

Web Technologies - Frameworks, Libraries and Plattformen

Chair of software engineering for business applications (sebis)

Technische Universität München

Let's Go Revel

Written Tutorial of the Live Coding Session

Author	Matthias Holdorf
Advisor	Alexander Waldmann
Term	Summer term 2015
Submission date	10.04.2015

Content

1	Purpose of this Tutorial.....	3
2	Installation of the Go language	4
3	Structure of a Go workspace	5
4	Installation of the Revel Framework	6
5	Structure of Revel workspace	7
6	Live Coding Application.....	8
6.1	Create the Application and get an Overview.....	8
6.2	Change the Version of bootstrap.css.....	8
6.3	First changes to the Index.html.....	8
6.4	Running the Application	9
6.5	Adding our Application Domain Model	10
6.6	Create the User Model.....	10
6.7	Adding Validation to our User Model	11
6.8	Additional changes to the Index.html file	11
6.9	Adding an Action Method to the Controller.....	12
6.10	Adding the ToDo-Controller	13
6.11	Adding Validation functionality to our Index.html.....	14
6.12	Adding the Index.html for the ToDo-Controller	15
6.13	Create the ToDo Model	16
6.14	Adding the AddToDo-Action method	17
6.15	Display the ToDo-List.....	18
6.16	Add the CompleteToDo- and Logout-Action method.....	19
6.17	Adding routes and safety	19
6.18	Adding Interceptors to our Application	20
6.19	Adding a Test to the application.....	21
6.20	The ToDo Application.....	22
6.21	What's next?	22

7	Learning Go and Revel	23
7.1	Books (https://github.com/golang/go/wiki/Books)	23
7.2	Online	23
7.3	Revel	23
8	Comparison of IDEs	24
7	Comparison of IDEs	25
7.1	Personal preference	25
7.2	Further Reading	25
8	Debugging Go Code with GDB	26
9	A HTTP Server in native Go	27

1 Purpose of this Tutorial

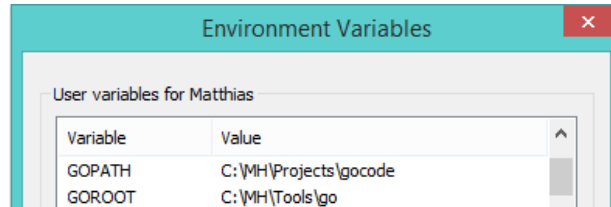
This tutorial is targeting programmers with a basic knowledge in any C-Like language, the MVC-Concept and web development.

After explaining how to install Go and Revel you will create an application that allows a user to add a todo to an existing list as well as completing todos from that list.

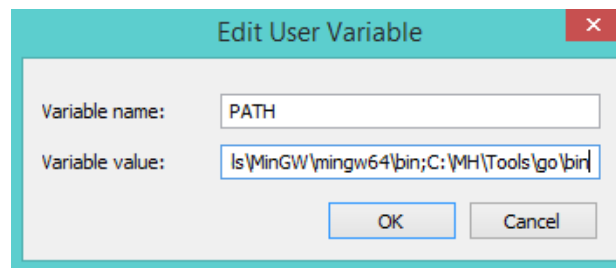
The purpose of this application is to demonstrate the main features of the [Revel Web Framework](#) in an understandable manner to the target audience.

2 Installation of the Go language

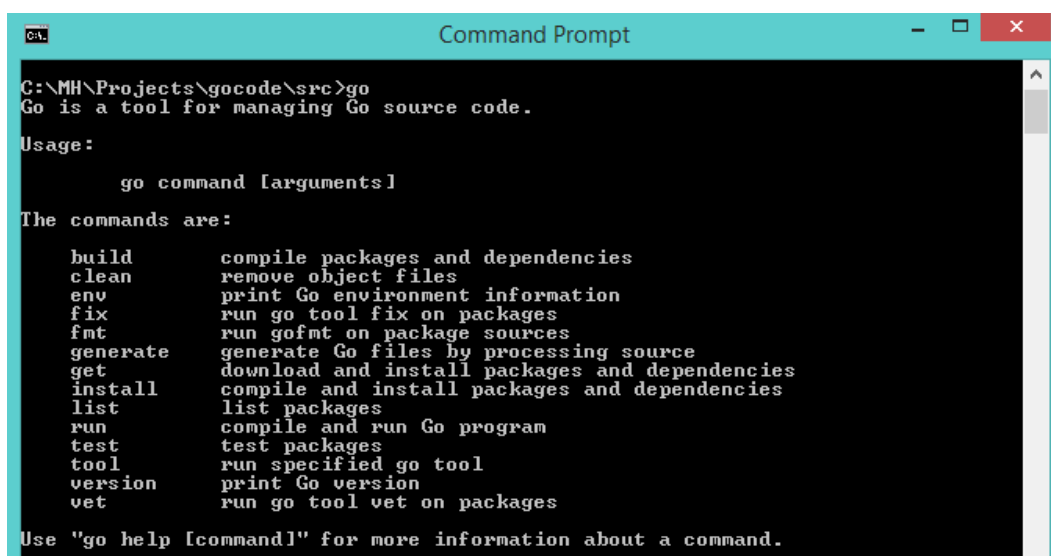
1. Download **Go** from <http://golang.org/dl/>
2. Set **GOROOT** to the destination folder



3. Set **GOPATH** (the workspace for your Go-Projects)
4. Add „...\\go\\bin“ to your **PATH**



5. Typing „go“ in the command line should result in an overview of [eligible commands](#)



```
C:\MH\Projects\gocode\src>go
Go is a tool for managing Go source code.

Usage:

    go command [arguments]

The commands are:

    build      compile packages and dependencies
    clean      remove object files
    env        print Go environment information
    fix        run go tool fix on packages
    fmt        run gofmt on package sources
    generate    generate Go files by processing source
    get        download and install packages and dependencies
    install    compile and install packages and dependencies
    list       list packages
    run        compile and run Go program
    test       test packages
    tool       run specified go tool
    version    print Go version
    vet        run go tool vet on packages

Use "go help [command]" for more information about a command.
```

3 Structure of a Go workspace

```
gocode/                # GOPATH
  bin/
    hello.exe          # executable command
  pkg/
    windows_amd64/
      github.com/mh/
        print.a        # package object
  src/
    github.com/mh/
      .git              # git repository metadata
      hello/
        hello.go        # command source
      print/
        print.go        # package source
```

4 Installation of the Revel Framework

1. Download **Revel** via the „go get“ [command](#)

```
go get github.com/revel/revel
```

2. Download the **Revel command line** tool via the „go get“ [command](#)

```
go get github.com/revel/cmd/revel
```

3. Download **Revel's samples** via the „go get“ [command](#)

```
go get github.com/revel/samples
```

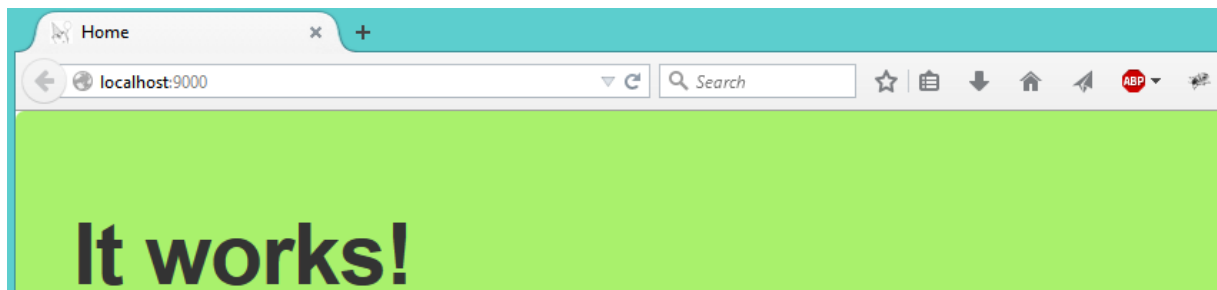
4. Create a **new application** via the „revel new“ [command](#)

```
revel new application_name
```

5. **Lunch** a new application or sample project via the „revel run“ [command](#)

```
revel run application_name
```

```
revel run github.com/samples/chat
```



5 Structure of Revel's Skeleton Application

```
gocode          # GOPATH
src
  revel          # Revel source code
  ...
  appName        # App root
  app            # App sources
    controllers  # App controllers
    init.go      # Interceptor registration
    models       # App domain models (e.g. User)
    routes       # Reverse routes (generated code)
    views        # Views/Templates
  conf           # Configuration files
    app.conf     # Main configuration file
    routes       # Routes definition
  messages       # Message files (for internationalization)
  public         # Public assets
    css          # CSS files
    js           # Javascript files
    images       # Image files
  tests          # Test suites
```


6 Live Coding Application

6.1 Create the ToDo Application and get an Overview

After installing the Revel Framework according to chapter 4, create a **new application** via the „revel new“ [command](#):

```
revel new todo
```

For an overview of the just created new application have a look at chapter 5, where the structure of the skeleton is briefly explained.

6.2 Change the Version of bootstrap.css

The Revel skeleton ships with an older version of the bootstrap CSS framework (v2.1.1). Please download a new version under <http://getbootstrap.com/>. The version used in this tutorial is v3.3.4. Then replace the file under the following path:

```
GOPATH/src/todo/public/css/bootstrap.css
```

6.3 First changes to the Index.html

At first we want to change the Index.html file of our new application located under:

```
GOPATH/src/todo/app/views/App/Index.html
```

Delete the existing source code and replace it with the following:

```
{{set . "title" "Login"}}
{{template "header.html" .}}

<div class="container">
  <div class="col-md-4">
    <h1>ToDo Example</h1>

    <form action="" method="POST">

      <div class="form-group">
        <label for="">User Name:</label>
        <input class="form-control" autofocus="" type="text"
          name="" value="" />
      </div>

      <button class="btn btn-default" type="submit">Login</button>

    </form>

  </div>
</div>

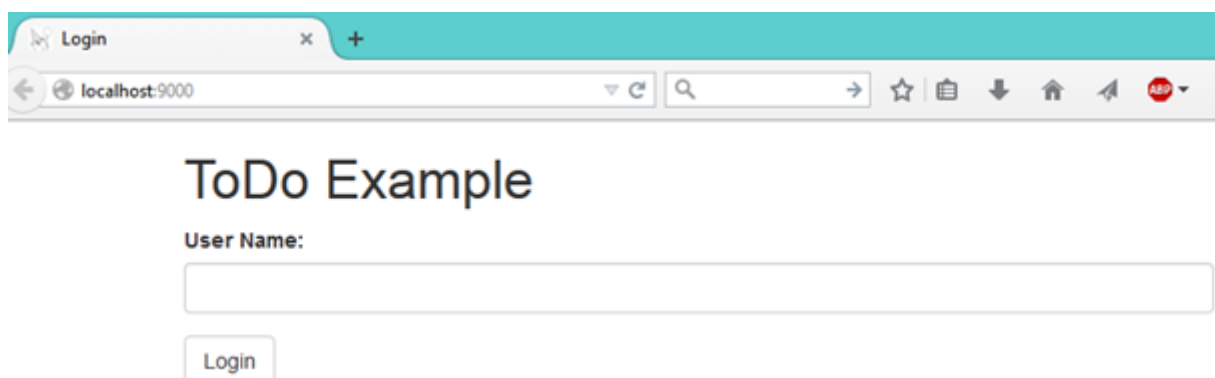
{{template "footer.html" .}}
```

6.4 Running the Application

At this point we can run our application via the „revel run“ [command](#):

```
revel run todo
```

The application should now look like the following:

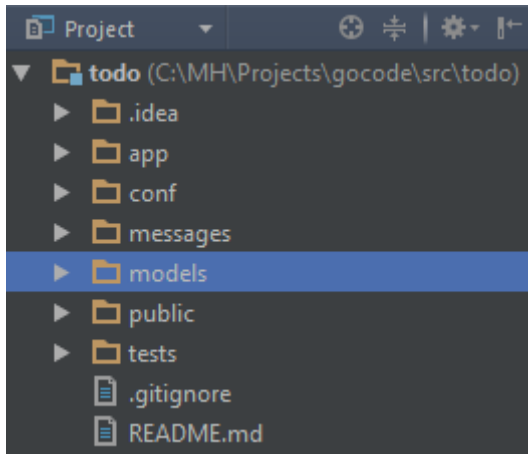


Note that a Revel application runs per default on port 9000. This can be changed in the GOPATH/src/todo/conf/app.conf file.

6.5 Adding our Application Domain Model

Revel uses the MVC Pattern. The „M” in MVC stands for model. This is achieved by using structs. For our application we will need a User model.

Create a folder under GOPATH/src/todo/ called models:



Within that folder create a new file called User.go.

6.6 Create the User Model

Edit the source code of User.go to correspond to the listing below:

```
package models

import (
    "github.com/revel/revel"
    "fmt"
)

type User struct {
    Name string
}

func (u *User) String() string {
    return fmt.Sprintf("User(%s)", u.Name)
}
```

The function String() with the receiver (u User), therefore better called a [method](#), implements the Stringer interface. This allows a custom formatting for the [fmt.Println\(\)](#) function.

6.7 Adding Validation to our User Model

Revel's [validation feature](#) allows us to implement the validation logic directly in the model itself. Later in this tutorial we will see how we can make use of this in the template.

Add the following source code to your `User.go` file:

```
func (user *User) Validate(v *revel.Validation) {
    v.Required(user.Name).Message("A User Name is required!")
    v.MinSize(user.Name, 4).Message("The User Name must be at
                                   least 4 characters long!")
    v.MaxSize(user.Name, 10).Message("The User Name may not be
                                   longer than 10 characters!")
}
```

6.8 Additional changes to the Index.html file

Add this point we can add additional features to our `Index.html` file located under:

GOPATH/src/todo/app/views/App/Index.html

We now add the following [template features](#) provided by our newly created user model to our `Index.html`:

```
{{set . "title" "Login"}}
{{template "header.html" .}}

<div class="container">
    <div class="col-md-4">
        <h1>ToDo Example</h1>

        <form action="{{url "App.Login"}}" method="POST">
            {{with $field := field "user.Name" .}}
                <div class="form-group">
                    <label for="{{ $field.Id }}">User Name:</label>
                    <input class="form-control" autofocus="" type="text"
                        name="{{ $field.Name }}" value="{{ $field.Flash }}" />
                </div>
            {{end}}

            <button class="btn btn-default" type="submit">Login</button>

        </form>

    </div>
</div>

{{template "footer.html" .}}
```

Note how changes directly show after refreshing your browser. There is no need to rebuild the application. This is due to Revel's hot code reload feature, which makes developing an application very fluent.

Next refresh your browser; you will very likely see the following error message:

Template Execution Error
4: executing "App/Index.html" at <url "App.Login">: error calling url: reversing App.Login: revel/controller: failed to find action Login

In App/Index.html (around line 8)

3:	
4:	<div class="container">
5:	<div class="col-md-4">
6:	<h1>ToDo Example</h1>
7:	
8:	<form action="{{url "App.Login"}}" method="POST">

With the last code snippet we also added an action method which doesn't exist yet.

Note how the Revel Framework informs you with a build-in error template.

6.9 Adding an Action Method to the Controller

Controllers are located under the path GOPATH/src/todo/app. Now we add the following action method to your app.go controller to handle the login:

```
func (c App) Login(user *models.User) revel.Result {
    user.Validate(c.Validation)

    // Handle errors
    if c.Validation.HasErrors() {
        c.Validation.Keep()
        c.FlashParams()
        return c.Redirect(App.Index)
    }

    // Storing the user's name in the session
    c.Session["userName"] = user.Name
    c.Session.SetNoExpiration()

    // Inform the user about the success on the next page
    c.Flash.Success("Welcome, " + user.Name + "!")

    return c.Redirect(Todo.Index)
}
```

This action method makes use of the previous defined validation rules from our user model: (see chapter 6.7)

```
user.Validate(c.Validation)
```

It also saves the `user.Name` in a session with no expiration time:

```
// Storing the user's name in the session
c.Session["userName"] = user.Name
c.Session.SetNoExpiration()
```

Important: Since we make use of our User model, we have to tell the controller to import it. Your imports in the `GOPATH/src/todo/app/controllers/app.go` file should look like this:

```
import (
    "github.com/revel/revel"
    "todo/models"
)
```

6.10 Adding the ToDo-Controller

If the login is successful, the `Login` method from the previous chapter will redirect to the `Todo.Index` method. This results in an error, since the method doesn't exist yet.

Therefore we add a new controller called `todo.go` with the following source code:

```
package controllers

import (
    "github.com/revel/revel"
    "todo/models"
    "todo/app/routes"
    "strconv"
)

type Todo struct {
    *revel.Controller
}

func (c Todo) Index() revel.Result {
    return c.Render()
}
```

Note that we will make use of some of the imports later. However, not using them now will result in an error.

6.11 Adding Validation functionality to our Index.html

If the login isn't successful, the Login method of chapter 6.9 will redirect to the App.Index page with an error message defined in the user model's validate method:

```
// Handle errors
if c.Validation.HasErrors() {
    c.Validation.Keep()
    c.FlashParams()
    return c.Redirect(App.Index)
}
```

To display these error messages, one last step has to be taken to complete our Index.html file located under:

```
GOPATH/src/todo/app/views/App/Index.html
```

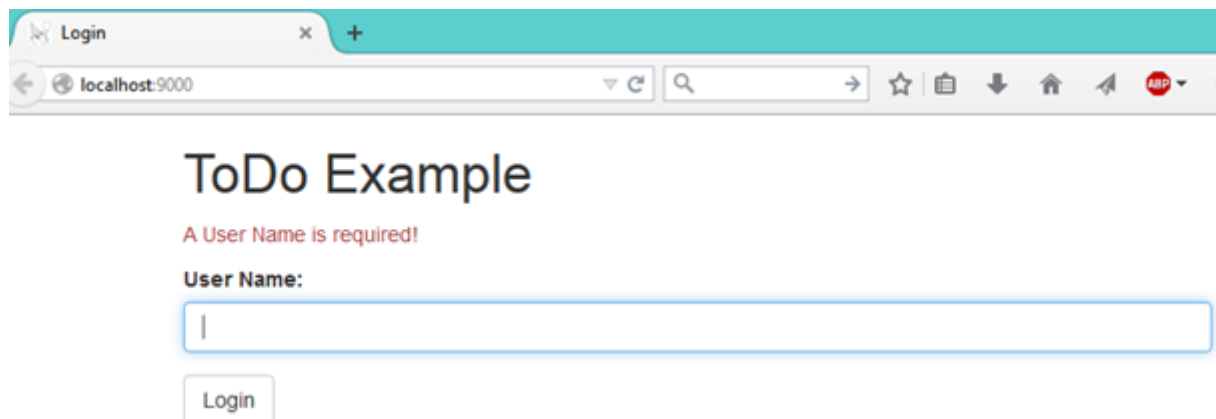
Below our headline:

```
<h1>ToDo Example</h1>
```

Add the following source code:

```
{{range .errors}}
    <p class="text-danger">{{.Message}}</p>
{{end}}
<p class="text-danger">{{.flash.error}}</p>
```

This will allow for error message from our user model, as well as for flash error message, which we will make us of later in this tutorial:



6.12 Adding the Index.html for the ToDo-Controller

Our newly created ToDo-Controller has an action method called `Index()` which will render a result.

After convention Revel will look inside `GOPATH/src/todo/app/views/todo/...` for the result. Since this file doesn't exist yet, we need to add another `Index.html` to our project:

```
{{set . "title" "ToDo"}}
{{template "header.html" .}}

<div class="container">
  <div class="col-md-4">
    <h1>ToDo Example</h1>

    {{range .errors}}
    <p class="text-danger">{{.Message}}</p>
    {{end}}
    <p class="text-success">{{.flash.success}}</p>

    <form action="{{url "ToDo.AddToDo"}}" method="POST">

      {{with $field := field "todo.Name" .}}
      <div class="form-group">
        <label for="{{$.field.Id}}">New ToDo:</label>
        <input id="new-todo" class="form-control" autofocus=""
          type="text" name="{{$.field.Name}}" value="{{$.field.Flash}}" />
      </div>
      {{end}}

      <button id="send-todo" class="btn btn-default" type="submit">
        Add ToDo
      </button>

    </form>

  </div>
</div>

{{template "footer.html" .}}
```

Note that this file looks very similar to the `Index.html` file of our application. Therefore we need to take the same steps as we did earlier. Maybe, you already guess what steps need to be taken now?

6.13 Create the ToDo Model

As we have done it before with the User model, we need to create a ToDo model accordingly.

```
package models

import (
    "fmt"
    "github.com/revel/revel"
)

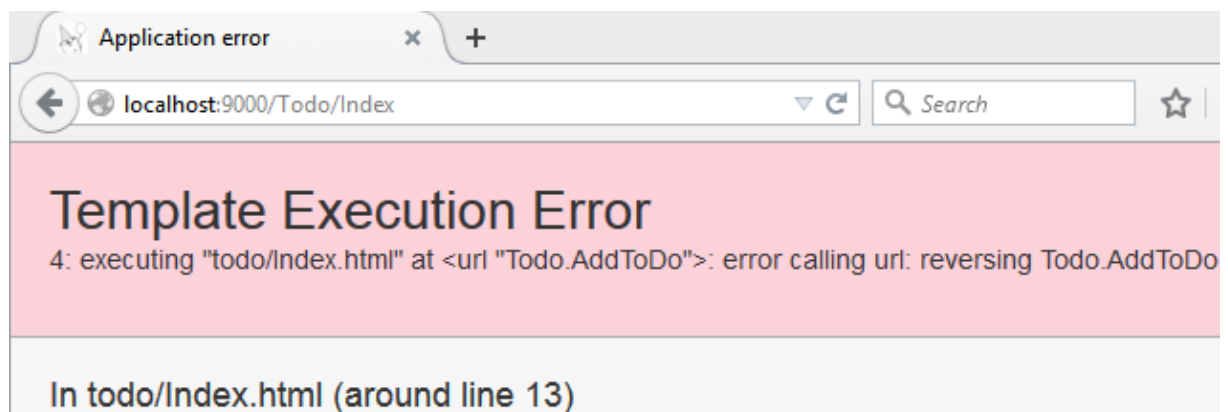
type Todo struct {
    Name        string
    Completed    bool
    Number       int
    Creator      string
}

func (t Todo) String() string {
    return fmt.Sprintf("ToDo: %s is done: %t", t.Name, t.Completed)
}

func (todo *Todo) Validate(v *revel.Validation) {
    v.Required(todo.Name).Message("A ToDo has to have a name!")
}
```

Note that in Go functions (or [methods](#) in this case) need to start with a capital letter to be accessible (public) for the importing Go-File. If a function or methods starts with a lower case letter it will be private and hence only accessible within the file itself.

Refreshing the application in your browser at this point will lead to the following error, which indicates that we miss an action method in the ToDo-Controller:



Let us add this method next!

6.14 Adding the AddTodo-Action method

Since the purpose of this application is to add a todo to an existing list of todos, we need to add the action method which handles this functionality. Therefore add the following source code to your ToDo-Controller:

```
var todoList = []models.TODO{}

func (c TODO) AddTodo(todo models.TODO) revel.Result {
    todo.Validate(c.Validation)

    // Handle errors
    if c.Validation.HasErrors() {
        c.Validation.Keep()
        c.FlashParams()
        return c.Redirect(TODO.Index)
    }

    // Setting the Number and Creator field of the task accordingly
    todo.Number = len(todoList)
    todo.Number++
    todo.Creator = c.Session["userName"]
    todoList = append(todoList, todo)

    return c.Redirect(TODO.Index)
}
```

We can now add a todo to the declared variable todoList. However, we can't display the list yet. This will be the scope of the next chapter.

6.15 Display the ToDo-List

In order to display our todoList we add the following source code to our Index.html file of the ToDo-Controller, located under the following path:

GOPATH/src/todo/app/views/todo/Index.html

```
<form action="{{url \"Todo.CompleteToDo\"}}" method="POST">
  <table class="table">
    <thead>
      <tr>
        <th><span class="caret"></span></th>
        <th>#</th>
        <th>Name</th>
        <th>Completed</th>
        <th>Creator</th>
      </tr>
    </thead>
    <tbody>
      {{range .todoList}}
      <tr>
        <td><input type="checkbox" name="{{.Number}}"></td>
        <th scope="row">{{.Number}}</th>
        <td>{{.Name}}</td>
        <td>{{.Completed}}</td>
        <td>{{.Creator}}</td>
      </tr>
      {{end}}
    </tbody>
  </table>

  <button class="btn btn-success" type="submit">Complete ToDo</button>

  <a href="{{url \"Todo.Logout\"}}">
    <button type="button" class="btn btn-danger pull-right">Logout</button>
  </a>
</form>
```

Note how the range function allows the template to iterate over the todoList passed by the ToDo-Controller. For more template functions have a look at the [documentation](#).

We also need to add the todoList variable to our action method Index():

```
func (c Todo) Index() revel.Result {
    return c.Render(todoList)
}
```

6.16 Add the CompleteTodo- and Logout-Action method

Refreshing the application at this point will result in an error. As before, the Revel will give you information about the error, indicating that the `Todo.CompleteTodo` and after adding this one, the `Todo.Logout` action methods are missing.

```
func (c Todo) CompleteToDo() revel.Result {
    c.Request.ParseForm()
    for i, _ := range c.Request.Form {
        t, _ := strconv.Atoi(i)
        t--
        todoList[t].Completed = true
    }

    return c.Redirect(Todo.Index)
}
```

```
func (c Todo) Logout() revel.Result {
    for k := range c.Session {
        delete(c.Session, k)
    }
    return c.Redirect(App.Index)
}
```

6.17 Adding routes and safety

The application is already running at full functionality. However, a few minor steps need to be taken. This will illustrate the [filter](#) and [interceptor](#) feature of the Revel Framework.

At first we need to add the following line to our `GOPATH/todo/conf/routes` file:

```
GET      /todo                                Todo.Index
```

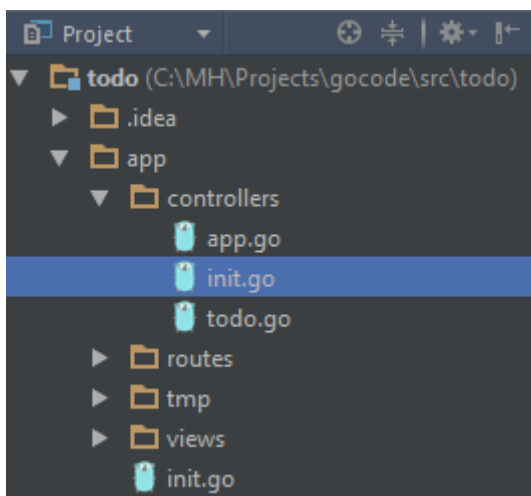
We then implement a `CheckLogin` action method in our `ToDo-Controller` which will check whether a user is logged on:

```
func (c Todo) CheckLogin() revel.Result {
    if _, ok := c.Session["userName"]; ok {
        return nil
    }

    c.Flash.Error("Please log in first!")
    return c.Redirect(routes.App.Index())
}
```

6.18 Adding Interceptors to our Application

In order to use our `CheckLogin` method from the previous chapter, we need to create a file called `init.go` in our `GOPATH/src/todo`. In this file we can register [interceptors](#) methods:



An interceptor is implemented as a filter. However, in comparison to a filter for e.g. the session filter, an interceptor has a receiver type of a concrete controller. In this case the `CheckLogin` method only applies to routes which are served by the `ToDo-Controller`. This is of course reasonable, since the method redirects to the `App.Index` if the user is not logged on:

```
package controllers

import (
    "github.com/revel/revel"
)

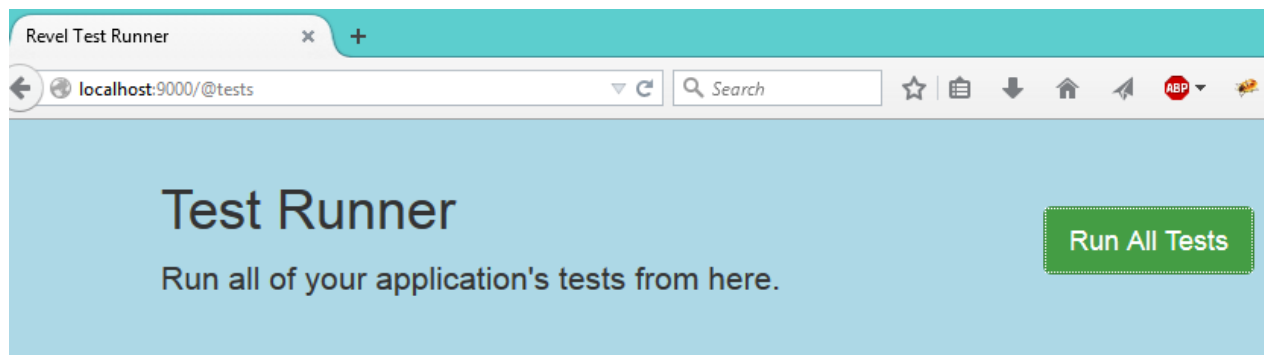
func init() {
    revel.InterceptMethod(Todo.CheckLogin, revel.BEFORE)
}
```

6.19 Adding a Test to the application

We add a simple test `TestThatAddTodoWorks` to the existing one:

```
func (t *AppTest) TestThatAddTodoWorks() {  
    t.PostForm("/Todo/AddTodo", url.Values{  
        "todo.Name" : {"Read Emails"},  
    })  
    t.AssertStatus(201)  
}
```

By adding `/@tests` to the url of our application, we are able to invoke the test framework of Revel directly in our browser. Next to the error template there is also a test template provided:

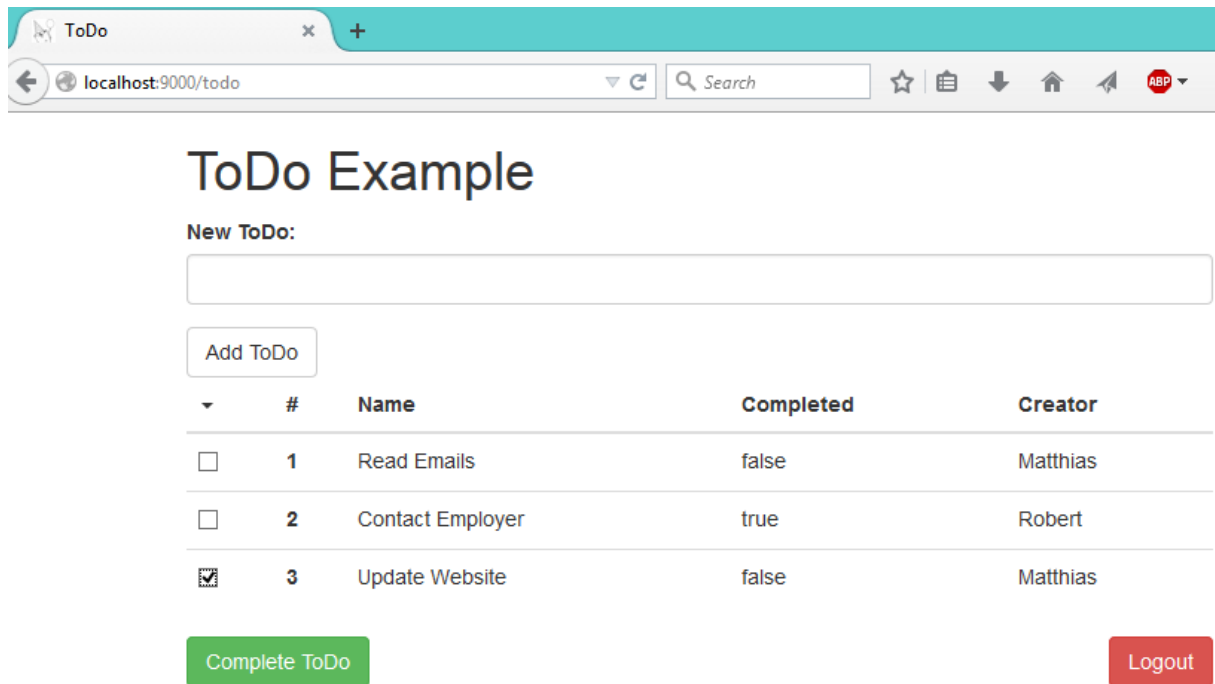


AppTest

TestThatAddTodoWorks	Run
TestThatAppIndexPageWorks	Run

6.20 The ToDo Application

You are now able to add a todo to your list of todos sharing them with all who are also logged in:



ToDo Example

New ToDo:

Add ToDo

	#	Name	Completed	Creator
<input type="checkbox"/>	1	Read Emails	false	Matthias
<input type="checkbox"/>	2	Contact Employer	true	Robert
<input checked="" type="checkbox"/>	3	Update Website	false	Matthias

Complete ToDo

Logout

6.21 What's next?

Based on this application you can now learn further feature of the Go Language and the Revel Framework by adding them to your application. For example adding a persistence layer with [gorp](#) or adding [websocket](#) support.

7 Learning Go and Revel

In this chapter I would like to highlight the learning resources I used to learn the Go language and the Revel Framework.

7.1 Books (<https://github.com/golang/go/wiki/Books>)

- Learning Go by *Miek Gieben*
(<https://github.com/miekg/gobook>)
- An Introduction to Programming in Go by *Caleb Doxsey*
(<http://www.golang-book.com/>)
- Programming in Go: Creating Applications for the 21st Century by *Mark Summerfield*
(<http://www.amazon.com/Programming-Language-Phrasebook-Developers-Library/dp/0321817141>)

7.2 Online

- [A Tour of Go](#) # Interactive only tutorial
- [Effective Go](#) # Document about how to write effective go
- [StackOverflow Tag](#) # StackOverflow overview of learning materials

7.3 Revel

- [Revel Tutorial](#) # Installing Revel and creating an application
- [Revel Manual](#) # Overview of the components provided by the framework

8 Comparison of IDEs

I evaluated a few IDEs before I started to write applications in Go and later in Revel.

The following tables are from a Blog Post¹ and give a nice overview.

IDE	License	Windows	Linux	Mac OS X	Other Platforms
SublimeText 2	Proprietary software	Yes	Yes	Yes	-
TextMate	Proprietary software	No	No	Yes	-
IntelliJ	Apache 2.0	Yes	Yes	Yes	JVM
LiteIDE	LGPL	Yes	Yes	Yes	-
Intype	New BSD Licence	Yes	No	No	-
Netbeans	Free	Yes	Yes	Yes	JVM
Eclipse	Eclipse Public License 1.0	Yes	Yes	Yes	JVM
Komodo Edit	Proprietary	Yes	Yes	Yes	-
Zeus	Proprietary	Yes	Yes	No	-

IDE	Debugger	Syntax Highlighting	Code Completion
SublimeText 2	No	Yes	Yes
TextMate	No	Yes	No
IntelliJ	No	Yes	Yes
LiteIDE	Yes	Yes	Yes
Intype	No	Yes	No
Netbeans	No	Yes	Yes
Eclipse	Yes	Yes	Yes
Komodo Edit	No	Yes	Yes
Zeus	Yes	Yes	Yes

¹ Source: <http://geekmonkey.org/articles/20-comparison-of-ides-for-google-go>

7 Comparison of IDEs

The below mentioned preferences of IDEs are my personal opinion after developing roughly two weeks with the new language.

7.1 Personal Preference

- **JetBrains IDE** (WebStorm/IntelliJ) with the **go-lang-plugin**
(<https://github.com/go-lang-plugin-org/go-lang-idea-plugin>)
[90.000+ Downloads 03/16/2015]
- **SublimeText** with the **GoSublime** plugin
(<https://github.com/DisposaBoy/GoSublime>)
- **Eclipse** with **GoClipse**
(<https://code.google.com/p/goclipse/>)
- **LiteIDE** (<https://code.google.com/p/liteide/>)
- **ZEUS** (<http://www.zeusedit.com/go.html>)
- **Online-DIE** (<http://play.golang.org/>)

For Web Development I recommend the use of a **JetBrains IDE** of your choice with the **go-lang-plugin** and as second choice **SublimeText**. For other purposes of programming I recommend the use of **Eclipse**, **LiteIDE** and **Zeus**, which make use of the **GDB** Debugger.

7.2 Further Reading

- **List of IDEs and Plugins for Go**
<https://code.google.com/p/go-wiki/wiki/IDEsAndTextEditorPlugins>
- **A blog post comparing a IDEs**
<http://geekmonkey.org/articles/20-comparison-of-ides-for-google-go>

8 Debugging Go Code with GDB

GDB does not understand **Go programs** well. The stack management, threading, and runtime contain aspects that differ enough from the execution model GDB expects that they **can confuse the debugger**, even when the program is compiled with gccgo. As a consequence, although GDB can be useful in some situations, it is **not a reliable debugger for Go programs**, particularly heavily concurrent ones. Moreover, it is not a priority for the Go project to address **these issues**, which are difficult. [...]

In time, a more Go-centric debugging architecture may be required."

Source: <https://golang.org/doc/gdb>

* text formatting not in the original source

9 A HTTP Server in native Go

At the end of this tutorial I would like to highlight Go's built-in "net/http"-package. This package provides full functionality to create a web server. Adding routing functionality, e.g. provided by the [Gorilla Framework](#), would allow for a good development workflow of an application already.

A HTTP-Server in Go

```
package main

import (
    "fmt"
    "io"
    "net/http"
)

func hello(res http.ResponseWriter, req *http.Request) {
    res.Header().Set(
        "Content-Type",
        "text/html",
    )
    io.WriteString(
        res,
        `<doctype html>
        <html>
        <head>
            <title>Hello World</title>
        </head>
        <body>
            Hello World!
        </body>
        </html>`,
    )
    fmt.Println(req)
}

func main() {
    http.HandleFunc("/", hello)
    http.ListenAndServe(":9000", nil)
    if err != nil {
        log.Fatal("ListenAndServe: ", err)
    }
}
```

A minimal HTTP-Server in Go

```
package main

import (
    "fmt"
    "net/http"
    "html"
)

func main() {
    http.HandleFunc("/bar", func(w http.ResponseWriter, r
    *http.Request) {
        fmt.Fprintf(w, "Hello, %q", html.EscapeString(r.URL.Path))
    })
    http.ListenAndServe(":9000", nil)
}
```