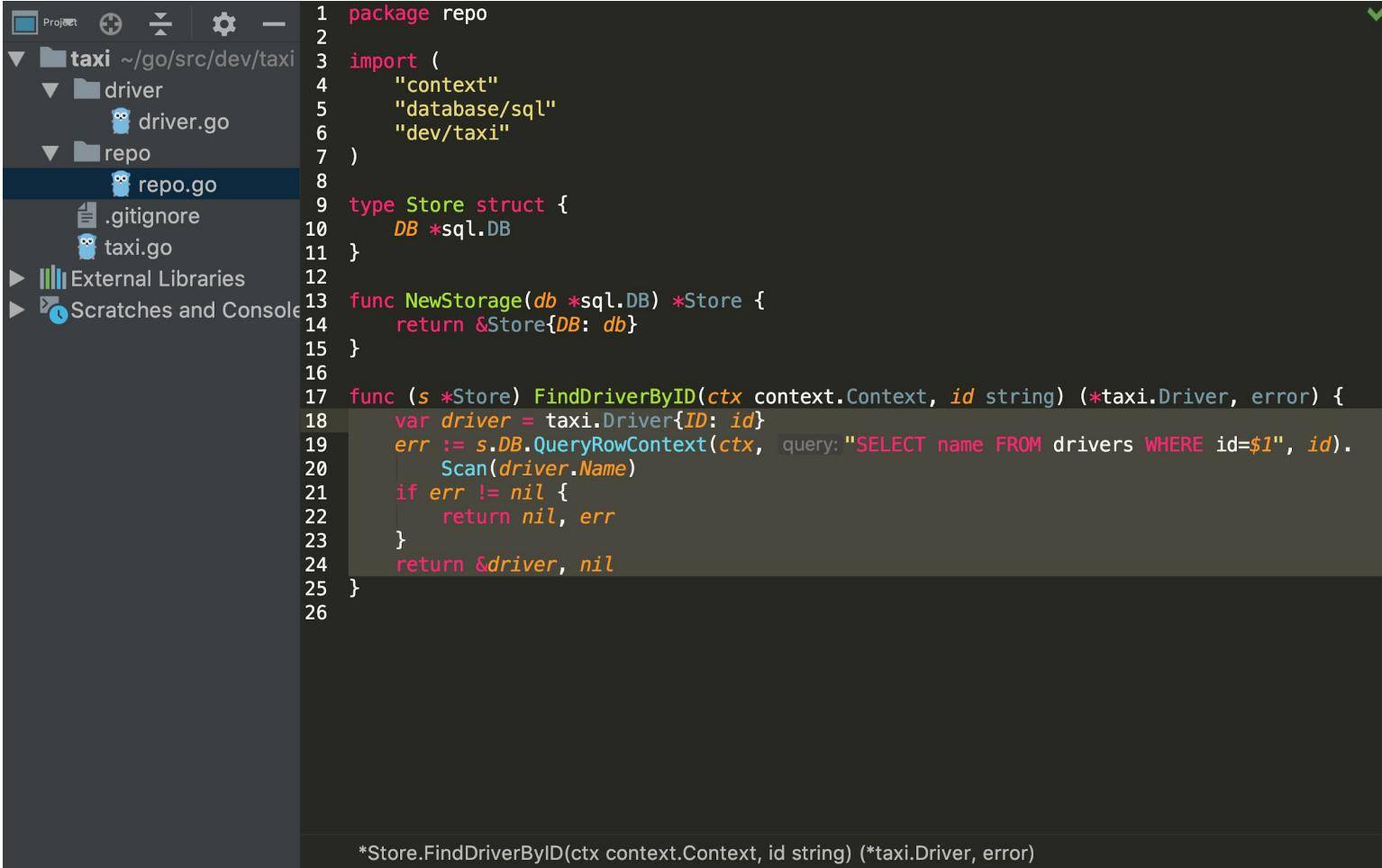
Repository pattern



The screenshot shows a code editor interface with a dark theme. On the left is a project navigation sidebar containing a 'taxi' folder with subfolders 'driver' and 'repo', and files 'driver.go', 'repo.go', '.gitignore', and 'taxi.go'. Below this are sections for 'External Libraries' and 'Scratches and Console'.

```
1 package driver
2
3 import (
4     "context"
5     "dev/taxi"
6 )
7
8 type Service struct {
9     Repo taxi.Storage
10 }
11
12 func NewService(repo taxi.Storage) *Service {
13     return &Service{Repo: repo}
14 }
15
16 func (s *Service) FindDriver(ctx context.Context, id string) (*taxi.Driver, error) {
17     driver, err := s.Repo.FindDriverByID(ctx, id)
18     if err != nil {
19         return nil, err
20     }
21     // business logic
22     return driver, nil
23 }
24 |
```

Repository pattern



```
1 package repo
2
3 import (
4     "context"
5     "database/sql"
6     "dev/taxi"
7 )
8
9 type Store struct {
10     DB *sql.DB
11 }
12
13 func NewStorage(db *sql.DB) *Store {
14     return &Store{DB: db}
15 }
16
17 func (s *Store) FindDriverByID(ctx context.Context, id string) (*taxi.Driver, error) {
18     var driver = taxi.Driver{ID: id}
19     err := s.DB.QueryRowContext(ctx, query:"SELECT name FROM drivers WHERE id=$1", id).
20             Scan(driver.Name)
21     if err != nil {
22         return nil, err
23     }
24     return &driver, nil
25 }
26
```

*Store.FindDriverByID(ctx context.Context, id string) (*taxi.Driver, error)

Requirements

1. Описание взаимодействия в виде интерфейса.
2. Интерфейс описывается методами и сообщениями запросов и ответов.

Запрос

- SQL строка "SELECT name FROM drivers WHERE id=\$1"
- Аргументы запроса
taxi.FindReq{
 ID: "d243b7e2-8833-4475-b6f7-70998918203b"
}

Ответ

```
taxi.Driver{  
    ID: "d243b7e2-8833-4475-b6f7-70998918203b",  
    Name: "Ivan Sokol",  
}
```

Requirements

3. Поддержка связывания переменных и подготовленных выражений (prepared statements).

```
stmt, err := db.PrepareContext(ctx, "SELECT name FROM drivers WHERE id=$1")
row = stmt.QueryRowContext(ctx, id)
```

Requirements

4. Поддержка именованных аргументов.

```
db.ExecContext(ctx, `DELETE FROM orders WHERE created_at < @end`,  
    sql.Named("end", endTime))
```

```
db.ExecContext(ctx, `DELETE FROM orders WHERE created_at < $1`, endTime)
```

Requirements

5. Связывание ответа базы данных с полями структуры данных сообщения.

```
zaur@taxi > SELECT * FROM drivers;
```

id	name
d243b7e2-8833-4475-b6f7-70998918203b	Ivan Sokol

(1 row)

```
type Driver struct {  
    ID string  
    Name string  
}
```

```
taxi.Driver{  
    ID: "d243b7e2-8833-4475-b6f7-70998918203b",  
    Name: "Ivan Sokol",  
}
```

Requirements

6. Поддержка нетипичных структур данных (массив, json).

```
db.Query(`SELECT * FROM t WHERE id = ANY($1)`, pq.Array([]int{235, 401}))
```

```
var x []sql.NullInt64  
db.QueryRow('SELECT ARRAY[235, 401]').Scan(pq.Array(&x))
```

Requirements

6. Прозрачная работа с транзакциями.

```
store := NewStorage(db)
```

```
tx := store.Begin()  
tx.CreateDriver(...)  
tx.UpdateOrder(...)  
tx.Commit()
```

Requirements

7. Встроенная поддержка промежуточных обработчиков (middleware).

- before/after hooks
- observability
- access to metadata

Input

- Interface
- Methods
- Arguments
- Tags

Reference library

The screenshot shows a GitHub README page for the `golang/mock` repository. The page title is `gomock`. It includes a green `build passing` badge and a blue `godoc reference` link. The text describes GoMock as a mocking framework for the Go programming language, noting its integration with Go's built-in testing package and its use in other contexts. The `Installation` section provides commands to install the `gomock` package and the `mockgen` tool. The `Documentation` section explains how to use `go doc` to get documentation, with a command example and a link to an online reference on GoPkgDoc.

gomock [build passing](#) [godoc](#) [reference](#)

GoMock is a mocking framework for the [Go programming language](#). It integrates well with Go's built-in `testing` package, but can be used in other contexts too.

Installation

Once you have [installed Go](#), run these commands to install the `gomock` package and the `mockgen` tool:

```
go get github.com/golang/mock/gomock
go install github.com/golang/mock/mockgen
```

Documentation

After installing, you can use `go doc` to get documentation:

```
go doc github.com/golang/mock/gomock
```

Alternatively, there is an online reference for the package hosted on GoPkgDoc [here](#).

Implementation

1. script input: package name, interfaces, destination file
2. build new script
 - a. run temporary script
 - b. analyse by reflect
 - c. encode (gob)
3. decode
4. generate code of client library
5. write a file

Solution: go-gad/sal

The screenshot shows the GitHub README.md page for the `go-gad/sal` repository. The page includes sections for the `sal` package, build status, documentation, and links to a Medium article and the generator's source code.

sal

Generator client to the database on the Golang based on interface.

Install

```
go get -u github.com/go-gad/sal/...
```

Usage

Read article <https://medium.com/@zaurio/generator-the-client-to-sql-database-on-golang-dccfeb4641c3>

```
salgen -h
Usage:
  salgen [options...] <import_path> <interface_name>

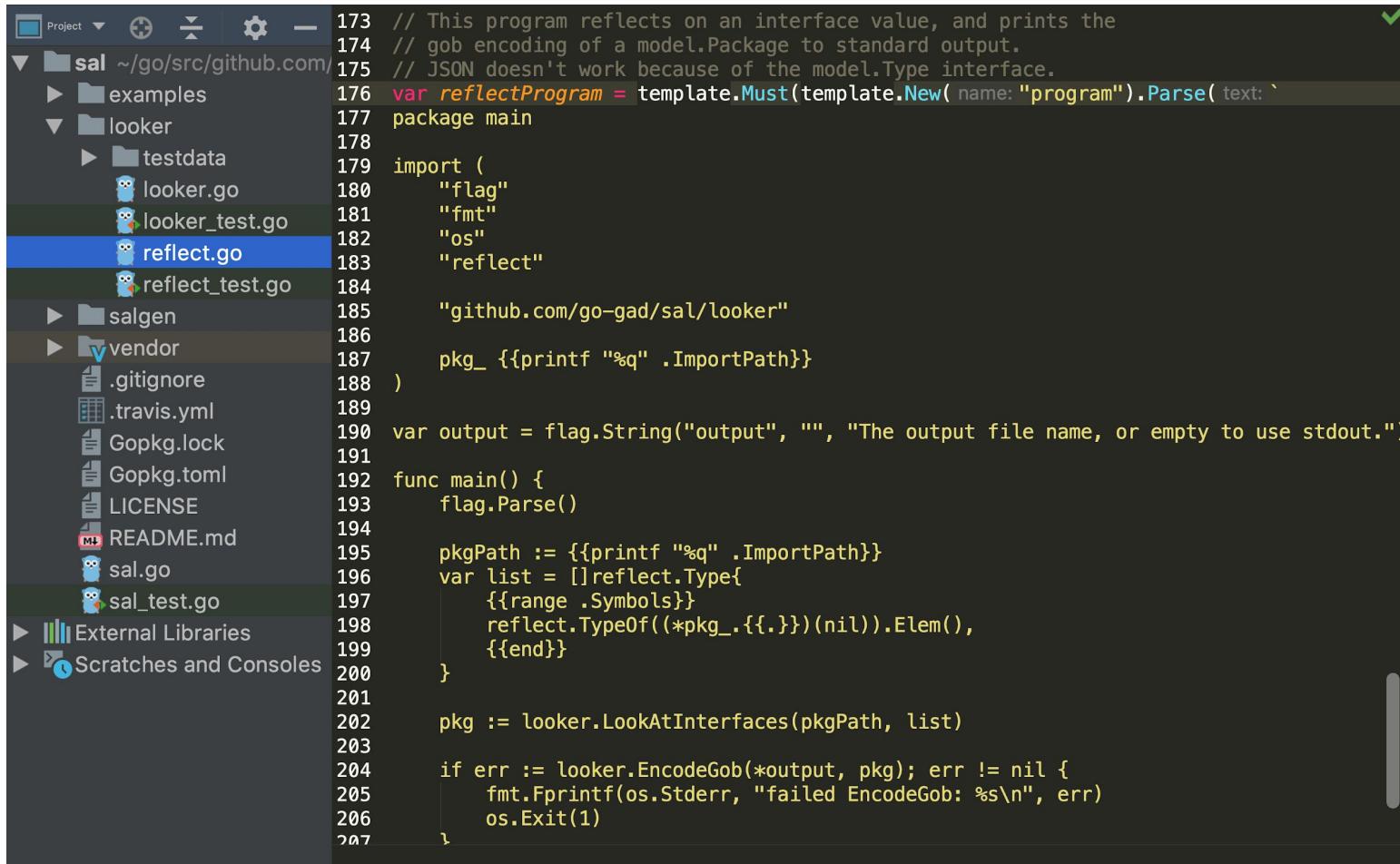
Example:
  salgen -destination=./client.go -package=github.com/go-gad/sal/examples/profile/storage github.com/go-
  <import_path>
    describes the complete package path where the interface is located.
  <interface_name>
    indicates the interface name itself.

Options:
  -build_flags string
    Additional flags for go build.
  -destination string
    Output file; defaults to stdout.
  -package string
    The full import path of the library for the generated implementation
```

With go generate:

```
//go:generate salgen -destination=./client.go -package=github.com/go-gad/sal/examples/profile/storage git
type Store interface {
  CreateUser(ctx context.Context, req *CreateUserReq) (*CreateUserResp, error)
}
```

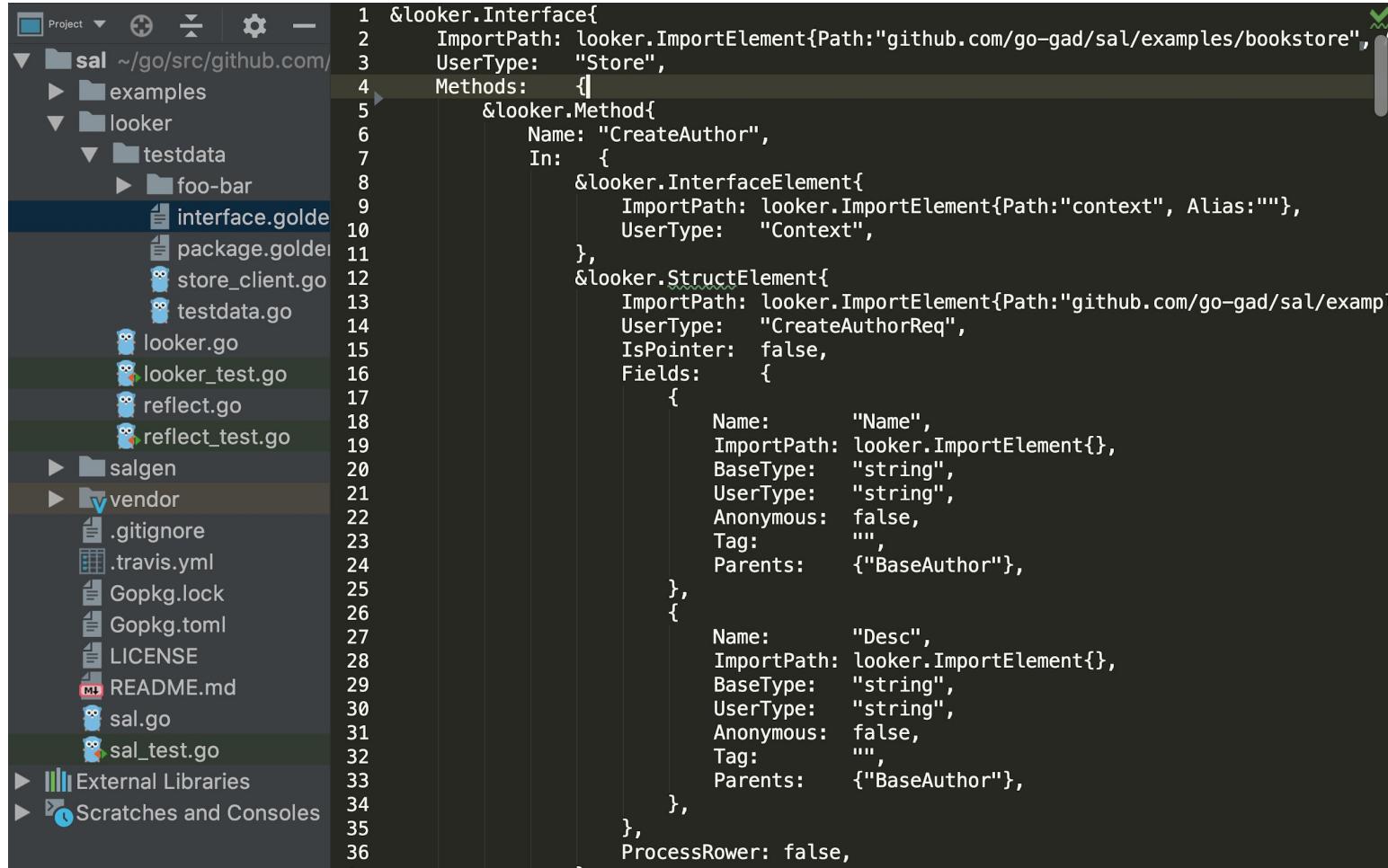
Implementation



The screenshot shows a Go code editor interface with a dark theme. On the left is a file tree for the 'sal' project, which includes subfolders like 'examples', 'looker', 'salgen', and 'vendor'. Several files are listed under 'looker': 'looker.go', 'looker_test.go', 'reflect.go' (which is currently selected), and 'reflect_test.go'. Other files in the project include '.gitignore', '.travis.yml', 'Gopkg.lock', 'Gopkg.toml', 'LICENSE', 'README.md', 'sal.go', and 'sal_test.go'. Below the file tree are sections for 'External Libraries' and 'Scratches and Consoles'. The main pane displays the content of 'reflect.go'. The code uses template literals to print package names and reflect.TypeOf to find interface values. It includes imports for 'flag', 'fmt', 'os', and 'reflect', and defines a 'main' function that prints package names and then calls 'looker.LookAtInterfaces' with a list of symbols.

```
173 // This program reflects on an interface value, and prints the
174 // gob encoding of a model.Package to standard output.
175 // JSON doesn't work because of the model.Type interface.
176 var reflectProgram = template.Must(template.New("program").Parse(``)
177 package main
178
179 import (
180     "flag"
181     "fmt"
182     "os"
183     "reflect"
184
185     "github.com/go-gad/sal/looker"
186
187     pkg_ {{printf "%q" .ImportPath}}
188 )
189
190 var output = flag.String("output", "", "The output file name, or empty to use stdout.")
191
192 func main() {
193     flag.Parse()
194
195     pkgPath := {{printf "%q" .ImportPath}}
196     var list = []reflect.Type{
197         {{range .Symbols}}
198             reflect.TypeOf((*pkg_.{{.}})(nil)).Elem(),
199         {{end}}
200     }
201
202     pkg := looker.LookAtInterfaces(pkgPath, list)
203
204     if err := looker.EncodeGob(*output, pkg); err != nil {
205         fmt.Fprintf(os.Stderr, "failed EncodeGob: %s\n", err)
206         os.Exit(1)
207 }
```

Implementation



The screenshot shows a code editor with a dark theme. On the left is a file tree for a project named 'sal' located at `~/go/src/github.com/`. The tree includes folders for 'examples', 'looker' (containing 'testdata' and 'foo-bar'), 'salgen', and 'vendor'. Under 'vendor', there are files like '.gitignore', '.travis.yml', 'Gopkg.lock', 'Gopkg.toml', 'LICENSE', 'README.md', 'sal.go', and 'sal_test.go'. The main pane displays a Go code snippet for implementing an interface:

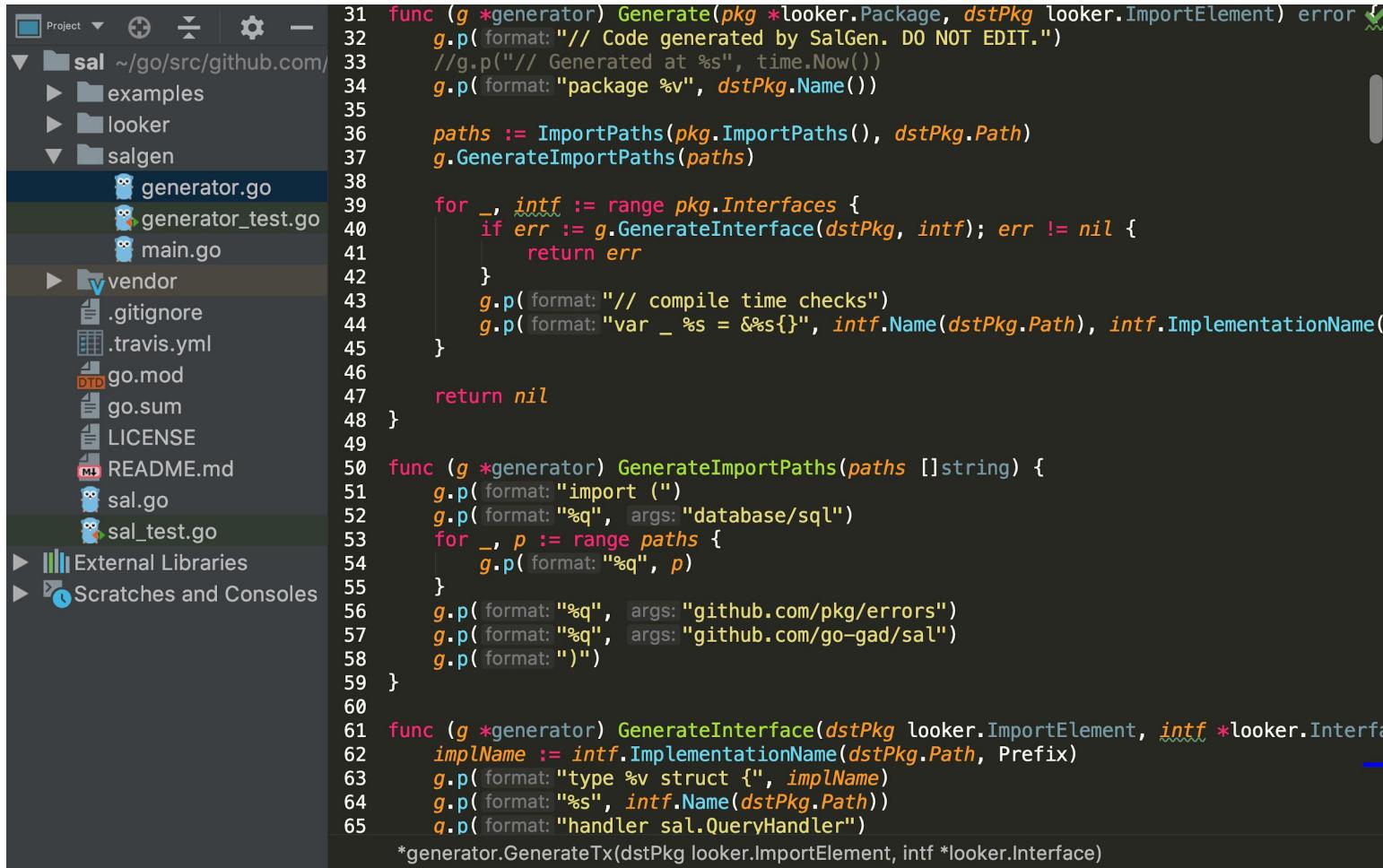
```
1 &looker.Interface{
2     ImportPath: looker.ImportElement{Path:"github.com/go-gad/sal/examples/bookstore", UserType: "Store", Methods: []}
3     &looker.Method{
4         Name: "CreateAuthor",
5         In: {
6             &looker.InterfaceElement{
7                 ImportPath: looker.ImportElement{Path:"context", Alias:""}, UserType: "Context", },
8             &looker.StructElement{
9                 ImportPath: looker.ImportElement{Path:"github.com/go-gad/sal/examples/bookstore", UserType: "CreateAuthorReq", IsPointer: false, Fields: []},
10                {
11                    Name:      "Name",
12                    ImportPath: looker.ImportElement{}, BaseType: "string", UserType: "string", Anonymous: false, Tag: "", Parents: {"BaseAuthor"}, },
13                {
14                    Name:      "Desc",
15                    ImportPath: looker.ImportElement{}, BaseType: "string", UserType: "string", Anonymous: false, Tag: "", Parents: {"BaseAuthor"}, },
16                },
17            },
18        },
19    },
20 }
```

Implementation

The screenshot shows a code editor interface with a dark theme. On the left is a file tree for a project named 'sal' located at '~/go/src/github.com'. The file tree includes subfolders 'examples', 'looker' (which contains 'testdata', 'looker.go', 'looker_test.go', and 'reflect.go'), 'salgen', 'vendor' (containing '.gitignore' and '.travis.yml'), and files like 'go.mod', 'go.sum', 'LICENSE', 'README.md', 'sal.go', and 'sal_test.go'. The file 'reflect.go' is currently selected and highlighted in blue. The main pane displays the source code for the 'EncodeGob' function, starting from line 143. The code uses standard Go syntax with imports for 'os' and 'fmt'. It handles opening and closing an output file, registering Gob elements for struct, slice, and interface types, and finally encoding the package into the outfile.

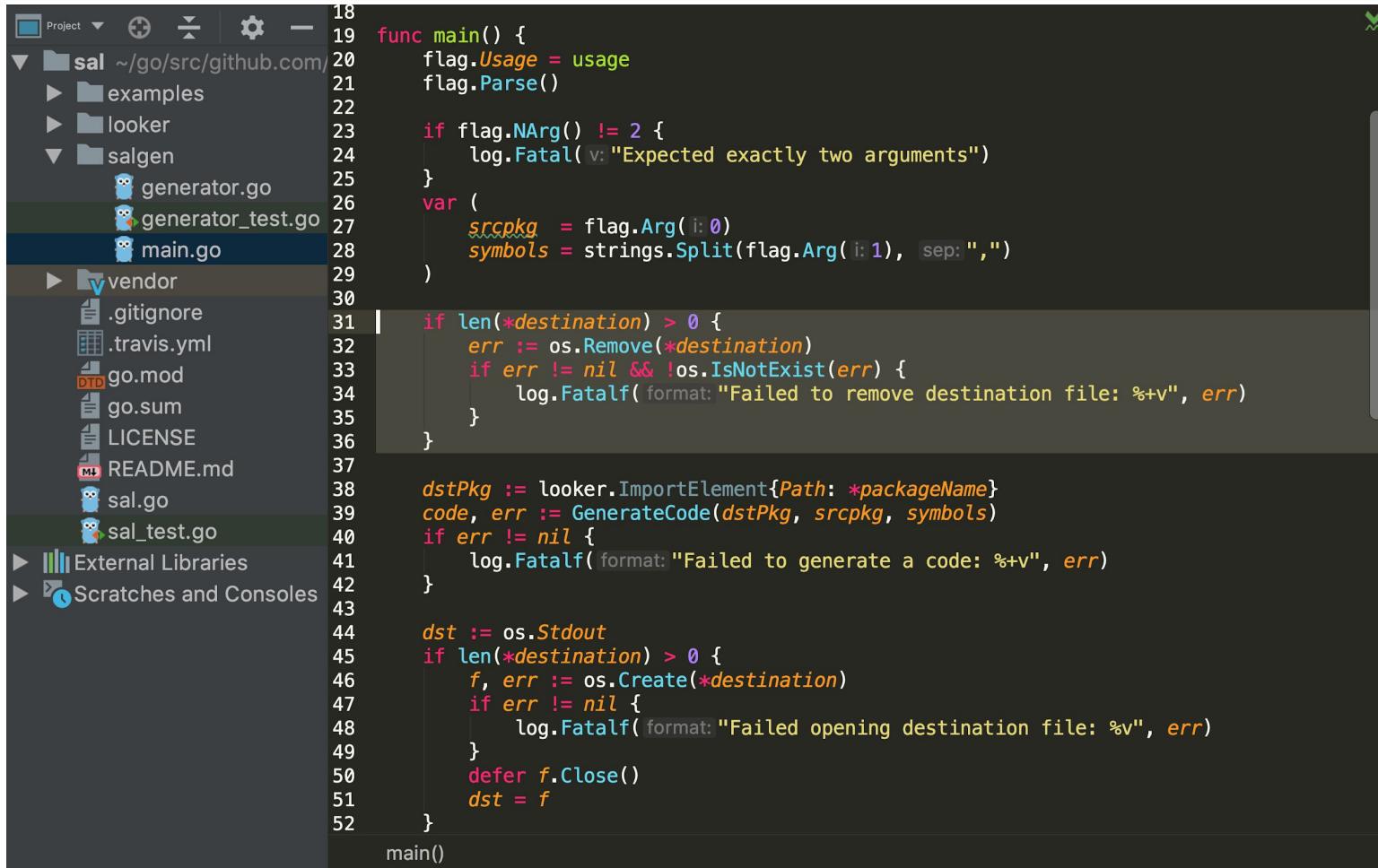
```
143
144
145 func EncodeGob(output string, pkg *Package) error {
146     outfile := os.Stdout
147
148     if len(output) != 0 {
149         var err error
150         if outfile, err = os.Create(output); err != nil {
151             return fmt.Errorf(format:"failed to open output file %q: %s", output, err)
152         }
153         defer func() {
154             if err := outfile.Close(); err != nil {
155                 fmt.Errorf(format:"failed to close output file %q: %s", output, err)
156             }
157         }()
158     }
159
160     gob.Register(&StructElement{})
161     gob.Register(&SliceElement{})
162     gob.Register(&InterfaceElement{})
163     //gob.Register(Parameters{})
164     //gob.Register(Field{})
165     //gob.Register(Fields{})
166
167     if err := gob.NewEncoder(outfile).Encode(pkg); err != nil {
168         fmt.Errorf(format:"gob encode: %s", err)
169     }
170
171     return nil
172 }
173
174
175
176
177 }
```

Implementation



```
31 func (g *generator) Generate(pkg *looker.Package, dstPkg looker.ImportElement) error {
32     g.p("// Code generated by SalGen. DO NOT EDIT.")
33     //g.p("// Generated at %s", time.Now())
34     g.p("package %v", dstPkg.Name())
35
36     paths := ImportPaths(pkg.ImportPaths(), dstPkg.Path)
37     g.GenerateImportPaths(paths)
38
39     for _, intf := range pkg.Interfaces {
40         if err := g.GenerateInterface(dstPkg, intf); err != nil {
41             return err
42         }
43         g.p("// compile time checks")
44         g.p("var _ %s = %s{}", intf.Name(dstPkg.Path), intf.ImplementationName())
45     }
46
47     return nil
48 }
49
50 func (g *generator) GenerateImportPaths(paths []string) {
51     g.p("import ")
52     g.p("%q", args: "database/sql")
53     for _, p := range paths {
54         g.p("%q", p)
55     }
56     g.p("%q", args: "github.com/pkg/errors")
57     g.p("%q", args: "github.com/go-gad/sal")
58     g.p(")")
59 }
60
61 func (g *generator) GenerateInterface(dstPkg looker.ImportElement, intf *looker.Interface) {
62     implName := intf.ImplementationName(dstPkg.Path, Prefix)
63     g.p("type %v struct {", implName)
64     g.p("%s", intf.Name(dstPkg.Path))
65     g.p("handler sal.QueryHandler")
66
67     *generator.GenerateTx(dstPkg looker.ImportElement, intf *looker.Interface)
```

Implementation



The screenshot shows a code editor interface with a dark theme. On the left is a file tree for a project named 'sal' located at `~/go/src/github.com/sal`. The tree includes subfolders like 'examples', 'looker', and 'salgen', and files such as 'generator.go', 'generator_test.go', 'main.go', 'vendor', '.gitignore', '.travis.yml', 'go.mod', 'go.sum', 'LICENSE', 'README.md', 'sal.go', and 'sal_test.go'. Below the file tree are sections for 'External Libraries' and 'Scratches and Consoles'. The main pane displays a block of Go code. The code defines a `main()` function that parses command-line flags, checks for exactly two arguments, and then attempts to remove a destination file if it exists. It then imports an element from the 'looker' package, generates code, and writes it to either `os.Stdout` or a specified destination file.

```
18
19 func main() {
20     flag.Usage = usage
21     flag.Parse()
22
23     if flag.NArg() != 2 {
24         log.Fatalf("Expected exactly two arguments")
25     }
26     var (
27         srcPkg = flag.Arg(0)
28         symbols = strings.Split(flag.Arg(1), sep:",")
29     )
30
31     if len(*destination) > 0 {
32         err := os.Remove(*destination)
33         if err != nil && !os.IsNotExist(err) {
34             log.Fatalf("Failed to remove destination file: %v", err)
35         }
36     }
37
38     dstPkg := looker.ImportElement{Path: *packageName}
39     code, err := GenerateCode(dstPkg, srcPkg, symbols)
40     if err != nil {
41         log.Fatalf("Failed to generate a code: %v", err)
42     }
43
44     dst := os.Stdout
45     if len(*destination) > 0 {
46         f, err := os.Create(*destination)
47         if err != nil {
48             log.Fatalf("Failed opening destination file: %v", err)
49         }
50         defer f.Close()
51         dst = f
52     }
53 }
```

main()

Implementation

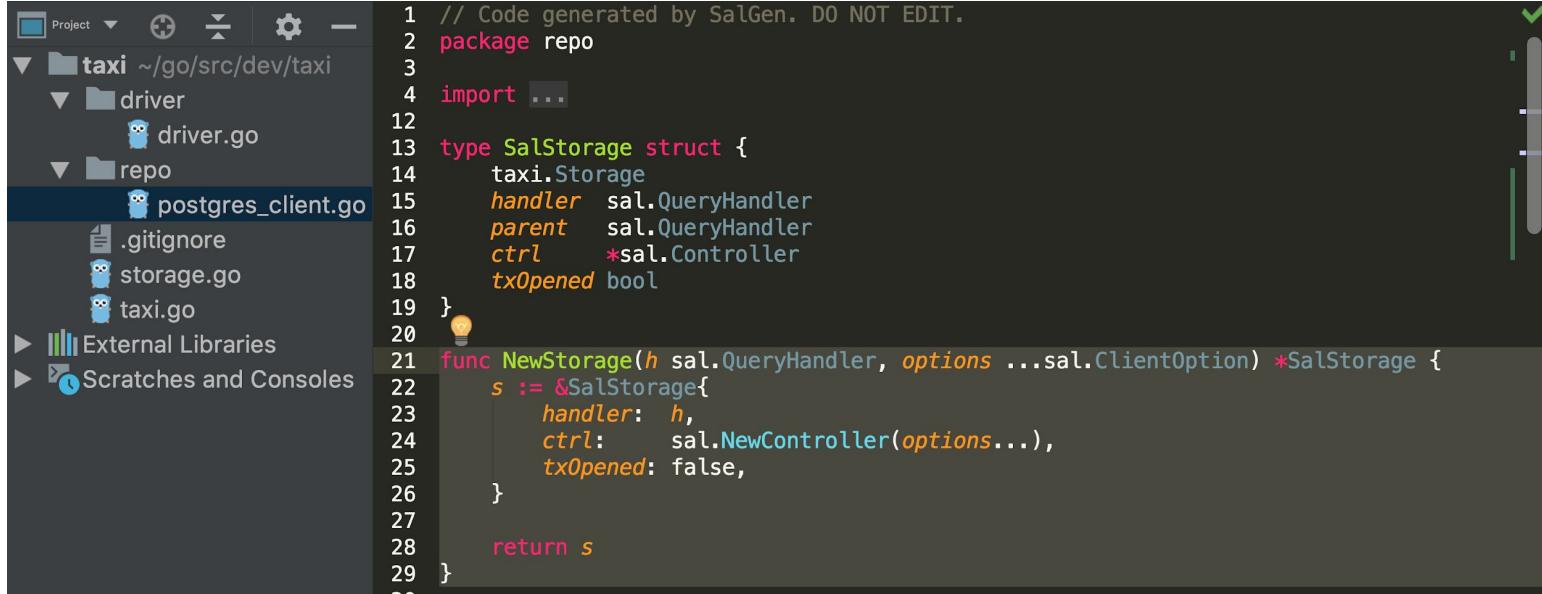
```
344     ctx = context.WithValue(ctx, sal.ContextKeyTxOpened, s.txOpened)
345     ctx = context.WithValue(ctx, sal.ContextKeyOperationType, val: "Exec")
346     ctx = context.WithValue(ctx, sal.ContextKeyMethodName, val: "UpdateAuthor")
347
348     pgQuery, args := sal.ProcessQueryAndArgs(rawQuery, reqMap)
349
350     stmt, err := s.ctrl.PrepareStmt(ctx, s.parent, s.handler, pgQuery)
351     if err != nil {
352         return errors.WithStack(err)
353     }
354
355     for _, fn := range s.ctrl.BeforeQuery {
356         var fnz sal.FinalizerFunc
357         ctx, fnz = fn(ctx, rawQuery, req)
358         if fnz != nil {
359             defer func() { fnz(ctx, err) }()
360         }
361     }
362
363     _, err = stmt.ExecContext(ctx, args...)
364     if err != nil {
365         return errors.Wrap(err, message: "failed to execute Exec")
366     }
367
368     return nil
369 }
370
371 // compile time checks
372 var _ Store = &SalStore{}
```

Usage: interface

```
1 package taxi
2
3 import "context"
4
5 //go:generate salgen -destination=./repo/postgres_client.go -package=dev/taxi/repo dev/taxi Storage
6 type Storage interface {
7     FindDriverByID(ctx context.Context, req *FindDriverByIDReq) (*Driver, error)
8 }
9
10 type FindDriverByIDReq struct {
11     ID string `sql:"id"`
12 }
13
14 func (r *FindDriverByIDReq) Query() string {
15     return `"SELECT name FROM drivers WHERE id=@id"`
16 }
17 |
```



Usage: constructor



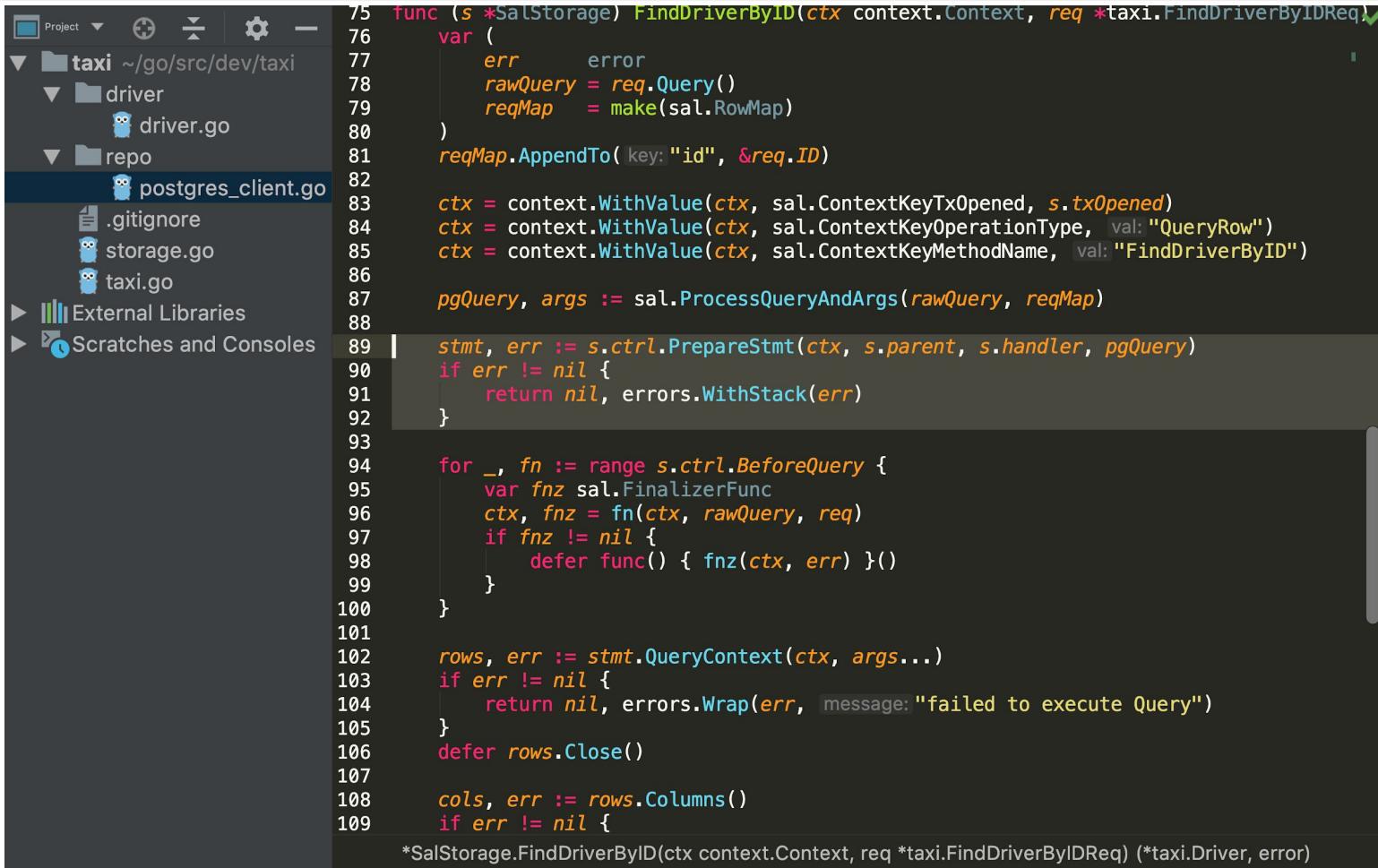
```
1 // Code generated by SalGen. DO NOT EDIT.
2 package repo
3
4 import ...
12
13 type SalStorage struct {
14     taxi.Storage
15     handler sal.QueryHandler
16     parent   sal.QueryHandler
17     ctrl     *sal.Controller
18     txOpened bool
19 }
20
21 func NewStorage(h sal.QueryHandler, options ...sal.ClientOption) *SalStorage {
22     s := &SalStorage{
23         handler: h,
24         ctrl:    sal.NewController(options...),
25         txOpened: false,
26     }
27
28     return s
29 }
```

```
db, err := sql.Open("postgres", "postgres://localhost/taxi?sslmode=disable")
store := repo.NewStorage(db)
```

Usage: methods

```
type Store interface {  
    CreateAuthor(ctx context.Context, req CreateAuthorReq) (CreateAuthorResp, error)  
    GetAuthors(ctx context.Context, req GetAuthorsReq) ([]*GetAuthorsResp, error)  
    UpdateAuthor(ctx context.Context, req *UpdateAuthorReq) error  
}
```

Usage: prepared statements



```
75 func (s *SalStorage) FindDriverByID(ctx context.Context, req *taxi.FindDriverByIDReq) (*taxi.Driver, error) {
76     var (
77         err      error
78         rawQuery = req.Query()
79         reqMap   = make(sal.RowMap)
80     )
81     reqMap.AppendTo(key: "id", &req.ID)
82
83     ctx = context.WithValue(ctx, sal.ContextKeyTxOpened, s.txOpened)
84     ctx = context.WithValue(ctx, sal.ContextKeyOperationType, val: "QueryRow")
85     ctx = context.WithValue(ctx, sal.ContextKeyMethodName, val: "FindDriverByID")
86
87     pgQuery, args := sal.ProcessQueryAndArgs(rawQuery, reqMap)
88
89     stmt, err := s.ctrl.PrepareStmt(ctx, s.parent, s.handler, pgQuery)
90     if err != nil {
91         return nil, errors.WithStack(err)
92     }
93
94     for _, fn := range s.ctrl.BeforeQuery {
95         var fnz sal.FinalizerFunc
96         ctx, fnz = fn(ctx, rawQuery, req)
97         if fnz != nil {
98             defer func() { fnz(ctx, err) }()
99         }
100    }
101
102    rows, err := stmt.QueryContext(ctx, args...)
103    if err != nil {
104        return nil, errors.Wrap(err, message: "failed to execute Query")
105    }
106    defer rows.Close()
107
108    cols, err := rows.Columns()
109    if err != nil {
```

*SalStorage.FindDriverByID(ctx context.Context, req *taxi.FindDriverByIDReq) (*taxi.Driver, error)

Usage: named arguments

The screenshot shows a dark-themed Go code editor interface. On the left, there's a sidebar with a 'Project' tab and several sections: 'taxi' (containing 'driver' and 'repo' folders with files like 'driver.go', 'postgres_client.go', and 'storage_test.go'), 'External Libraries' (listing 'Go SDK 1.9.4' and 'GOPATH <taxi>'), and 'Scratches and Consoles'. The main pane displays the following Go code:

```
1 package taxi
2
3 import ...
4
5 //go:generate salgen -destination=./repo/postgres_client.go -package=dev/taxi/repo dev/
6
7 type Storage interface {
8     FindDriverByID(ctx context.Context, req *FindDriverByIDReq) (*Driver, error)
9 }
10
11
12 type FindDriverByIDReq struct {
13     ID string `sql:"id"`
14 }
15
16 func (r *FindDriverByIDReq) Query() string {
17     return `SELECT id, name FROM drivers WHERE id=@id`
18 }
19
```

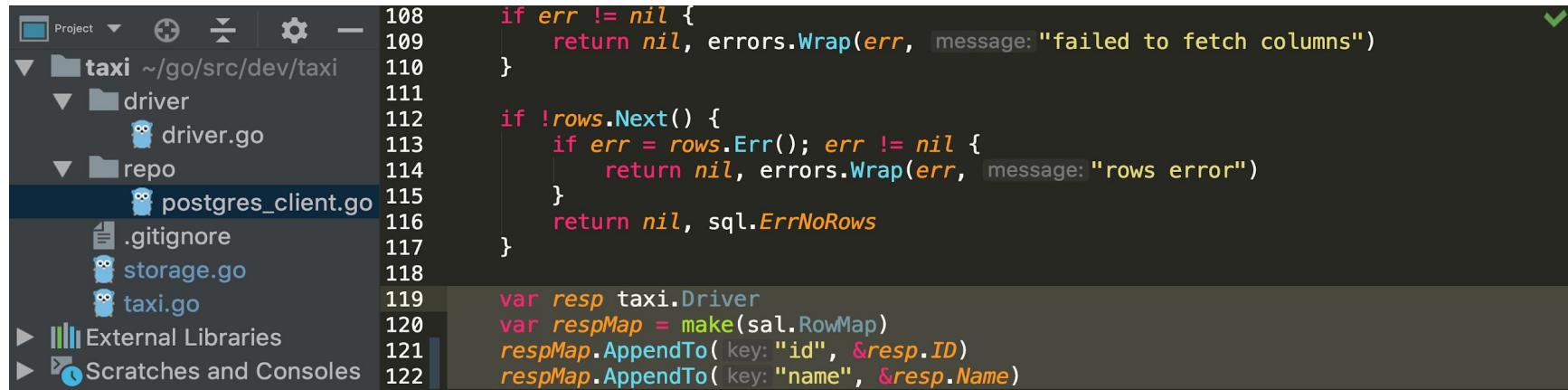
The code defines a package 'taxi' with an interface 'Storage' containing a method 'FindDriverByID'. It also defines a struct 'FindDriverByIDReq' with a field 'ID' annotated with 'sql:"id"'. A function 'Query' returns a SQL query string using the placeholder '@id'.

Usage: mapping



A screenshot of a Go code editor showing a project structure and some code. The project is named 'taxi' and contains 'driver' and 'repo' packages. The 'repo' package has files like '.gitignore', 'storage.go', and 'taxi.go'. The main code block shows the definition of a 'DriverService' interface and a 'Driver' struct.

```
1 package taxi
2
3 import "context"
4
5 type DriverService interface {
6     FindDriver(ctx context.Context, id string) (*Driver, error)
7 }
8
9 type Driver struct {
10    ID   string `sql:"id"`
11    Name string `sql:"name"`
12 }
```



A screenshot of a Go code editor showing a continuation of the 'postgres_client.go' file. It includes error handling for database rows and demonstrates how to map a database row into a 'taxi.Driver' struct using a 'RowMap'.

```
108     if err != nil {
109         return nil, errors.Wrap(err, message:"failed to fetch columns")
110     }
111
112     if !rows.Next() {
113         if err = rows.Err(); err != nil {
114             return nil, errors.Wrap(err, message:"rows error")
115         }
116         return nil, sql.ErrNoRows
117     }
118
119     var resp taxi.Driver
120     var respMap = make(sal.RowMap)
121     respMap.AppendTo(key: "id", &resp.ID)
122     respMap.AppendTo(key: "name", &resp.Name)
```

Usage: complex data types

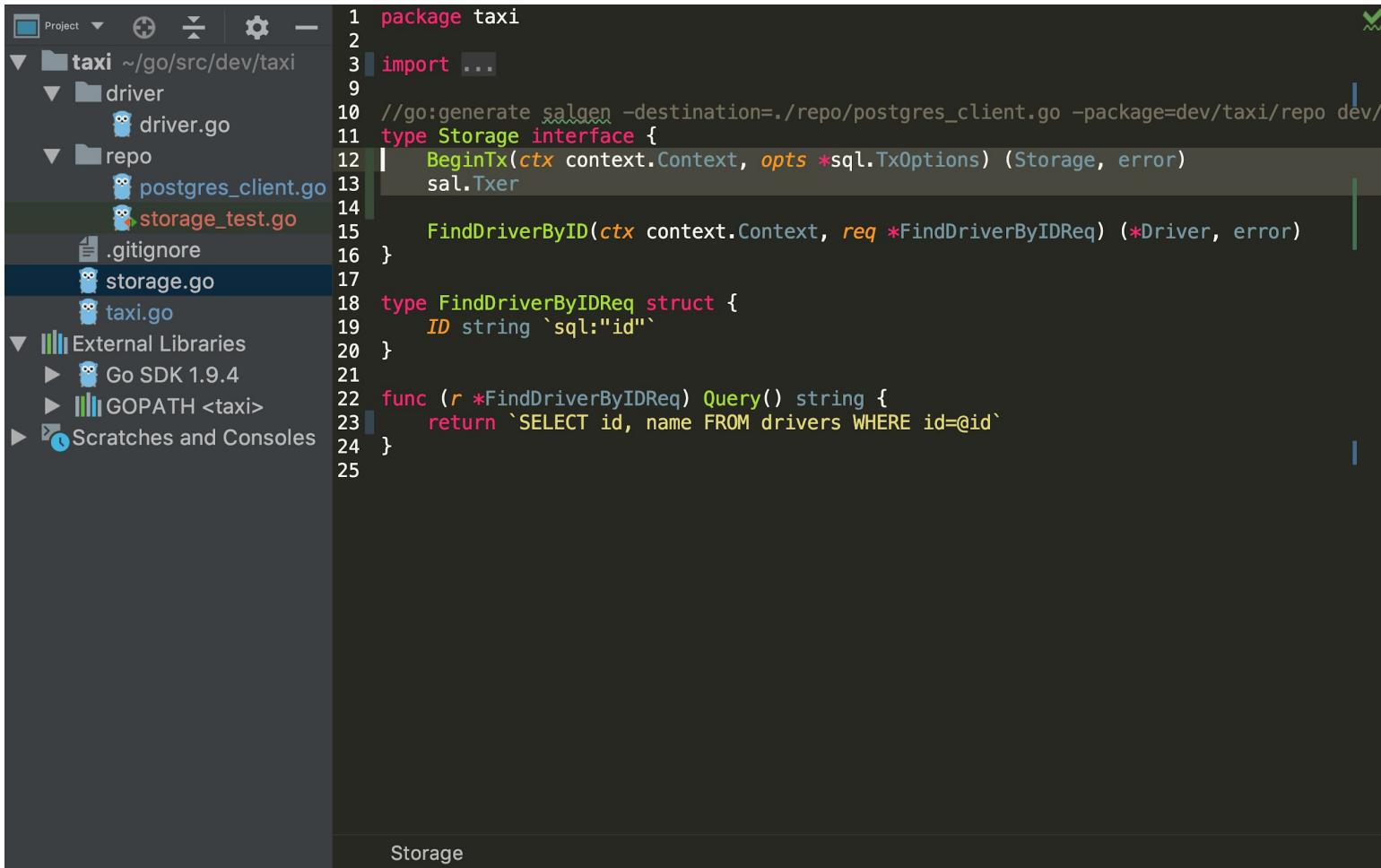
```
type ProcessRower interface {
    ProcessRow(rowMap RowMap)
}

type DeleteAuthrosReq struct {
    Tags []int64 `sql:"tags"`
}

func (r *DeleteAuthorsReq) ProcessRow(rowMap sal.RowMap) {
    rowMap.Set("tags", pq.Array(r.Tags))
}

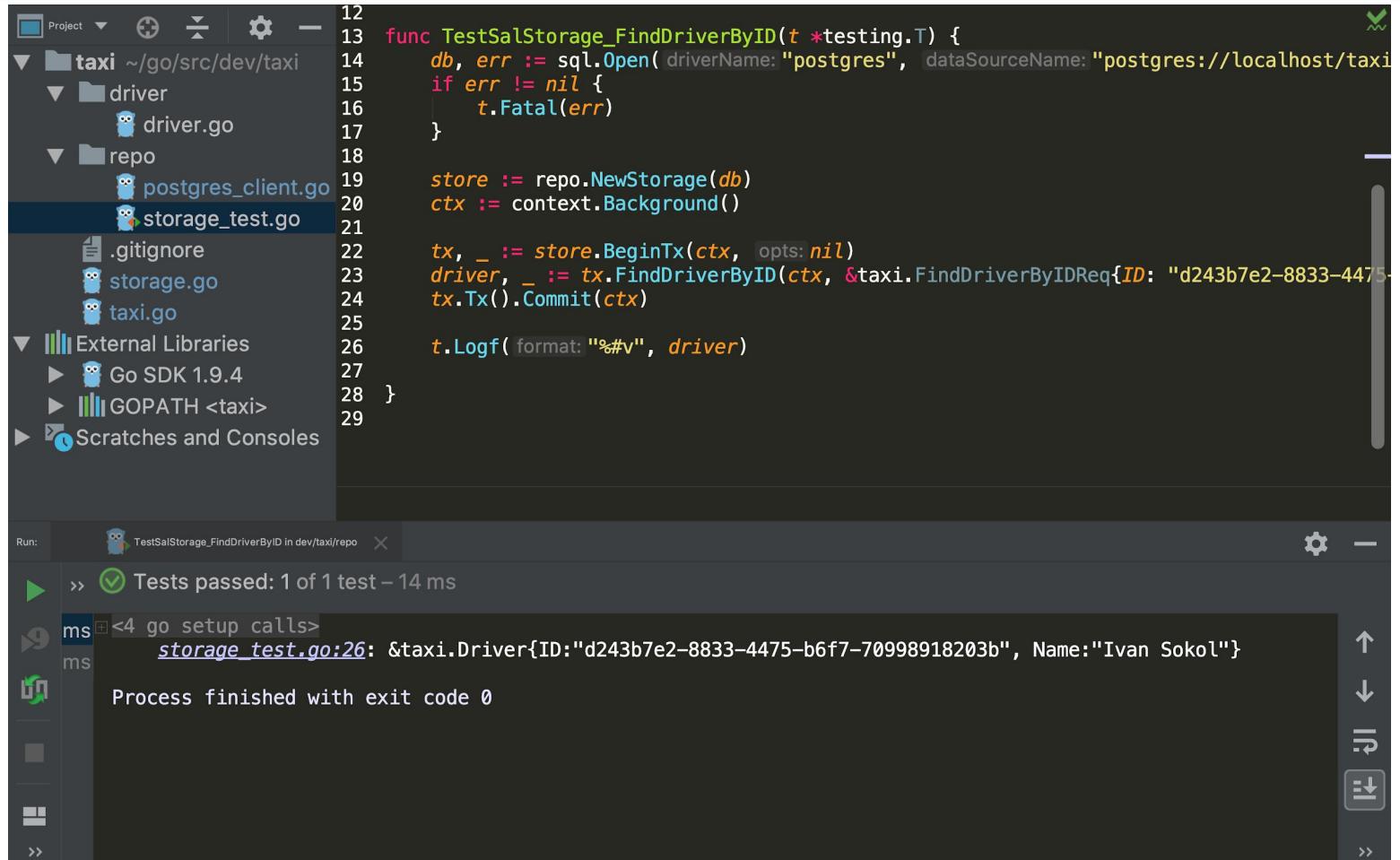
func (r *DeleteAuthorsReq) Query() string {
    return `DELETE FROM authors WHERE tags=ANY(@tags::UUID[])`
}
```

Usage: transactions



```
1 package taxi
2
3 import ...
4
5 //go:generate salgen -destination=./repo/postgres_client.go -package=dev/taxi/repo dev/
6 type Storage interface {
7     BeginTx(ctx context.Context, opts *sql.TxOptions) (Storage, error)
8     sal.Txer
9
10    FindDriverByID(ctx context.Context, req *FindDriverByIDReq) (*Driver, error)
11 }
12
13 type FindDriverByIDReq struct {
14     ID string `sql:"id"`
15 }
16
17 func (r *FindDriverByIDReq) Query() string {
18     return `SELECT id, name FROM drivers WHERE id=@id`
19 }
20
21 }
```

Usage: transactions



The screenshot shows a code editor interface with a dark theme. On the left is a sidebar showing a project structure for a 'taxi' application. The 'repo' directory contains 'postgres_client.go' and 'storage_test.go'. The 'storage_test.go' file is open in the main editor area, showing the following Go code:

```
12
13 func TestSalStorage_FindDriverByID(t *testing.T) {
14     db, err := sql.Open(driverName: "postgres", dataSourceName: "postgres://localhost/taxi")
15     if err != nil {
16         t.Fatal(err)
17     }
18
19     store := repo.NewStorage(db)
20     ctx := context.Background()
21
22     tx, _ := store.BeginTx(ctx, opts: nil)
23     driver, _ := tx.FindDriverByID(ctx, &taxi.FindDriverByIDReq{ID: "d243b7e2-8833-4475-b6f7-70998918203b"})
24     tx.Tx().Commit(ctx)
25
26     t.Logf(format: "%#v", driver)
27
28 }
```

At the bottom of the screen, the terminal output shows the test results:

```
Run:  TestSalStorage_FindDriverByID in dev/taxi/repo
ms >>  Tests passed: 1 of 1 test - 14 ms
ms <4 go setup calls>
ms   storage_test.go:26: &taxi.Driver{ID:"d243b7e2-8833-4475-b6f7-70998918203b", Name:"Ivan Sokol"}
ms
Process finished with exit code 0
```

The terminal also includes navigation icons for up, down, left, right, and search.

Usage: hooks

```
// BeforeQueryFunc is called before the query execution but after the preparing stmts.  
// Returns the FinalizerFunc.  
type BeforeQueryFunc func(ctx context.Context, query string, req interface{}) (context.Context,  
FinalizerFunc)  
  
// FinalizerFunc is executed after the query execution.  
type FinalizerFunc func(ctx context.Context, err error)
```

Usage

```
func TestSalStore_CreateAuthor(t *testing.T) {
    db, mock, err := sqlmock.New()
    if err != nil {
        t.Fatalf("an error '%s' was not expected when opening a stub database connection", err)
    }
    defer db.Close()
    b1 := func(ctx context.Context, query string, req interface{}) (context.Context, sal.Finalizer) {
        start := time.Now()
        return ctx, func(ctx context.Context, err error) {
            t.Logf(
                "format: \"%q > Opeartion %q: %q with req %#v took [%v] inTx[%v] Error: %+v",
                ctx.Value(sal.ContextKeyMethodName),
                ctx.Value(sal.ContextKeyOperationType),
                query,
                req,
                time.Since(start),
                ctx.Value(sal.ContextKeyTxOpened),
                err,
            )
        }
    }
    client := NewStore(db, sal.BeforeQuery(b1))
}
```

Run: [TestSalStore_CreateAuthor in github.com/go-gad/sal/examples/bookstore](#)

Tests passed: 1 of 1 test – 0 ms

ms [io setup calls](#)

ms [sal_client_test.go:24: "CreateAuthor" > Opeartion "Prepare": "INSERT INTO authors \(Name, Desc, CreatedAt\)"](#)

ms [sal_client_test.go:24: "CreateAuthor" > Opeartion "QueryRow": "INSERT INTO authors \(Name, Desc, CreatedAt\)"](#)

Process finished with exit code 0

Features list

- Generated client based on the interface
- Prepared statements
- Named arguments
- Mapping
- Transactions
- Middleware

<https://godoc.org/github.com/go-gad/sal>

**Thank you
Zaur Abasmirzoev
Gett
<http://zaur.io>**