

— Introduction lecture



# A bit of history

What is Golang, What it's purpose?

Golang United

# Designed By



# Golang United

## Ken Thompson

- Designed and implemented the original **Unix OS**
- One of the creators and early developers of the **Plan 9 OS**
- Invented the **B programming language**
- contributions included his work on **regular expressions**
- Worked on **UTF-8**

## Rob Pike

- member of the **Unix** team
- Involved in the creation of the **Plan 9**
- **Sam** and **Acme** text editors
- Co-author of **The Practice of Programming** and **The Unix Programming Environment** books.
- Co-creator of **UTF-8**

## Robert Griesemer

- The Google's **v8** JavaScript engine
- The **Sawzall** language
- The **Java HotSpot** virtual machine
- The **Strongtalk** system.

# Golang United

12 years of jorney

## 2009 Public release

Golang become open-source project

## 2015 1.5 release

In 1.4 and 1.5 version internals of the language was refactored, all language tooling was rewritten in Go.

## 2022 1.18 release

Generic types and fuzzing testing

## 2007 Development started

Development started as internal project at Google

## 2012 v1.0 release

Golang released v1.0, language team promised compatibility of all 1.\* versions

## 2018 v1.11 release

Go modules release, this change improved dependency management a lot





# Why Golang is so good?

*Typing, Compilation, Concurrency, Standard library*

# Golang United



*No legacy.*



*Golang is used a lot for  
cloud native development*



*Golang is simple and fast*



*Concurrent by design*

# Golang United

*In real world Golang used a lot, for writing distributed systems.*

*Such companies as (Google, Lyft, Uber, Twitter and Facebook, Gitlab) already use Golang for their systems.*

*A lot of systems which drive current world such as Docker, Kubernetes and Google Cloud written at Golang as well.*



# Golang United

Huge standard library(<https://pkg.go.dev/std>):

- <https://pkg.go.dev/compress@go1.17.6> – compression via different algorithms
- <https://pkg.go.dev/crypto@go1.17.6> – supports many cryptographic algorithms
- <https://pkg.go.dev/encoding@go1.17.6> – supports many encoding formats
- <https://pkg.go.dev/net/http@go1.17.6> – built-in http server with http1.1/http2 support
- <https://pkg.go.dev/html@go1.17.6> – built-in html template engine
- <https://pkg.go.dev/text@go1.17.6> – built-in text template engine
- <https://pkg.go.dev/database/sql@go1.17.6> – built-in sql library



# The key differences of Golang

Type system, Interfaces, Concurrency

# Golang United

*Golang is explicit language*

*Code is the main documentation of itself*

*By default, all parameters passed by value not a reference*

*Using value type in most cases faster then use references*

*Code generation over generic programming (So far)*

*Data structures is simple, there is no such Collection framework (so far)*

# Golang United

Strong typing – every variable in golang has type

```
var d1 int64  
var d2 int32  
var d3 int
```

Type system

```
var (  
    d1 int64  
    d2 int32  
)  
fmt.Println(d1 == d2)
```

Compilator do the work for you

# github.com/burov/snippets/webserver  
[./main.go:22:17](#): invalid operation: d1 == d2 (mismatched types int64 and int32)

# Golang United

Concurrency dramatically simple

```
var wg sync.WaitGroup
message := []string{"Hello", "from", "Golang", "United", "Team"}
for _, str := range message {
    wg.Add(delta: 1)

    go func(s string) {
        fmt.Println(s)
        wg.Done()
    }(str)
}

wg.Wait()
```

## Output:

Team  
Hello  
United  
Golang  
from

# Golang United

An example of simple web-server

```
package main

import (
    "fmt"
    "net/http"
)

func main() {

    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        _, _ = fmt.Fprintf(w, format: "Hello World")
    })

    host := "localhost:8080"
    fmt.Printf( format: "Listen and Serve on %q\n", host)
    if err := http.ListenAndServe(host, handler: nil); err != nil {
        panic(err)
    }
}
```

```
% curl http://localhost:8080/
Hello World
```



# Golang comparison

# Golang United

Strong typing – every variable in golang has type

```
package com.company;

public class Main {

    public static void main(String[] args) {
        Integer d1 = 0;
        long d2 = 0;

        System.out.println(d1 == d2);
    }
}
```

Output

true

# Golang United

Strong typing – every variable in golang has type

```
var d1 int64  
var d2 int32  
var d3 int
```

Type system

```
var (  
    d1 int64  
    d2 int32  
)  
fmt.Println(d1 == d2)                                Compilator do the work for you
```

# github.com/burov/snippets/webserver  
[./main.go:22:17](#): invalid operation: d1 == d2 (mismatched types int64 and int32)

# Golang United

Java:

- Types system is very complex, some of the decisions have to be made, due to the initial design weaknesses

Golang:

- Type system really simple and explicit, there are no hacks in it

# Golang United

```
package main

import "fmt"

type Printer interface {
    Print(s string)
}

type ConsolePrinter struct{}

func (c *ConsolePrinter) Print(s string) {
    fmt.Println(s)
}

func main() {
    var p Printer = &ConsolePrinter{}

    p.Print(s: "Hello World")
}
```

```
package com.company;

interface Printer {
    public void print(String s);
}

class ConsolePrinter implements Printer {
    public void print(String s) {
        System.out.println(s);
    }
}

public class Main {

    public static void main(String[] args) {
        Printer printer = new ConsolePrinter();

        printer.print("Hello World");
    }
}
```

# Golang United

Java:

- Interface should be defined by producer
- Interface should be explicitly implemented by Class

Golang:

- Interface might be defined by consumer or producer
- Interface implicitly implemented if type has all the methods defined in interface (Duck typing)

# Golang United

## Concurrency in Java

```
package com.company;

public class Main {

    public static void main(String[] args) {
        var message = new String[]{"Hello", "from", "Golang", "United", "Team"};

        for (String msg : message) {
            new Thread(() -> {
                System.out.println(msg);
            }).start();
        }
    }
}
```

### Output:

Team  
Hello  
United  
Golang  
from

# Golang United

Concurrency dramatically simple

```
var wg sync.WaitGroup
message := []string{"Hello", "from", "Golang", "United", "Team"}
for _, str := range message {
    wg.Add(delta: 1)

    go func(s string) {
        fmt.Println(s)
        wg.Done()
    }(str)
}

wg.Wait()
```

## Output:

Team  
Hello  
United  
Golang  
from

# Golang United

Java:

- Create system thread for each Thread object
- Static stack
- Communication by shared memory

Golang:

- Go routines is a local object, no system thread creation
- Dynamic stack, each go routine can have stack size up to a few gigabytes
- Communication by shared memory or built-in messaging (channels)

# Golang United

An example of simple web-server in Java

```
package org.example;

public class App {
    public static String getHello() { return "Hello world"; }
}
```

```
% curl http://localhost:8080/
Hello World
```

# Golang United

An example of simple web-server

```
package main

import (
    "fmt"
    "net/http"
)

func main() {

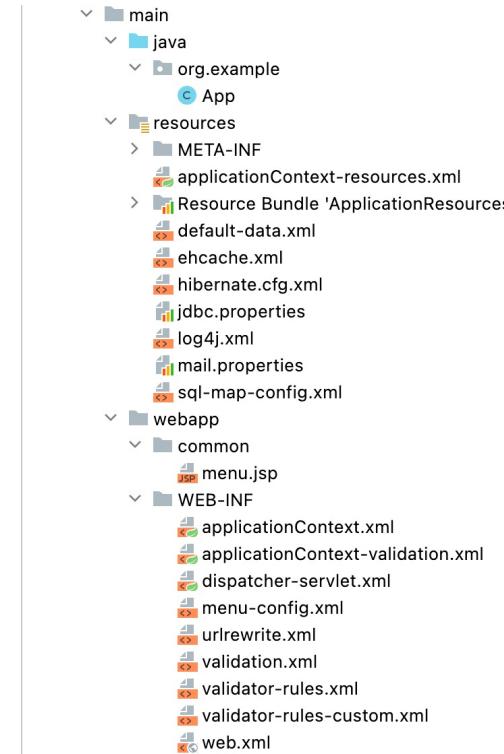
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        _, _ = fmt.Fprintf(w, format: "Hello World")
    })

    host := "localhost:8080"
    fmt.Printf( format: "Listen and Serve on %q\n", host)
    if err := http.ListenAndServe(host, handler: nil); err != nil {
        panic(err)
    }
}
```

```
% curl http://localhost:8080/
Hello World
```

# Golang United

## Web project configuration in Java



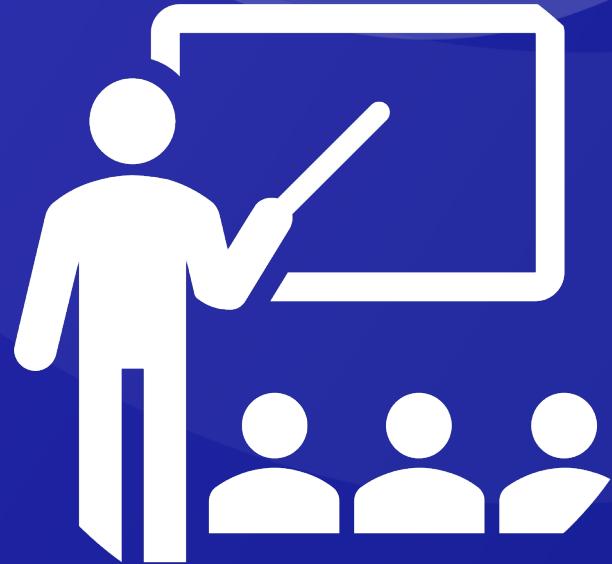
# Golang United

## Golang:

- Built-in server
- Additional configuration isn't required
- No external dependencies
- Build just with plain compiler

## Java:

- Additional server required before servlets container
- Required configuration for frameworks
- Servlets containers and JavaEE libs required
- Gradle or Maven required to build project



## About the course

How the process looks like?

# What you will learn

- 0. Introduction lecture
- 1. Golang basic syntax
- 2. Golang memory model
- 3. Golang data structures
- 4. Golang byte, rune, string & error
- 5. Golang structures & methods & interfaces
- 6. Golang interfaces & OOP in Golang
- 7. STD packages io, context, os
- 8. Database & How to use in Golang
- 9. Web development with Golang
- 10. Concurrency in Golang

#First stage

#Individual phase

# How you will learn

- **1** lecture in a week
- Homework each week
- Additional materials after each week
- Test after some lectures

# What you will learn

- How to cooperate inside a team
- How to build scalable, distributed applications
- How to build CI/CD pipelines
- How to work in an agile environment
- Will work with such tools as Docker, Gitlab CI/CD

#Second stage

#Team work

# How to

How to get your place in **#second\_stage** shuttle ?

- Complete all your homework, on **last lecture date + 2 weeks**
- Be prepared to speak about what you've learned

# Development environment

## Goland



- *Install Golang* -  
<https://golang.org/doc/install>
- *Install IDE* -  
<https://www.jetbrains.com/go/download>
- *Students license* -  
<https://www.jetbrains.com/student/>

# Development environment

## Visual Studio Code



Visual Studio Code

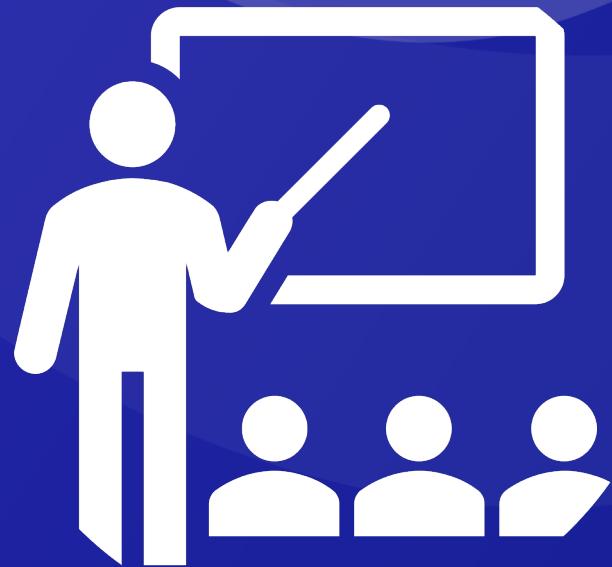
- *Install Golang* - <https://golang.org/doc/install>
- *Install VS Code* - <https://code.visualstudio.com/download>
- *Add support for Golang* - <https://code.visualstudio.com/docs/languages/go>

# Development environment

## Vim



- *Install Golang -*  
<https://golang.org/doc/install>
- *Install Vim -*  
<https://www.vim.org/download.php>
- *Generate .vimrc file and download it* <https://vim-bootstrap.com> *or install go vim plugin* [https://github.com/fatih/vim-go  
\*\(this is more for vim experts\)\*](https://github.com/fatih/vim-go)



# Homework

# Golang United

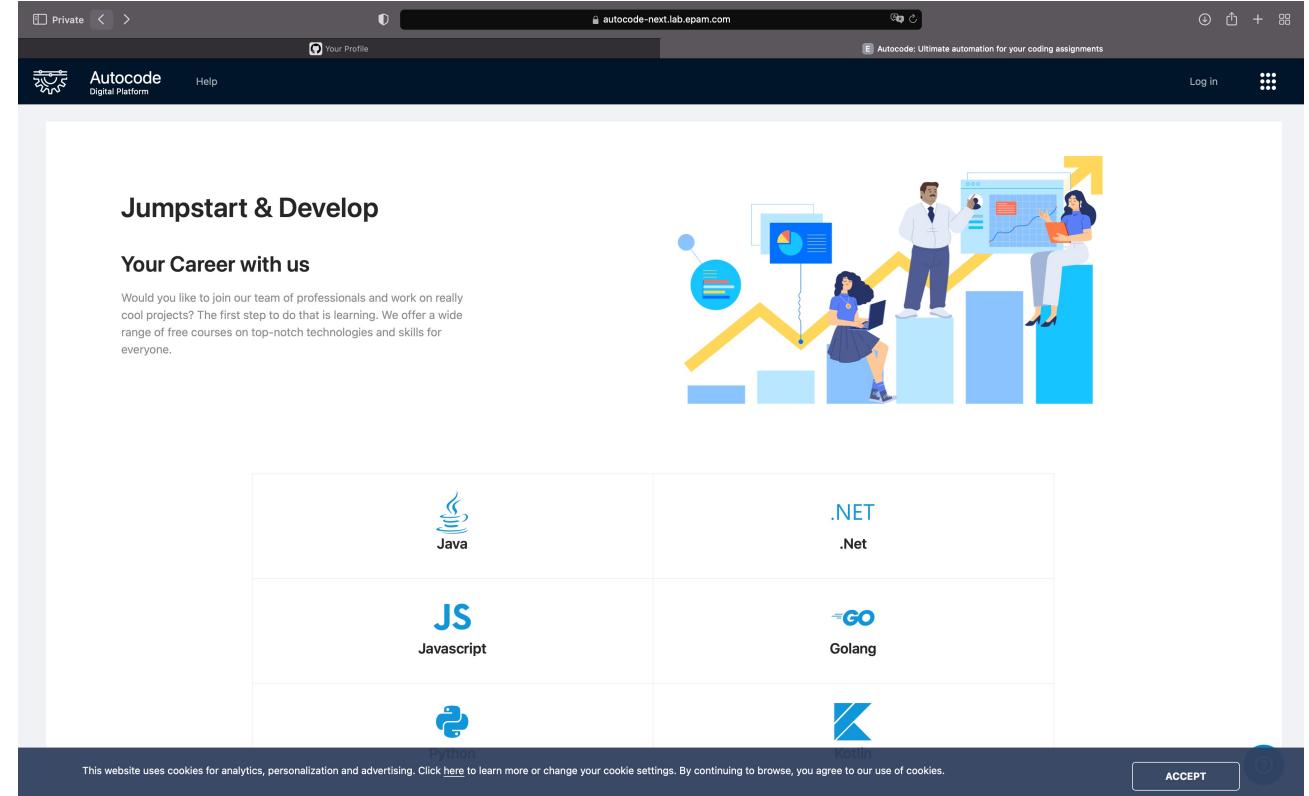
<https://autocode-next.lab.epam.com>

- Complete task for "Introduction module"
- Find instruction below

# Golang United

Find platform for homework verification on

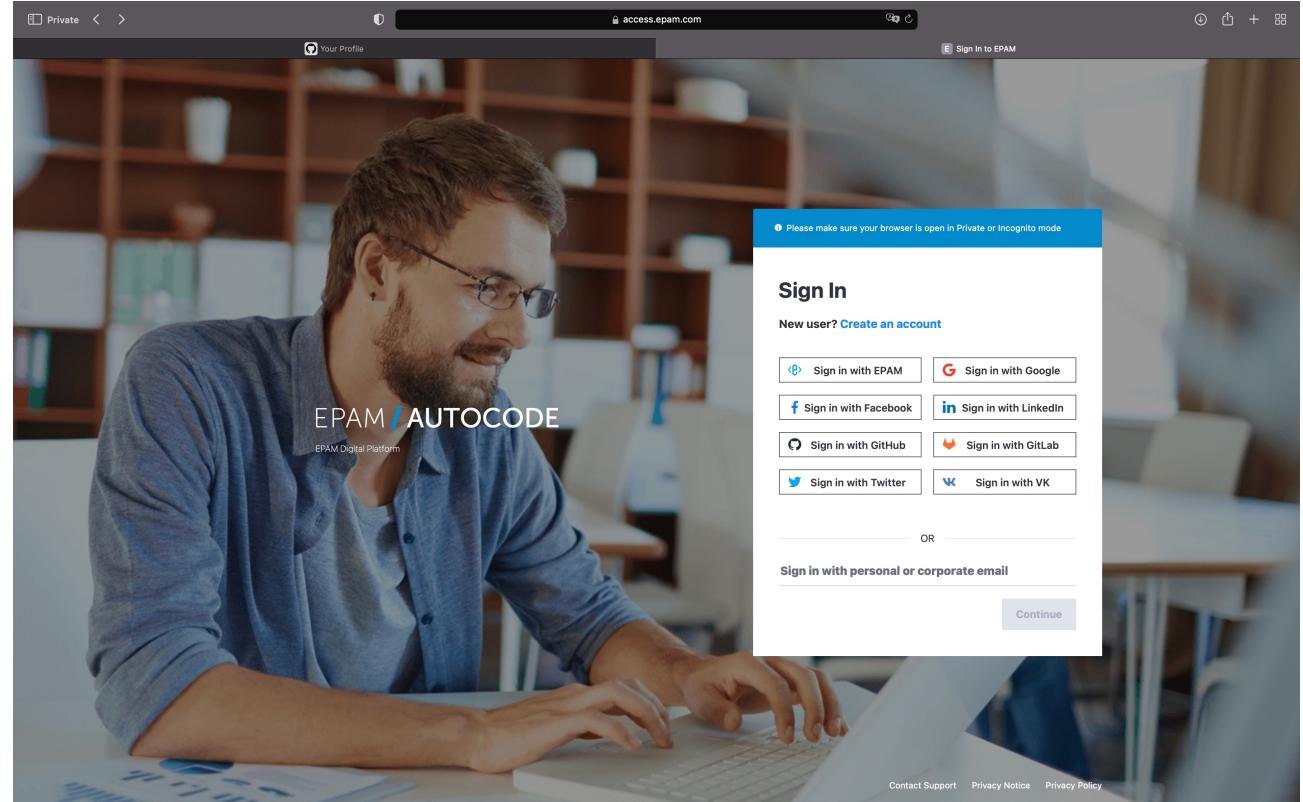
<https://autocode-next.lab.epam.com>



# Golang United

Login to the platform

- Use **github** or **gitlab** account



# Golang United

## Main page

- Verify that you're signed-in

The screenshot shows the Autocode Digital Platform main page. At the top, there's a navigation bar with links for "Your Profile", "Autocode Digital Platform", "My Learning", "My Contribution", "Explore", and "Help". A user profile for "Aliaksei Bura" is visible on the right. The main content area features a section titled "Jumpstart & Develop" with the sub-section "Your Career with us". Below this, there's a brief description: "Would you like to join our team of professionals and work on really cool projects? The first step to do that is learning. We offer a wide range of free courses on top-notch technologies and skills for everyone." To the right of the text is a colorful illustration of three people working on laptops on a bar chart, with a large upward-pointing arrow. Below this section is a grid of six boxes representing different technologies:

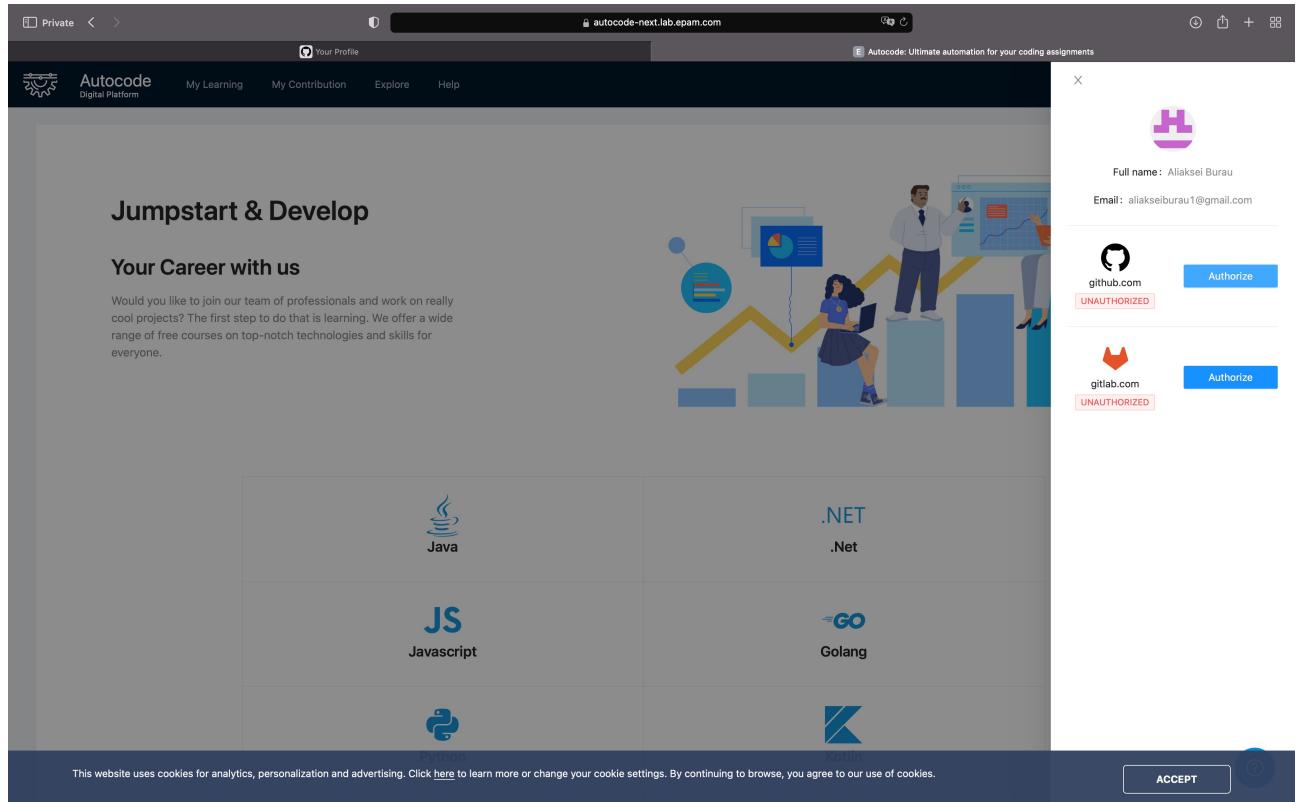
Java	.NET
Javascript	Go
Python	Kotlin

At the bottom of the page, a cookie consent banner reads: "This website uses cookies for analytics, personalization and advertising. Click [here](#) to learn more or change your cookie settings. By continuing to browse, you agree to our use of cookies." It includes "ACCEPT" and "REFUSE" buttons.

# Golang United

Authenticate platform to your github/gitlab account

- Go to "My Profile" tab
- Give permissions to the **github or gitlab**
- Avoid using both



# Golang United

Find the course

- Go to the **Explore** tab
- Find course with name

**"Golang United School - first stage"**

The screenshot shows the Autocode Digital Platform interface. The top navigation bar includes 'Private', 'Your Profile', 'Autocode Digital Platform', 'My Learning', 'My Contribution', 'Explore' (which is highlighted in blue), and 'Help'. On the right, there's a user profile for 'Aliaksei Bureau' and various icons. The main content area has a search bar and tabs for 'Courses' and 'Single tasks'. A sidebar on the left contains filters for 'Course status' (Draft, Published, In progress, Finished, Canceled), 'Education type' (Self-paced, Virtual class), and 'Tags' (a long list including .NET, Java, Python, Golang, Javascript, PHP, Ruby, Kotlin, Scala, Typescript, Software Development, Software Testing, Business Analysis, Experience Design, Backend, Frontend, Mobile, Test Automation, Technical Writing, Demo). The main list shows one course: 'Golang United School - first stage', which is 'In progress', 'Online', and scheduled from 'February 11, 2022 - May 31, 2022'. At the bottom, there are pagination controls ('Total 1 items', page 1, '20 / page'), a cookie consent banner, and an 'ACCEPT' button.

# Golang United

Join the course

- Press "enroll" button

The screenshot shows a web browser window for the Autocode Digital Platform at the URL [autocode-next.lab.epam.com](https://autocode-next.lab.epam.com). The page title is "Golang United School - first stage". The navigation bar includes links for "Your Profile", "Autocode Digital Platform", "My Learning", "My Contribution", "Explore", and "Help". On the right side, there's a user profile for "Aliaksei Bura" with a purple icon, and a "Leave course" button.

The main content area displays the syllabus for the "Golang United School - first stage". It shows a progress bar indicating "Included" tasks (0 / 1). Below the progress bar, a sidebar lists numbered topics from 00\_introduction to 10\_concurrency. A message box says "Select task from the left-side panel to display details". At the bottom of the page, a cookie consent banner states: "This website uses cookies for analytics, personalization and advertising. Click [here](#) to learn more or change your cookie settings. By continuing to browse, you agree to our use of cookies." with "ACCEPT" and "REFUSE" buttons.

# Golang United

Choose the task

- Find task for lecture
- Press the **start** button

The screenshot shows a web browser window for the "Golang United School - first stage" on the "autocode-next.lab.epam.com" domain. The user profile "Aliaksei Burau" is logged in. The "Syllabus" tab is active, displaying a list of chapters:

- 00\_introduction (selected)
- 01\_basic\_syntax
- 02\_memory\_model
- 03\_data\_structures
- 04\_strings\_errors
- 05\_structures\_methods
- 06\_interfaces\_oop
- 07\_io\_context\_os
- 08\_databases
- 09\_web\_development
- 10\_concurrency

A specific task, "Hello World", is selected and shown in detail. The task status is "In progress". It was started "a few seconds ago". The task description is "Introduction task". The purpose is to figure out how to use basic tooling of the Golang as well as check your IDE setup. The task details include:

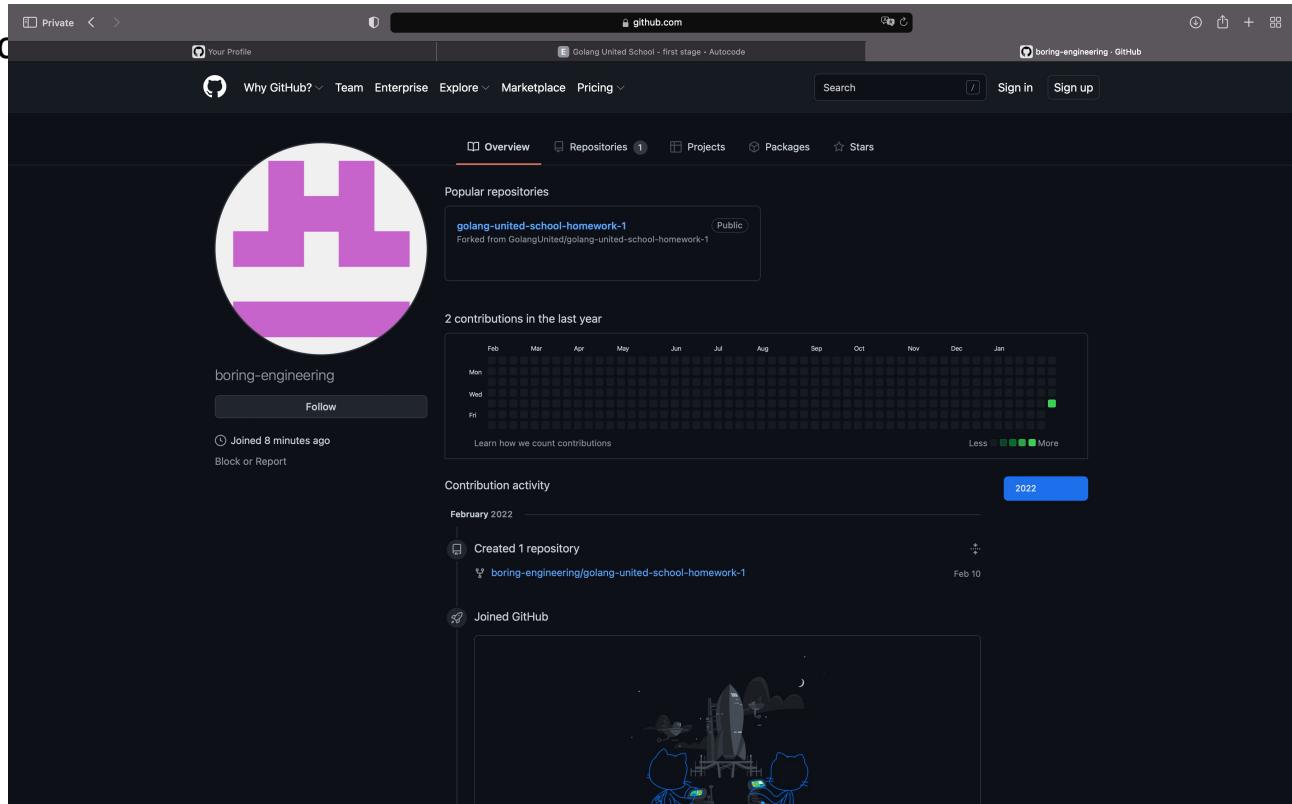
- Task:
  - Initialize project with Go modules
  - Add dependency "github.com/kyokomi/emoji" to add emoji into the string
  - Using Sprint function from this package build a message "Hello ! Hint (code for template engine \"world\_map\")"

Below the task details, there are fields for "Minimum pass score" (set to 60) and "Last submission score" (set to 0/100). A "Finish" button is visible. At the bottom, there is a GitHub repository link for "golang-united-school-homework-1" and a "Change branch" button. A cookie consent banner at the bottom states: "This website uses cookies for analytics, personalization and advertising. Click [here](#) to learn more or change your cookie settings. By continuing to browse, you agree to our use of cookies." with "ACCEPT" and "Decline" buttons.

# Golang United

Check that you have forked repo

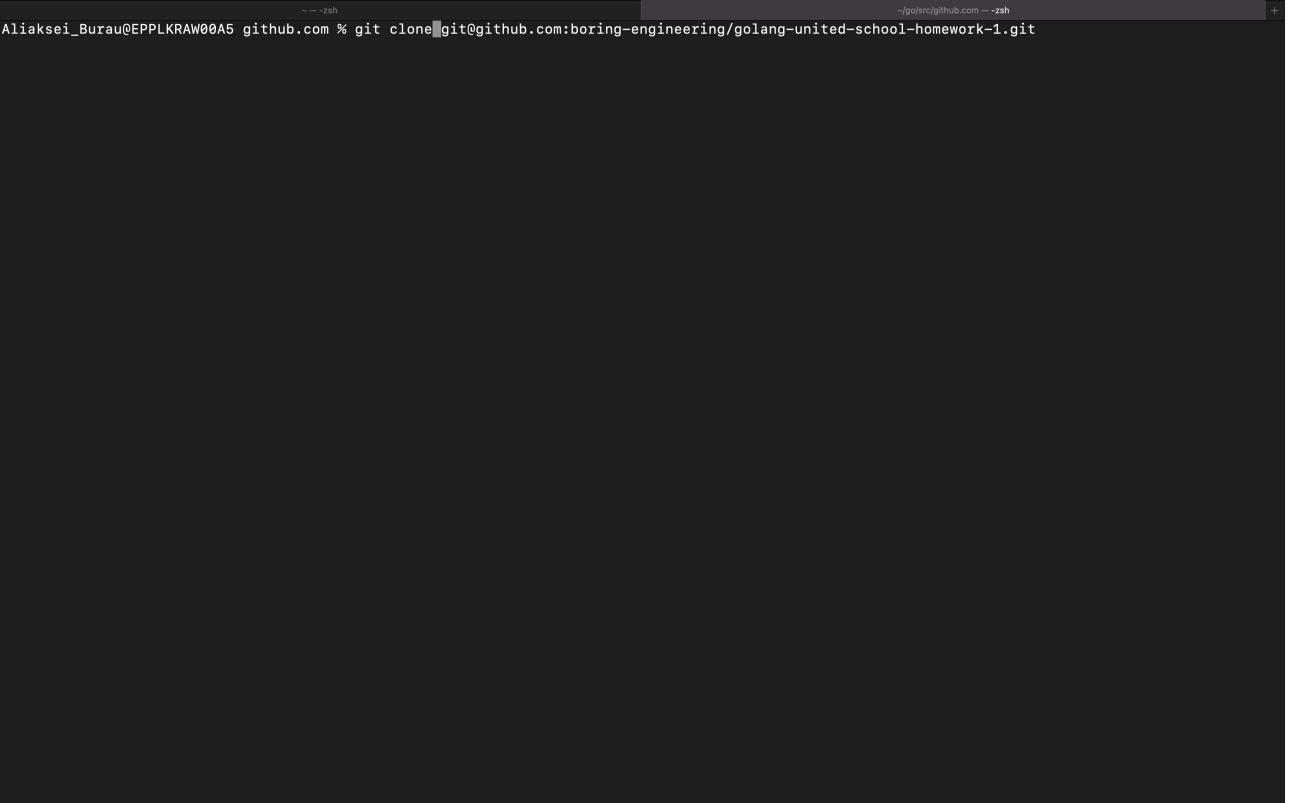
- After that you will see new repository on your account



# Golang United

## Setup laptop

- Clone forked repo to your local machine
- You can use any text editor to write code

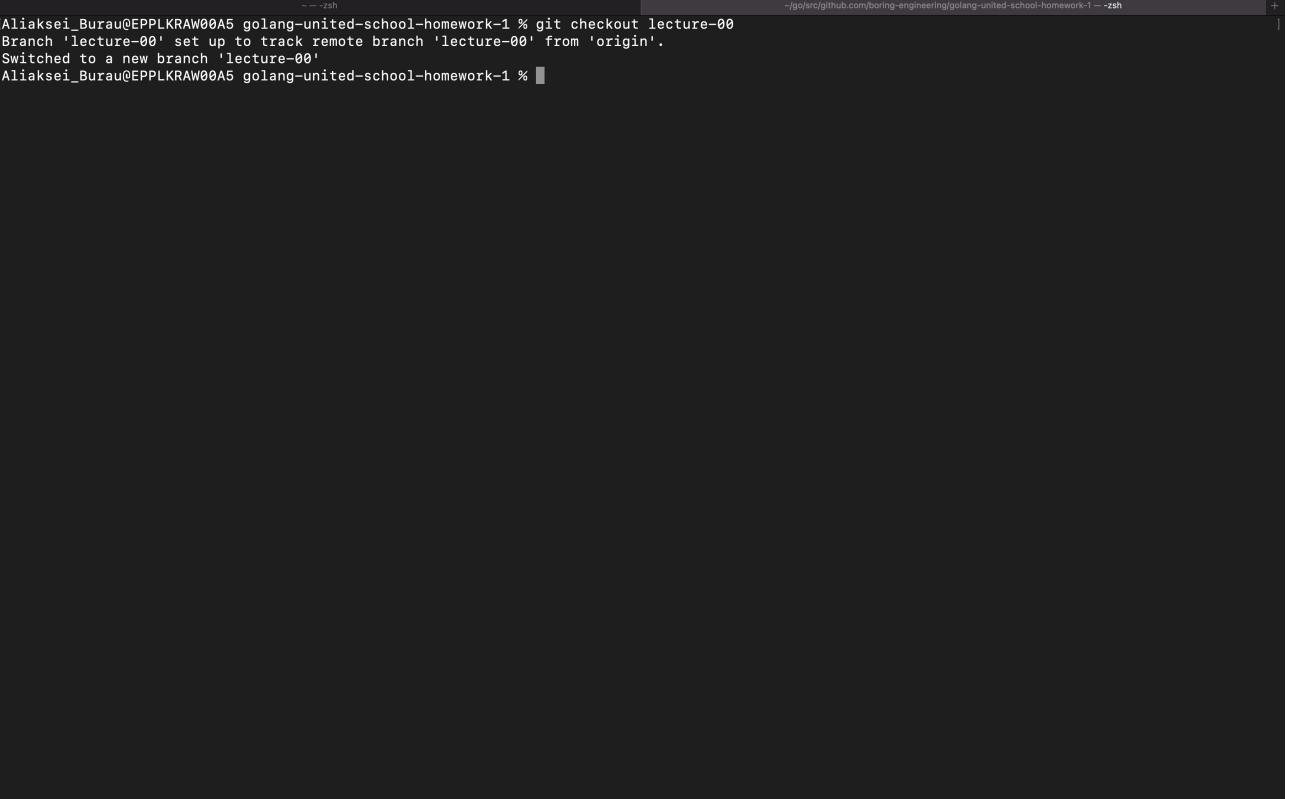


A terminal window showing a git clone command being run. The command is: `git clone git@github.com:boring-engineering/golang-united-school-homework-1.git`. The terminal is titled `~|go/src/github.com -->zsh`.

# Golang United

Switch to branch with task

- Each lecture will be published to the new **branch**
- Find task for "**Introduction lecture**" in branch  
**"lecture-00"**

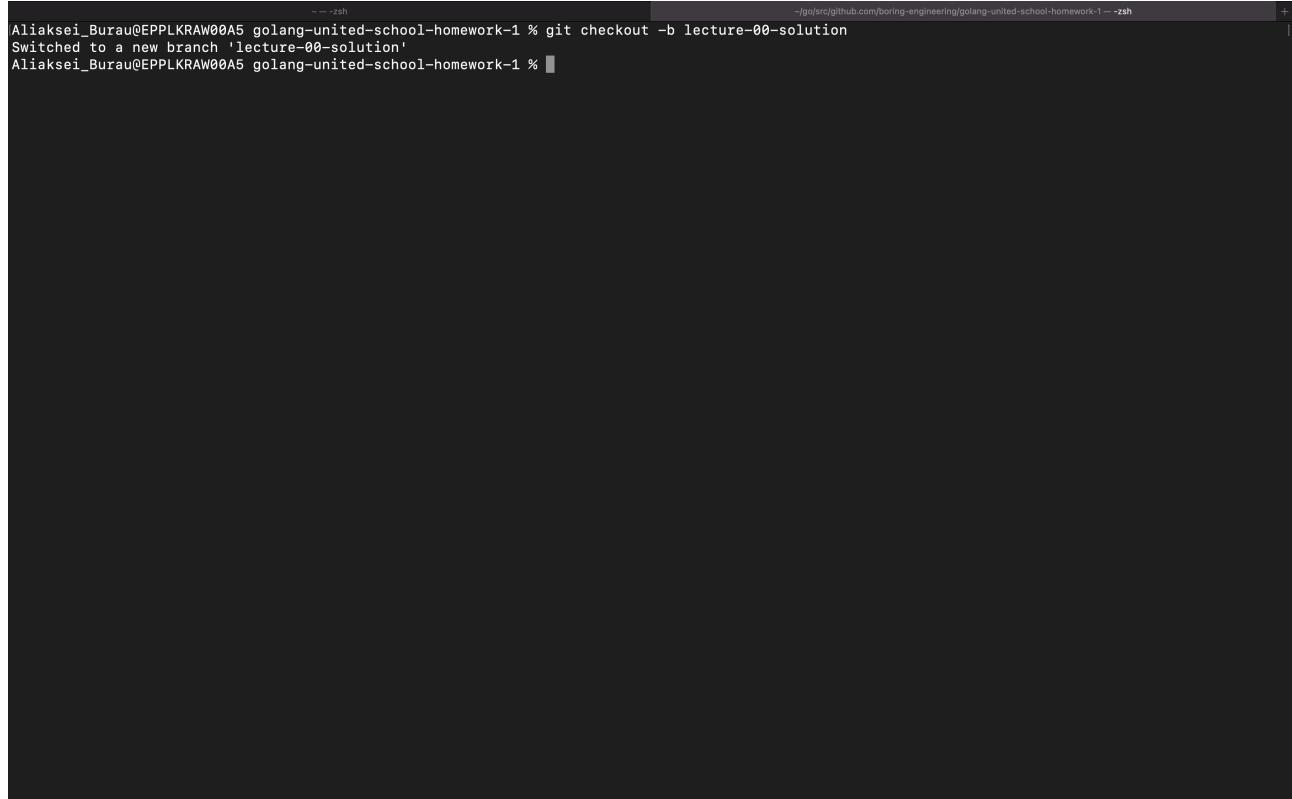


```
Aliaksei_Burau@EPPLKRAW00A5 golang-united-school-homework-1 % git checkout lecture-00
Branch 'lecture-00' set up to track remote branch 'lecture-00' from 'origin'.
Switched to a new branch 'lecture-00'
Aliaksei_Burau@EPPLKRAW00A5 golang-united-school-homework-1 %
```

A screenshot of a terminal window titled "zsh". The window shows a command being run: "git checkout lecture-00". The output indicates that a new local branch named "lecture-00" has been created, tracking a remote branch of the same name from the "origin" repository. The user's name and session ID are visible at the top left, and the full path of the repository is at the top right.

# Golang United

- To avoid conflicts please don't push code into the same branch
- Create new one f.e. "lecture-00-solution"



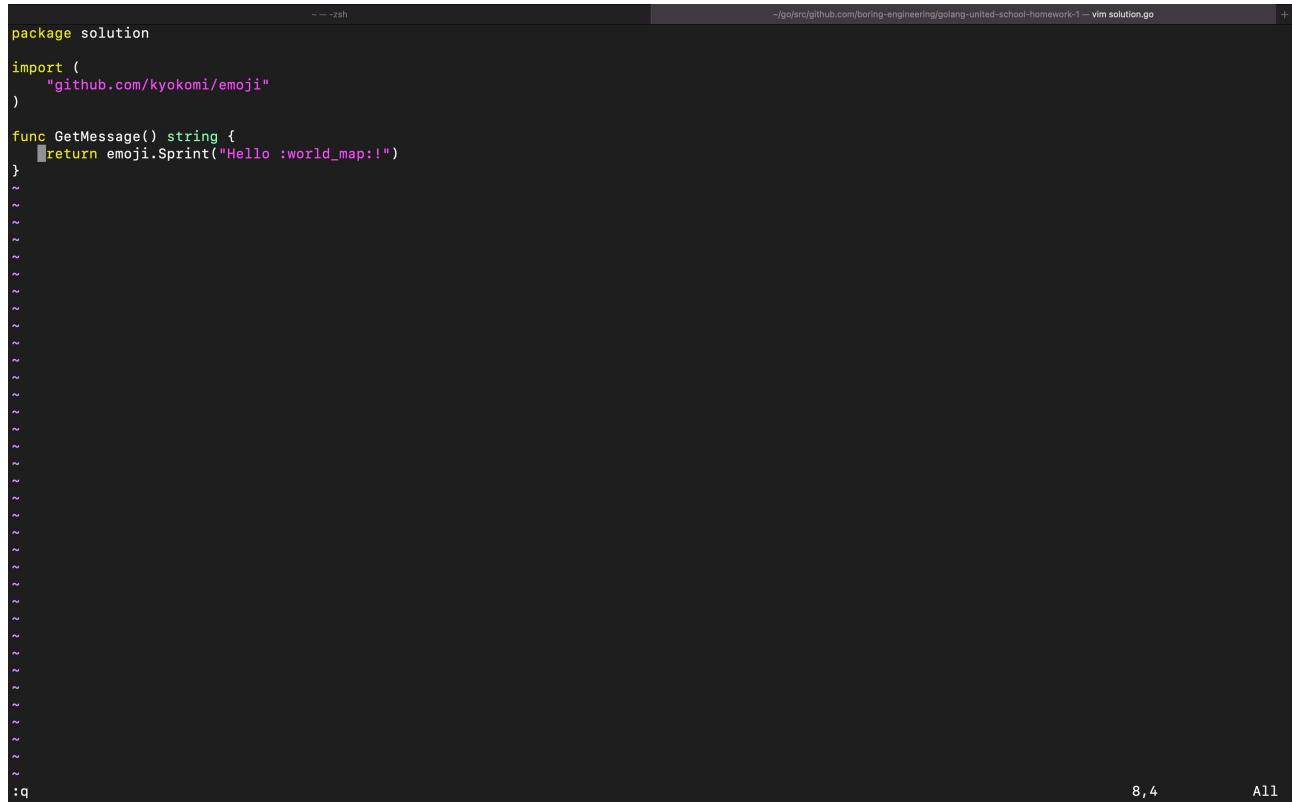
```
Aliaksei_Bureau@EPPLKRAW00A5 golang-united-school-homework-1 % git checkout -b lecture-00-solution
Switched to a new branch 'lecture-00-solution'
Aliaksei_Bureau@EPPLKRAW00A5 golang-united-school-homework-1 % █
```

A terminal window showing a git command being run. The command is 'git checkout -b lecture-00-solution'. The output shows that the branch 'lecture-00-solution' has been switched to. The terminal is located in a directory 'golang-united-school-homework-1'. The prompt ends with a black square character.

# Golang United

Write code

- In **README.md** you can find instructions
- Complete task accounrding to instructions
- See screenshot for the hints



A screenshot of a terminal window showing a Go code editor in Vim. The code defines a package named 'solution' that imports the 'github.com/kyokomi/emoji' package. It contains a single function 'GetMessage' that returns a string with an emoji. The terminal window has two tabs: one for zsh and another for vim solution.go. The status bar at the bottom shows '8, 4' and 'All'.

```
package solution
import (
    "github.com/kyokomi/emoji"
)

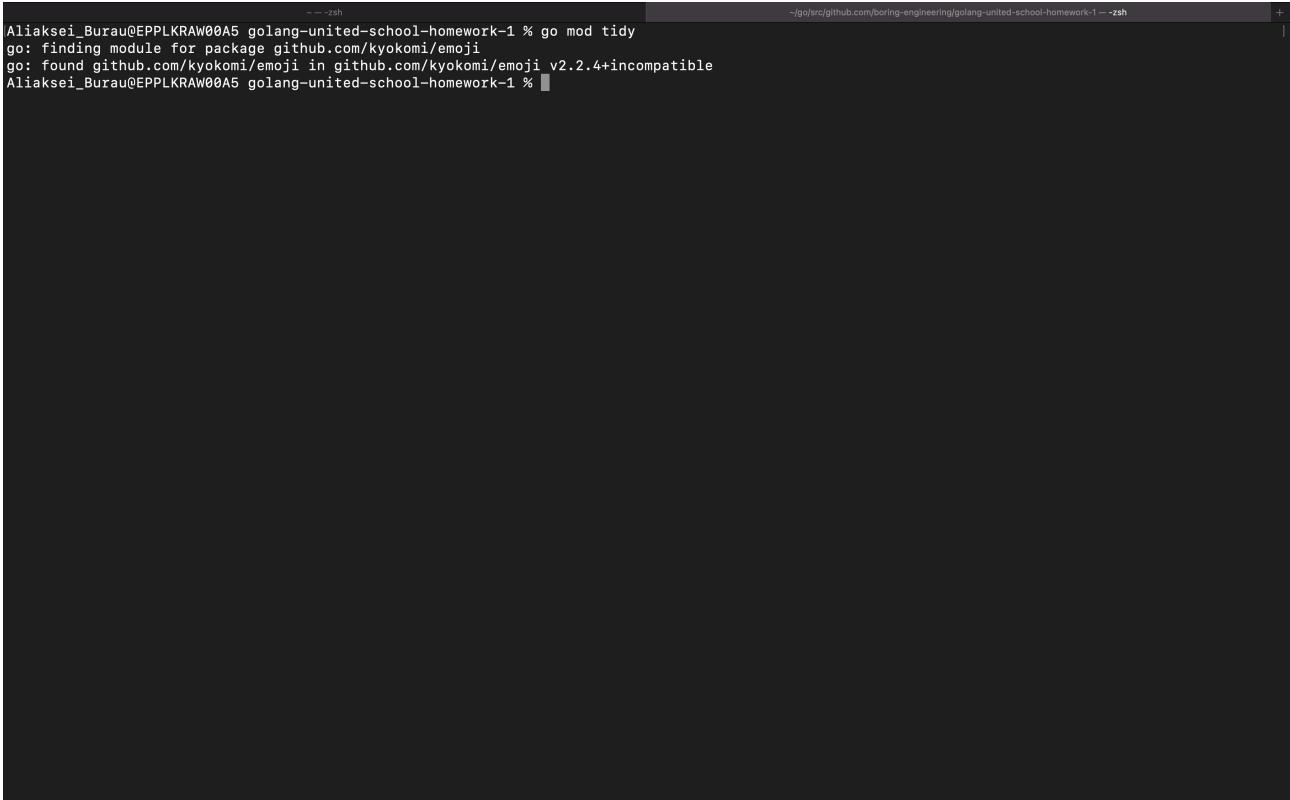
func GetMessage() string {
    return emoji.Sprint("Hello :world_map:!")
}
```

# Golang United

Initialize repository

- Run “**go mod init**” to initialize **Go** project
- Run “**go mod tidy**” to fetch dependencies

You should run this commands in repo



```
Aliaksei_Bureau@EPPLKRAW00A5 golang-united-school-homework-1 % go mod tidy
go: finding module for package github.com/kyokomi/emoji
go: found github.com/kyokomi/emoji in github.com/kyokomi/emoji v2.2.4+incompatible
Aliaksei_Bureau@EPPLKRAW00A5 golang-united-school-homework-1 %
```

# Golang United

## Push the code

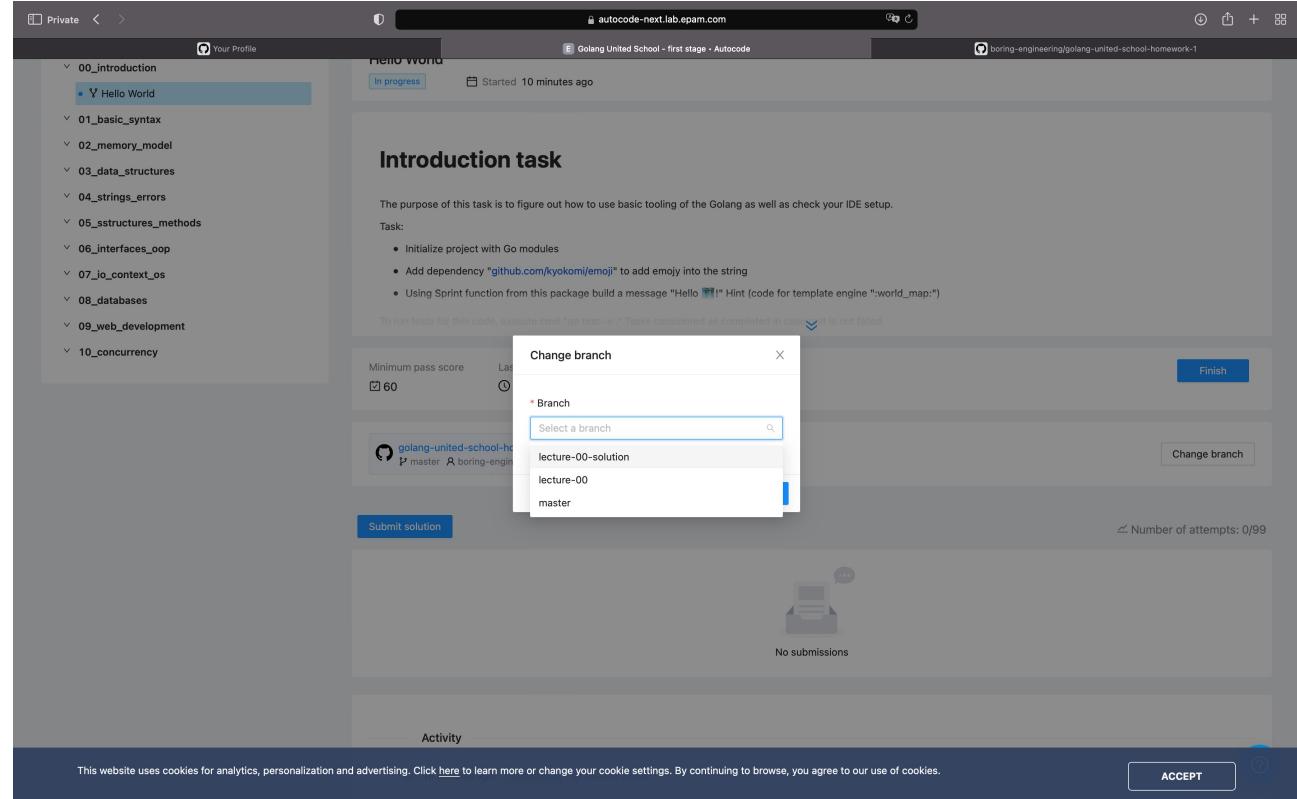
- Push new branch to the server

```
-- zsh
Aliaksei_Bureau@EPPLKRAW00A5 golang-united-school-homework-1 % git push --set-upstream origin lecture-00-solution
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.42 KiB | 1.42 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'lecture-00-solution' on GitHub by visiting:
remote:   https://github.com/boring-engineering/golang-united-school-homework-1/pull/new/lecture-00-solution
remote:
To github.com:boring-engineering/golang-united-school-homework-1.git
 * [new branch]  lecture-00-solution -> lecture-00-solution
Branch 'lecture-00-solution' set up to track remote branch 'lecture-00-solution' from 'origin'.
Aliaksei_Bureau@EPPLKRAW00A5 golang-united-school-homework-1 %
```

# Golang United

## Test submission

- Choose branch with code



# Golang United

## Test submission

- Press button “submit solution”

The screenshot shows a web browser window with the URL `autocode-next.lab.epam.com`. The page title is "Golang United School - first stage - Autocode". The main content area contains a task description and a list of sub-tasks:

**Task:**

- Initialize project with Go modules
- Add dependency "`github.com/kyokomi/emoji`" to add emoji into the string
- Using Sprint function from this package build a message "Hello ! Hint (code for template engine `:world_map:`)"

To run tests for this code, execute cmd: `go test -v`. Tasks considered as completed in case is not failed.

Minimum pass score: 60      Last submission score: 0 / 100

Submit solution

Activity

11 minutes ago In progress by Aliaksei Burau

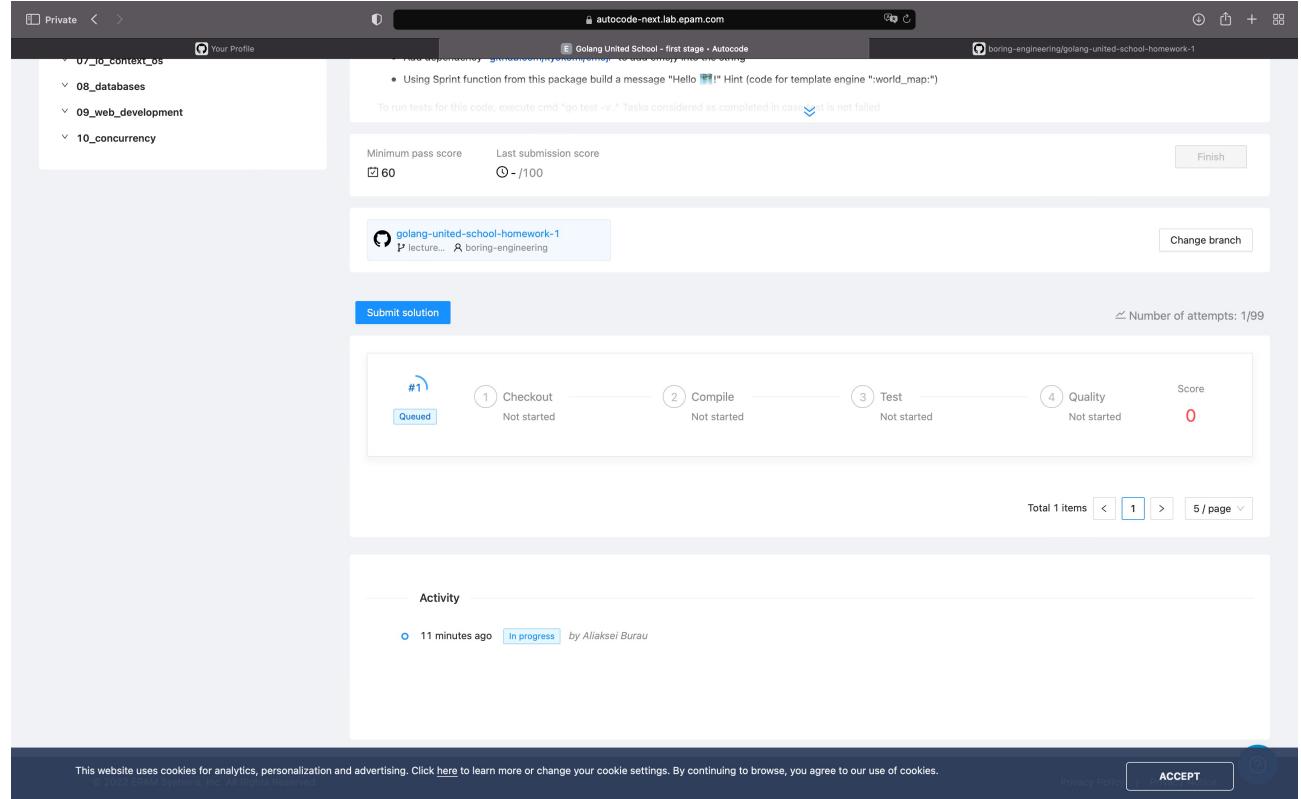
This website uses cookies for analytics, personalization and advertising. Click [here](#) to learn more or change your cookie settings. By continuing to browse, you agree to our use of cookies.

Privacy Policy | **ACCEPT**

# Golang United

Wait for test result

- Press button “submit solution”



# Golang United

## Test submission

- In case you satisfied with mark, press “Finish”

