MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Designing a comprehensive course on Go

Bachelor's Thesis

STANISLAV ZEMAN

Brno, Spring 2024

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Designing a comprehensive course on Go

Bachelor's Thesis

STANISLAV ZEMAN

Advisor: RNDr. Martin Ukrop, Ph.D.

Department of Computer Science

Brno, Spring 2024


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Stanislav Zeman

Advisor: RNDr. Martin Ukrop, Ph.D.

Acknowledgements

I would like to express my gratitude to all my friends, colleagues, and family who have supported, inspired, and motivated me throughout this journey, as this would not be possible without you.

Special thanks to my advisor, Martin Ukrop, for his guidance and extreme patience, and to Pavel Tišnovský and Ivan Nečas for bringing Go to our faculty and laying the foundations for this work.

Abstract

Go is a modern programming language known for its simplicity, efficiency, and concurrency model. It is widely used in the cloud and to build web services and command-line interfaces. This thesis deals with designing and implementing a new course on Go. The course focuses on practical knowledge required for building production applications and, as such, goes beyond the fundamentals of the language itself and explores modern libraries and technologies. Additionally, the design of the course puts the Faculty of Informatics of Masaryk University into context and reflects some of the shortcomings of currently existing courses. The thesis designs and implements a new set of materials, primarily including lectures, some of which are based on an existing Go course by Red Hat, and completely new homework assignments and exercises. All of the created materials collectively constitute a comprehensive course on Go that anyone can adapt, as all of its materials are open-source and publicly available on GitHub.

Keywords

Go, Golang, course, course design, lectures, exercises, homework, projects, programming, Red Hat

Contents

1 Introduction

As the field of software development is rapidly evolving, it is important to be acquainted with the latest technologies and programming languages. This knowledge then allows developers, software architects, and other stakeholders to make better design decisions when building production software.

This thesis introduces a new course focused on the Go[1] programming language, which is a modern language known for its simplicity, efficiency, and concurrency model. Because of these and other aspects, Go is a unique language that is worth exploring. However, the scope of the course goes well beyond Go itself as it covers various technologies and skills required for building production-ready applications, such as Application Programming Interface (API) design, database integration, containerization, infrastructure provisioning, automation of code integration and deployments, and telemetry instrumentation, among others.

Following this introduction, the thesis presents the Go programming language in chapter 2. It mentions some of its strong characteristics and describes its common use cases. Next, the work maps the ground and the current state of related courses at the Faculty of Informatics of Masaryk University in chapter 3. It delves into the strong and weak points of the related courses and provides an overview of the practical state of such courses. These observations are then further reflected in the overall design of the course in chapter 4. All of the decisions taken and the reasoning behind them are described throughout the chapter. Chapter 5 delves in-depth into the content of the course, with each topic introduced separately. It introduces all of the content of the course with descriptions and further explanations. Lastly, the chapter 6 summarizes the work and declares future plans for the course.

Unless otherwise specified, all of the work discussed throughout the thesis is my own. As the thesis builds on the foundation laid by the Red Hat Go Course[2], it clearly states the extent of usage of the original materials and the transformations made to them in chapter 5

where each course topic is inspected in depth. Whenever I use prular, I refer to the team behind the organization[1] of the Go course.

The text of this thesis is licensed under the Creative Commons Attribution 4.0 Attribution International [2] license. All of the course materials are further summarized in appendix A and are publicly accessible on GitHub[3] under the Creative Commons Attribution 4.0 Attribution-ShareAlike International [3] license unless otherwise specified.

1. Martin Ukrop, Pavel Tišnovský, Ivan Nečas, and Stanislav Zeman.
2. `https://creativecommons.org/licenses/by/4.0/`
3. `https://creativecommons.org/licenses/by-sa/4.0/`

2 Go programming language

This chapter introduces the Go programming language. It describes some of its characteristics and strong points in section 2.1 followed by an overview of its most common use cases in section 2.2.

Who else might introduce Go than Rob Pike, one of the original creators:

"Go started in September 2007 when Robert Griesemer, Ken Thompson, and I began discussing a new language to address the engineering challenges we and our colleagues at Google were facing in our daily work."

"When we first released Go to the public in November 2009, we didn't know if the language would be widely adopted or if it might influence future languages. Looking back from 2020, Go has succeeded in both ways: it is widely used both inside and outside Google, and its approaches to network concurrency and software engineering have had a noticeable effect on other languages and their tools."

"Go has turned out to have a much broader reach than we had ever expected. Its growth in the industry has been phenomenal, and it has powered many projects at Google." [4]

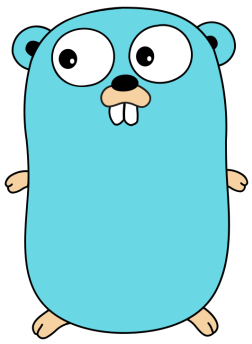

Figure 2.1: Gopher, the Go mascot.

2.1 Characteristics

Go is characterized as a high-level, statically typed, garbage-collected, and minimalist programming language as it prioritizes simplicity over features. It is mostly known for its concurrency design that is inspired by the Communicating Sequential Processes [5]. This paradigm favors passing messages over sharing memory for concurrent processes and is a fundamental principle in Go's design.

2.2 Usage

The usage of Go spans across numerous domains. The most notable are described in the following sections.

2.2.1 Cloud & Network Services

Go enjoys extensive usage in the cloud-native environment. The cloud environment uncovers concurrency problems that are easier to tackle with Go's concurrency system. To clearly illustrate, over seventy-five percent of the Cloud Native Computing Foundation (CNCF)[6] projects are implemented in Go. [7]

2.2.2 Command-line Interfaces (CLIs)

Go supports multiplatform builds with fast build times, which is crucial when designing CLI applications. Go is also statically built into a single binary, which further helps with portability as it does not require any runtime dependencies. [8]

2.2.3 Web Development

Go is one of the few programming languages that incorporates the implementation of an HTTP server directly in the standard library. Third-party libraries then leverage this implementation and further build upon this foundation.

Go itself is also a performant language. While not as fast as Rust[9] or C[10], it strikes a great balance of developer productivity and performance. [11, 12]

2.2.4 DevOps

Thanks to having great performance and build times as described in sections 2.2.2 and 2.2.3 respectively, it is also great for all types of automation processes, such as continuous delivery and deployment (CI/CD), often leveraged in Development Operations. [13]

3 Current state at FI MUNI

As this thesis focuses on designing a course in the context of the Faculty of Informatics Masaryk University (FI MUNI)[1], it first maps the ground and explores the current state of related local courses focused on providing practical knowledge while describing both their strong points and shortcomings and providing references. Some of these points are reflected throughout the design of the course, and comparisons with the new Go course are made in section 4.2.

It is worth noting that all of the information further described is my own observation as a last year bachelor's degree student studying the Programming and Application Development program (PVA)[2] and that the observations are mostly concerned with practical applications.

3.1 Up-to-date technologies and practical knowledge

A common academic mindset is that the faculty's goal is not to teach its students specific technologies but to provide fundamental knowledge of informatics. I agree that this is valued in the long run as technologies evolve and change, but the fundamentals remain the same. However, I would like to somewhat oppose this idea as I firmly believe that combining these two approaches results in students gaining solid fundamentals that they can apply or reflect on specific technologies that they can later leverage in practice.

The faculty currently maintains profession-oriented study programs, namely the PVA and Software Engineering (SWE)[3] (although some of the points mentioned, in my opinion, apply to this program as well, the chapter is mostly concerned with the PVA program as I can better reflect on it). Compared to more academic programs, these programs are intended to focus more on practical applications.

I argue that the present situation in these programs regarding this type of applied knowledge is rather lacking and that more stress

1. `https://www.fi.muni.cz/index.html.en`
2. `https://www.fi.muni.cz/admission/bc/programming-and-development.html.en`.
3. `https://www.fi.muni.cz/admission/mgr/software-engineering.html.en`.

should be given to practical usage and application of the acquired knowledge rather than solely sticking to the fundamentals, as the current state of such courses is insufficient.

Both of the mentioned study programs require attending an internship[4][5]. These internships are a great form of driving students into the commercial environment. However, the faculty should, at least to some extent, prepare the students for such endeavors and not expect this knowledge to be given to them during their internships, as this is often not the case.

To give a clear example, students are required to attend an internship without ever having to connect an application to a database or design and implement an API. The PB138: Web development course described in section 3.2.1 mostly covers this knowledge. Unfortunately, the PVA plan suggests enrolling in this course during the last semester; therefore, this knowledge is not covered before attending the internship. This seems to be inappropriate for a program that specifies application development in its name. Hence, students often have to settle for relatively limited and often underpaid internship positions even though they could go straight to working as fully-fledged juniors. It is worth noting that I do not want to degrade internship positions in any way. I am only noting that the faculty could provide students with more practical knowledge that is usually expected when applying for such positions.

The programs also do not set any expectations, introduce the aspects of the interview process, or provide any related soft skills. Although there is the course PV206: Communications and Soft Skills[6] dedicated to exploring soft skill and the PV296: The Job Market in IT[7] which was introduced last semester and covered some of these aspects like the creation of curriculum vitae (CV), or introduced characteristics of employment statuses, these courses are not well suited for these use cases. The PV206 course is rather limited in capacity

4. See `https://www.fi.muni.cz/studies/internship/programming.html.en` for PVA and `https://www.fi.muni.cz/studies/internship/software-engineering.html` for SWE.

5. Even though the courses themselves are called internships, any employment position is accepted. I.e., you do not need to be an intern.

6. `https://is.muni.cz/predmet/fi/jaro2024/PV206?lang=en`

7. `https://is.muni.cz/predmet/fi/jaro2024/PV296?lang=en`

and, judging by its syllabus as I have personally not attended this course, goes overly in-depth for this use case. The PV296 course is not currently planned to be taught yearly but could serve as a ground for covering such topics in the future.

3.2 Related faculty courses

This section describes courses that try to provide practical knowledge or somehow relate to the discussed topics. Some of the described courses are additionally mentioned and compared to the design of the new Go course in the section 4.2.

3.2.1 PB138: Web development and markup languages

The PB138: Web development and markup languages[8] is a course originally named Modern Markup Languages and Their Applications that has been reworked from a solely markup language course, which dedicated all of its time to mark-up technologies like Extensible Markup Language (XML) and HyperText Markup Language (HTML).[9]

Nowadays, the course includes programming and developing web applications using the Typescript programming language[14]. The scope of the subject is rather large, as it covers both frontend and backend development and has to cover all of the topics from designing a database scheme and implementing a persistence layer and API to creating single-page applications using React[15].

The course currently covers a rather vast amount of topics. One potential way to improve the course would be to split it up into two courses. The first one would concern the backend design, while the second would cover the client side of modern web applications. This would help make the course more maintainable and allow the newly created courses to go more in-depth.

With all this said, this is currently the only mandatory course for the PVA program that addresses most of the program's shortcomings and teaches the knowledge mostly required in practice.

8. `https://is.muni.cz/predmet/fi/jaro2024/PB138?lang=en`
9. The course was last run in this form in the Spring of 2019. See `https://is.muni.cz/course/fi/spring2019/PB138` for more information.

3.2.2 PV281: Programming in Rust

The PV281: Programming in Rust[10] course, besides teaching the Rust programming language itself, went beyond just the language as it taught topics like software architecture design, introduced API development and databases basics. From my personal experience, this course has been a great source of knowledge, especially for me as a second-year student, that I could mostly apply to practice. I attended the course in the Autumn of 2022 as a student and later joined in on its third run in the Autumn of 2023 as a seminar tutor.

The course certainly has its shortcomings. Docker[16], which was heavily used throughout its latter seminars, was not properly introduced even though it was never even mentioned in its prerequisites. Some Docker knowledge was, therefore, implicitly expected. To this day, only the PB176: Basic Code Quality and Management course described in section 3.2.3 delves into the topics of containerization with its practical usage. The course also tries to extend the topics covered to full-stack development thanks to lectures on HTMX[17] and Desktop development using the Tauri framework[18]. Both of these technologies are certainly interesting and should be mentioned. However, the course should consider exploring the Rust language in depth and the aspects of backend development rather than exploring these web technologies to such an extent.

Because of the scope of the course, it did not properly delve into the depth of the Rust programming language. One topic that is worth mentioning is macros, which the course lacked and are leveraged by the language quite heavily. Macros are only included as a part of a bonus lecture that at that time, did not occur. Therefore, it appeared to be an afterthought.

3.2.3 PB176: Basic Code Quality and Management

The PB176: Basic Code Quality and Management[11] course tries to cover topics implicitly expected by many courses but never previously cov-

10. `https://is.muni.cz/predmet/fi/podzim2023/PV281?lang=en`
11. `https://is.muni.cz/predmet/fi/jaro2024/PB176?lang=en`

ered. These topics include fundamentals of git, build tools, continuous integration (CI), and Docker[12].

Although this course delves into the problematics of Docker, it does so in a way that makes it voluntary. The course requires the students to finish three of the four homework assignments introduced throughout the semester. Unfortunately, Docker is explored in its final lectures and its last fourth homework assignment. Consequently, the Docker homework becomes optional if students have completed the previous three assignments and they can freely ignore it.

3.2.4 PA165: Enterprise Applications in Java

The PA165: Enterprise Applications in Java[13] is the last course in the trio of Java courses. As such, it primarily focuses on technologies rather than teaching Java fundamentals, which are mostly taught by the preceding courses. This course is intended to be attended by graduate students as no undergraduate programs suggest its attendance in their study plans.

Besides the Java Spring Framework[19], the course explores observability tools like Prometheus[20] or Grafana[21], authentication using OAuth2[22], continuous integration and delivery, and messaging while leveraging the Docker platform throughout the course.

All of these topics and technologies are usually greatly used in practice and, as such, should be introduced to students sooner in their studies. Additionally, to attend the course, students must pass both the previous Java courses.

3.3 Red Hat Go Course

During the Autumn semester of 2023, a new first trial run for the Go programming language course was planned under the PB173 Domain-specific Development course[14].

The course was organized by Martin Ukrop and led by Pavel Tišnovský and Ivan Nečas, both of whom maintain the existing Go

12. This information is mostly based on the course run of Spring 2023, which I had personally attended.
13. `https://is.muni.cz/predmet/fi/jaro2024/PA165?lang=en`
14. `https://is.muni.cz/predmet/fi/podzim2023/PB173?lang=en`

Course [2] under Red Hat. At that time, I had been studying Go for several months, and as I was deeply interested and dedicated to mastering it, I decided to join in on the efforts as a tutor.

The Red Hat Go Course was originally designed to cover the language fundamentals, and thanks to Go being a minimalistic language, teaching it only took half of the semester. The second half was lecture-free, and the students were left to implement their projects. A subsequent project defense was organized for the last week of the semester, during which students provided a description and a demonstration of the projects. The projects were also submitted for review, which we provided two weeks after the defense.

3.4 Extending the Red Hat Go Course

The course was short and limited in scope, so it felt like it missed several opportunities. To name a few, it did not cover any interesting technologies, API design basics, integration with databases, or instrumenting telemetry. All of these skills are required in practice when designing and implementing production applications. As a result, we have decided to expand the scope of the course with additional materials and rework the existing ones for the next run.

3.4.1 Questionnaire

After the successful end of the first run, we issued a questionnaire to gather student feedback for things that could be improved and incorporated into the new course. We managed to gather feedback from nine students out of the nineteen who finished the course. The course originally started with twenty attendees. Although this sample does not reflect the majority of opinions and is not necessarily representative, it still represents a significant portion of the course graduates and should be considered.

All questionnaire questions and answers referenced in the following subsections were translated from the Czech and Slovak languages to English using the ChatGPT[23] language model.

Additional exercises

One of the greatest shortcomings I suspected before handing out the questionnaire, and was often pointed out by the questionnaire participants, was a lack of programming exercises. The final project was the only output we required from the students, and as they were mostly implemented throughout the final weeks of the semester, students had no motivation to put in any extra effort throughout the first weeks.

The students unequivocally agreed that additional activities would be welcome. However, they generally opposed the idea of dedicated seminars as the Go language was too simple and preferred homework exercises. Some of the excerpts to the related question:

Q: *"Would you welcome dedicated seminars or additional assignments with feedback to accompany the lectures? If so, which of these two options would you prefer and why?"*

A1: *"For me, assignments are more important; seminars in programming subjects generally don't make much sense to me."*

A2: *"Definitely homework. I don't think seminars and working with a tutor are necessary. Homework can be solved anytime; they are not time-bound or tied to a specific time."*

A3: *"Seminars aren't very necessary; I understood everything well from the lectures and could immediately apply it. Homework might help with revision. Also, the project was practically a homework assignment."*

Sneakpeek to the new course outline

The students were presented with a course outline that had existed then. The outline's content was similar to the current one presented in fig. 4.1 with one major change; the older one included an additional Event-driven communication lecture. This lecture was later discarded as I deemed it less important than the Observability lecture described in section 5.16. The removed lecture is also mentioned in the section 4.6.3. In general, the students welcomed the showcased

changes:

Q: *"For the next run, we plan to significantly revamp the course content to cover the entire semester. Do you have anything you would like to add regarding the content above? What do you think is a great addition, what seems unnecessary, or do you have any suggestions for content to be included in the course?"*

A1: *"The second half sounds really great. Too bad it wasn't there this year; it's exactly what would greatly improve the course."*

A2: *"It looks great. I hope that the materials for the next run will be available for graduates of this course, too. I would love to learn something new."*

A3: *"The content looks great; it's good that there's room for more interesting stuff."*

However, some concerns were raised by the new material additions:

A4: *"For me, the question is whether topics 7-11*[15] *should belong to the Go programming subject or a separate half-course. What initial knowledge do you expect students to have in these topics? Each of these topics seems quite broad to me, so you would have to go superficially for students who haven't encountered the topic yet or assume some basic knowledge. Certainly, the initial questionnaire for students will help you with this, but I would expect that without prerequisites, there will always be students who are completely unfamiliar with the topic."*

A5: *"Will the conclusion still involve a project? Won't it be too much in conjunction with assignments?"*

These concerns are further reflected in the course design described in chapter 4.

15. This references the same lectures as described by the overview in fig. 4.1 with the addition of the discarded Event-driven programming lecture.

General feedback

Some of the additional general feedback provided on the course:

Q: *"Are there any concepts or specific aspects of Go that you feel you still don't fully understand even after completing the course?"*

A1: *"During project creation, I struggled with structuring the project and how to design it. Also, I found it challenging to create packages sensibly and avoid cyclic imports."*

A2: *"Perhaps goroutines, but I don't think we focused on them too much."*

A3: *"Perhaps delve more into packages."*

A4: *"Slices and arrays, although it might be because I didn't work with them much in my project."*

Q: *"Did you lack any knowledge during project implementation that the course didn't cover adequately?"*

A1: *"I didn't understand how to run Docker and how to use it."*

A2: *"Perhaps how to set up a repository and how to structure a larger project. It's nothing that couldn't be looked up, but it would be nice to have it included in the course."*

A3: *"Slices and arrays, although it might be because I didn't work with them much in my project."*

A4: *"If I had paid more attention (or if greater attention had been demanded from me, for example, through exercises or assignments) during the semester, I wouldn't have had to rush to catch up on everything during project work."*

These comments are also reflected throughout the design chapter 4.

3.4.2 Interest

The first run of the Red Hat Go Course was Go's first occurrence at our faculty. There is certainly a great interest in the Go programming language. I judge this by having personal conversations with peers and other tutors, the overall sixty-four positive reactions on the faculty Discord when this course was announced, and by the spike[16] in course enrollments before the semester that I perceived. However, it must be noted that the Domain-specific Development course under which the Go Course ran shared it with two additional seminars, so this fact may be inconclusive.

16. Over one hundred students enrolled for the course at one point.

4 Course design

This chapter gives high-level insight into the course design and the reasoning behind the decisions taken. Detailed explanations about particular lectures, exercises, and homework are given in the chapter 5.

4.1 Overview

The course primarily focuses on teaching the Go programming language. However, its goal is not only to teach the fundamentals of Go, but the course also focuses on equipping its students with the knowledge often required in practice to develop production applications.

The course builds upon the foundations of an existing Go Course maintained under Red Hat[2]. This Red Hat course consisted of multiple lectures covering the fundamentals of Go. These lectures were reworked, restructured, and incorporated into the new course.

The newly created course is intended to be taught throughout the span of twelve weeks. The first eleven weeks are regular lessons, and the last week is reserved for consultation, as a potential substitute class, or as a ground for inviting external guests working with Go.

The course is designed to be taught in an academic environment where lectures and seminars are usually one hundred minutes long. This is reflected in the expected lecture times and exercises.

All of the course materials are publicly accessible on GitHub. The respective links are provided in the appendix A. The GitHub platform was chosen as it is the most popular code hosting platform[24]. Additionally, it is a great opportunity to introduce it to students who do not regularly use it.

The course materials are structurally split into three major categories: lectures, exercises, and homework, each with a dedicated git repository. All of these categories are examined in the following subsections. The last subsection of this chapter briefly talks about the final projects.

The following fig. 4.1 showcases the course outline to better understand the schedule.

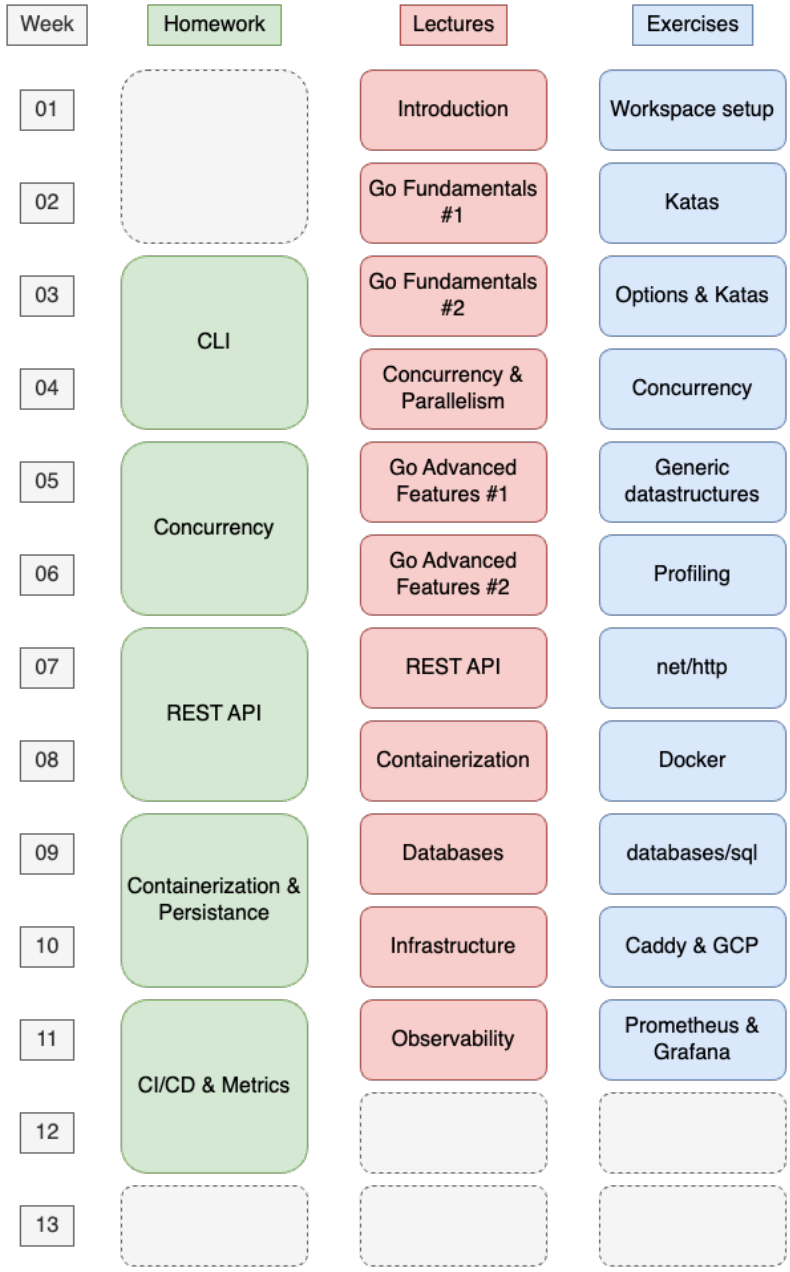

Figure 4.1: Outline of the course.

4.1.1 Lectures

The first six lessons are adaptations of the original Red Hat Go Course slides, which underwent major changes and restructuring. The changes were made to make the taught topics more cohesive and easier to follow, as some of the code samples were unnecessarily long or complex. Several lessons, specifically those on REST API and Concurrency, presented outdated principles that warranted updating. Multiple lectures, such as Math & Go or Python & Go, were removed altogether as I did not deem them important.

The last five lectures are completely new and extend the course with numerous technologies and third-party libraries. The original course focused solely on using the standard library. Some of the parts of the standard library, notably the default HTTP web router, are generally not leveraged in practice[1]. The new lectures focus more on practical usage; therefore, they are not limited in this way.

All of the lectures are estimated to take around eighty to ninety minutes. The lectures should be followed by starting the implementation of the exercises further described in section 4.1.2.

Tooling

I considered multiple tools for creating the lecture slides. For numerous reasons, I reviewed only markdown-oriented technologies. The major reason was that all materials were intended to be versioned using git, so a plain-text solution was necessary. This was done to ease collaboration with potential open-source contributors, which can further extend the course or fix any existing mistakes. The course also includes code exercises and examples that needed to be versioned, so merging all of these materials under a single GitHub organization seemed the most reasonable.

The original Red Hat Go course used the Go Present tool[25]. This solution, created by the Go team, implements an exceptional feature: it can run the Go code in your slides and immediately present the output. For this reason, I decided to continue using it for the new lectures, as any other markdown-styled solution did not offer any similar viable alternative.

1. Although this might change as it was extended in the Go version 1.22 release.

I also considered Marp[26] as the runner-up. This tool is used throughout the PV281: Rust course previously mentioned in the section 3.2.2. Compared to the Go Present tool, the main benefits include much better formatting capabilities, the possibility of exporting slides to multiple data formats like PDF or HTML, and a more straightforward syntax.

4.1.2 Exercises

The exercises are designed to be complementary assignments to lectures. The general feedback on the first course run, as discussed in the section 3.4.1, was a lack of hands-on experience. These exercises were designed to compensate for that. They are, by no means, an equivalent substitute for traditional standard seminars. The students are expected to spend only up to thirty minutes solving them.

Each lecture is paired with a single exercise that can, in some cases, be split into up to two tasks. Both of these, in some way, relate to the topics covered in the lecture.

The exercises are not intended to be graded. However, they can be reviewed or evaluated as they can provide useful on-site insight into the student's understanding of the covered topic, which can be reflected later. These exercises should also serve as the ground for answering potential follow-up questions on the lecture topics.

It is worth noting that the exercises are not necessarily intended to be completely finished. They are expected to take up around twenty to thirty minutes. That amount of time will usually not be left after giving the lecture. As noted previously, some exercises comprise multiple tasks; in such cases, you can prioritize which one you deem more important and instead do just the single task.

4.1.3 Homework

The homework assignments are a form of individual work. Their goal is to prepare students for their final projects. The first two homework assignments are independent, while the last three depend on each other and demonstrate building an application from the ground up. If students miss any of the first two of these three assignments, they will be given a sample solution to continue with the following homework.

The students start by forking the original homework assignment repository, making it private, and adding their tutors to the repository. The assignments are then submitted as a pull request into these private repositories, where they will be reviewed.

Starting from the third week, the assignments are published with a ten-day deadline followed by a four-day review period. The review period is designed to give students feedback that they will later incorporate into their solutions. The feedback does not need to be incorporated throughout the specified four days. These four days are set aside just for the tutors to review the submission. Students can then incorporate the feedback in the next seven days. This might invertible cause the homework assignments to overlap, which may cause problems, especially with the last three assignments that depend on each other. Nevertheless, as the assignments differ in scope, they can, to an extent, be worked on simultaneously. Students will then have to rebase their solution, which might result in some numerous merge conflicts. However, that is also one of the wonders of real-life software development.

4.1.4 Projects

The projects are the final part of the course. They are estimated to take fifty to sixty hours[2] for each student. The size of the projects should approximately match that estimation.

To initiate the projects, students are presented with a set of technologies or libraries they can leverage in their project implementations in a bonus lecture described thoroughly in section 5.18. This lecture subordinately serves as a form of motivation.

The general incentive is to make students come up with their own projects. Therefore, no "official" project assignments will be published. This was deeply considered as it may demotivate some students in the final part of the semester. However, I believe that the exploratory bonus lecture is sufficient for driving the students' projects even without predefined assignments.

2. This requirement will need to be decreased to twenty-five to thirty hours if the course is to be run under the Domain-specific Development course. This is done solely to match the course extent.

No matter the choice of project, all of them are finished with a short defense during the examination period. These defenses can be organized either physically or online. Students are expected to give a short presentation and a demonstration. Students should also cover used technologies and libraries and briefly describe their experiences and struggles.

There are two reasonable approaches on how to review the projects.

- The first one is to review the projects before the defense and confront the student during the defense. This allows confronting the students and having a discussion to reflect on the student's solutions and their potential mistakes. On the other hand, it slightly prohibits public defenses in the presence of other students.
- The second one is to review the projects after the defense. This permits the creation of public sessions that multiple students can attend at once, as no personal questions are raised. On the other hand, it makes the follow-up review less personal, and the student cannot be reasonably confronted.

There are multiple means of completing the project, each offering a different set of advantages and disadvantages. The course is designed not to be overly restrictive, therefore giving flexibility of choice for the projects.

The project assignment can be fulfilled in three ways:

- Implementing a team project.
- Implementing a solo project.
- Creating an open-source contribution.

Team projects

The team projects required coordination between the team members; therefore, it simulates practical collaboration the most. However, it restricts students as they have to agree on a single project assignment.

The students can group into up to four-member teams, but two or three-membered teams are preferred. The size of the projects should

be linearly scaled with the number of team members. I do not recommend exceeding these numbers, as it is generally observed that any higher number may negatively affect the teams' performance. It mostly only brings an additional overhead and introduces large project requirements.

When evaluating projects, it should be verified that the work was roughly evenly distributed between the students. As often happens, some students leave most of the work for their team members and only join in on minor tasks, if at all. Such lack of activity should be reflected in the student's evaluation.

It should also be noted that sometimes, a student implements most of the project assignments independently without giving the rest of the team a chance to contribute appropriately. Differentiation between the two cases is merely impossible. The only way is to consult the team members directly, and even then, the active students tend not to report on and abandon their peers.

Solo projects

The solo projects give the most flexibility. Students can implement their own assignments without consulting or cooperating with other team members. This, for example, allows them to implement a project they have long thought about but did not find enough time for its realization.

On the other hand, the lack of interaction with peers is also the biggest disadvantage. The goal of these projects should, at least to some extent, be to get students familiar with real-life software project collaboration. Solo developers hardly implement such projects in practice.

Based on the first course run, students opted for this option the most as it is perceived as the path of least resistance. Hence, it should be expected that most projects will be of this kind.

Open-source contributions

The open-source contributions are rather experimental. The main motivation behind this variant is to allow students to get involved in

the open-source community, give them a guiding hand, and provide feedback during their first steps.

This path presents several challenges to the students. First, the students must find open-source projects to contribute to. Secondly, they must engage, review current issues, and participate in the discussion and codebase. Lastly, they must implement the changes and get them reviewed and merged.

The pull or merge requests do not have to be merged at the time of the defense, as the tutors cannot guarantee the maintainers' activity. This can be tackled by providing projects maintained by tutors or their colleagues. However, such appropriate "good-first" projects that are simultaneously maintained by the tutors often hardly exist.

The contributions can be split up into multiple pull or merge requests, even to different projects. Therefore, opening a larger pull request and opening multiple smaller ones are both acceptable variants.

The size expectations for the open-source contributions are much lower than those for its project alternatives. Upon its evaluation, the process of searching for an open-source project to contribute to and further involvement in the codebase is considered.

It should be noted that contributing to projects with low activity or without active maintainers should be avoided.

Our faculty does not promote open-source collaboration enough. The faculty's lack of permanent open-source-oriented courses exploring open-source workflow and etiquette can further support this statement. Some efforts to introduce such courses have been made in collaboration with industrial partners in the past few years. These open-source projects also try to push this motion forward.

4.2 Comparing with existing courses

The overall course design was mostly inspired by the existing PV281: Rust course described in section 3.2.2 and additionally resembles[3] the PA165: Enterprise Java course described in section 3.2.4.

3. The Go course was mostly designed before I had the chance to attend the Java course.

Compared to the PV281: Rust course, the Go course does not cover any client-side development[4], and instead solely focuses on the back-end development. The Go course also covers containerization and the Docker platform described in section 5.12, which the Rust course did not cover and implicitly expected its knowledge.

The PV281: Rust course also inspires the overall teaching methods with some deviations. Instead of dedicated Rust seminars, the Go course has exercises. Instead of ten iterations[5], the Go course has five homework assignments. Lastly, the Go course offers more flexibility of choice as the projects do not have predefined topics as the Rust course dost[6] and do not require four-member teams like the Rust course.

Compared to the PA165: Enterprise Java course, the Java course covers OAuth2, microservices, and various methods of authentication and messaging, while the Go course covers infrastructure, containerization, and Docker in greater depth. Otherwise, the scope of the Go and Java courses is similar.

4.3 Reflecting feedback

The lectures now also explore the discussed topics in section 3.4.1 in greater detail. Some of them, like the Docker or the structuring of Go projects, were previously not covered at all.

The new outline also raised some concerns in section 3.4.1 regarding the scope of the course.

The first was that the scope of the course got rather large, with topics not directly related to Go. As was previously declared, the goal of the course is to equip its students with the knowledge required to build production Go applications. Although some of the topics do not necessarily relate to Go, namely Observability, Infrastructure, or Containers lectures, they are applied to the Go environment and demonstrate the process that is involved when implementing, deploying, and maintaining such production applications. For this reason, I decided that they were a good addition to the course's content. However, it is worth noting that these topics could be later extracted to a

4. This is covered by the HTMX and Tauri lecture in the Rust course.
5. Small homework tasks, one for each week.
6. Students can also create their own custom topics after consulting with tutors.

separate course as described in the section 4.6.4. Currently, there is no such opportunity to do this so for the time being, these topics will be covered in the Go course directly.

The second concern is that the course will become time-consuming with the addition of homework assignments combined with the projects. The reality is that the course's first run was overcredited as we did not require any attendance, and the only required output was the completion of the project. The parent course, Domain-driven Development, under which the Go course runs, currently has an extent of three European Credit Transfer and Accumulation System (ECTS) credits. Adding these additional homework assignments will align the extent of the course more with its parent course.

4.4 Evaluation

Students can acquire up to one hundred points in total. From these one hundred points, a minimum of seventy has to be acquired. The first half of these points can be received by completing the homework assignments, while the second half is obtained by completing the project.

There is a common requirement for all of the graded tasks. This requirement is that all Go code is written idiomatically, avoiding common mistakes and misconceptions.

4.4.1 Homework

The points for homework assignments are not distributed evenly as the two first homework are smaller in size and scope. Therefore, the first two homework assignments are graded by up to seven points each, while the last three are twelve points each. Each homework defines a set of requirements and expectations.

4.4.2 Projects

Defining precise grading criteria for projects is slightly problematic as they can vary heavily in scope and focus. However, some basic common criteria can be declared and are further described in this section. None of the following requirements apply to the open-source

contribution projects, as they do not make sense.

All projects should:

- follow the standard structure as specified by the de facto Go project layout standard[27].
- include a continuous integration pipeline that lints, formats, builds, and tests the projects and a continuous delivery pipeline that publishes a docker image to a public repository of any choice.
- implement minimal happy-path test cases using the Go testing framework.
- should include single or multiple scenarios that demonstrate its usage and that can be executed to verify basic functionality.

Bonus points can be gained by performing additional actions such as deploying the application, instrumenting the application with telemetry, setting up monitoring, creating an OpenAPI specification, heavily testing the application, etc.

4.4.3 Bonus points

Additional bonus points can be acquired by being active during lectures and exercises or submitting exceptional solutions for homework and projects. This rewards engaged students and promotes further activity during the lectures.

4.5 Prerequisites

The course is designed to be accessible to both master's and bachelor's degree students. However, it is certainly unsuitable for freshman-year students as a couple of prerequisites are expected throughout the course.

The students need to have programming fundamentals as they are not explored in-depth. The course instead gives an overview of things like control flow, etc. Students are also expected to be proficient in SQL and how databases work in general, as it is required for the database lecture described in section 5.13 and in the fourth homework described

in section 5.14. Lastly, fundamentals of networking and virtualization are also both required for Containerization and Infrastructure lectures described in sections 5.12 and 5.15 and their related exercises and homework.

4.6 Limitations

This section describes some possible inefficiencies that were later perceived but were currently not incorporated into the course. Some of these will be considered as additions to the course in the future.

4.6.1 Auth

The course does not cover authorization or authentication in any way. It is up to debate if the course should cover this topic and, if so, to what extent. I believe that education in this regard, especially with practical application, is insufficient at our faculty. A dedicated course tackling topics like JSON Web Tokens (JWT)[28], Cookies, and OAuth2[22] and their usage should exist instead. None of these concepts are currently covered in depth by any course.

4.6.2 APIs

The course currently covers designing and implementing REST APIs. This is still the standard for most service communication. However, during the past decade, multiple other approaches emerged that are often incorporated into systems in practice, notably gRPC[29] and GraphQL[30]. These were not added to the course but could be considered in the future as they are highly related to backend development.

4.6.3 Event-driven communication

Event-driven communication solves multiple problems imposed by today's world of distributed architectures. Giving a basic overview of how these systems work and how to integrate such systems should be considered in the future. Notable Go projects that exist regarding this topic are the NATS[31] message broker, the event-driven commutation library Watermill[32], and the messaging server Cetrinfugo[33].

4.6.4 DevOps/Ops overlap

Some of the lectures, namely Containerization, Infrastructure, and Observability (all covered in depth in sections 5.12, 5.15 and 5.16 respectively) overlap with the DevOps/Ops field. Although the lectures provide Go examples and try to apply the general knowledge to Go, these topics are not, in broad, Go specific and could be extracted to an additional course in the future. This would free up space for other topics, such as those previously mentioned. However, no such course currently exists.

4.6.5 Homework

No sample solutions for the homework assignments currently exist. However, they are planned to be implemented before the commencement of the semester to further verify the assignment's clarity and overall feasibility. Furthermore, the sample solutions must be implemented for students who miss one of the ReelGoofy assignments described in sections 5.11, 5.14 and 5.17. If such cases occur, they will be given the sample solution to continue with the following assignment.

Additionally, no automated test suites are currently prepared for the homework assignments. This quite limits the scalability of the course, as all homework submissions will have to be manually reviewed. It is currently not a big problem as we intend to open the course to only twenty students, but it should be considered if the number is to be increased. Some homework assignments, the second one described in section 5.8 the most, can also vary quite heavily in their solutions, and the tests could create boundaries. On the other hand, I personally believe that not giving students tests upfront motivates them to create their own suite of tests to verify their solution. This will have to be further discussed.

4.7 Licensing

The original course is open-sourced under the Creative Commons Attribution-ShareAlike (CC BY-SA 4.0 Deed) license[7], and as it is a

7. `https://creativecommons.org/licenses/by-sa/4.0/`

share-alike license, the new course had to respect it. There was an option to migrate to the GPLv3 license[8], which is compatible, but in the end, the Creative Commons license was preferred.

The original idea was to keep the new course open-sourced either way, so reusing and adapting the existing course was not a deterrent. The motivation behind this was to keep the course in the public space. Anyone who would like to remix or reuse the course would be free to do so. There is no reason to keep anyone from using any of its materials, even those with commercial intentions.

8. https://www.gnu.org/licenses/gpl-3.0.en.html

5 Course content

The content is described in the same order as it is presented to students throughout the semester.

The lecture sections start by briefly explaining the lecture's content, followed by the expected input knowledge before attending the lecture and the expected output knowledge gained following the lecture's attendance. A precise lecture outline is then given, followed by a more detailed description of the lecture's content. The first six lectures also contain a section describing the changes made to them as they are based on the Red Hat Go Course.

The homework sections briefly describe the homework assignments and their scope. However, these sections focus more on the assignments' requirements and motivation rather than reviewing the assignment details themselves. View the materials directly instead to learn more about the exact specifications of the assignments.

The descriptions of the materials may appear repetitive. It is done so that the content is described in a uniform manner, sharing the same structure, and is therefore easier to follow.

The attachment to this thesis includes all the materials and is further described in the appendix A with relative links.

5.1 Lecture 00: The course

This lecture is designed solely to provide elementary information about the course, such as its requirements and outline.

5.2 Lecture 01: Introduction

This lecture introduces the Go programming language and explores its current state and strong points. It also goes over the installation of Go and basic CLI usage and demonstrates Go basics using a Hello World example.

Following the lecture, students are expected to understand the current state of the language and be able to run Go code using the Go executable. No prior knowledge is expected.

Outline

- Introduction to Go (Origins, Characteristics & Comparisons, Motivation)
- IDEs & editors
- Installing Go
- Running Go (Go executable, Modules, Packages, Hello World)

Content

The first section explores the Go's history and the motivation behind its creation. It also compares it to other related languages and showcases its common use cases in practice.

The second section shows a list of recommended code editors and IDEs that support the Go programming language.

The third section guides the students through the installation of the Go toolkit. It covers both the original Go installation guide and commands for the most commonly used package managers.

The last section delves into the Go itself and describes the Go CLI and all of its commands. It introduces the concepts of packages and modules and shows basic project setup and commonly used layout standards. This section finishes off with running a simple Hello World application.

Changes

The first slides are a makeover of the original *go worth learning* lecture. The new lecture unifies the format, adds more examples, and removes some of the slides, mainly those with pictures, as they did not seem important.

The new lecture takes the first few slides[1] from the original *lesson1* and reverses their order. The additional slides are left for the second lecture described in section 5.3.

Some minor changes were made, like extending the list of characteristics and companies using Go and adding proper links to external materials.

1. Up to the slides about functions.

The completely new slides and topics in this lecture include the origin and history of Go, going through the installation process of Go, covering packages, imports, and modules much better and more in-depth, and introducing the Go executable and the standard project structure.

5.2.1 Exercise 01

The first exercise guides students through setting up a working environment on their machines. Students will install Go itself as well as set-up all relevant environmental variables. Finally, they will practice a basic usage of Go CLI through commonly used commands.

5.3 Lecture 02: Fundamentals #1

This lecture starts with exploring the core fundamentals of Go. It delves into topics like data types, functions, user-defined data types, and pointers.

After this lecture, students will be conscious of all the basic data types Go provides and all possible control flow keywords and their semantics. They will be able to write functions, define their own data types, and understand and appropriately use pointers throughout their code. No prior knowledge is expected.

Outline

- Types
- Variables
- Control flow
- Functions
- User-defined data types
- Pointers

Content

The first section starts with the core fundamental knowledge of basic data types and their default values. These include all numeric variants and their aliases, strings and their encodings, and booleans.

The second section describes all possible declarations of variables and their uses. It goes over their differences and implications and inspects the shadowing concepts.

All control flow variants are then introduced in the third section. It maps all of the keywords with example usage and semantics.

The fourth section introduces functions and declarations. It describes both named and anonymized functions, higher-order functions, or closures. It dedicates multiple slides to the defer keyword, which is extremely useful when freeing resources.

Custom structures and type aliasing are introduced in the fifth section. As Go itself provides no support for enumeration types, an example workaround is provided as it is commonly used in Go. Concepts like anonymous structures and structure embedding are also introduced.

The last section is dedicated to pointers. It shows their basic usage, like referencing and dereferencing, and their interoperation with editable structures as method receivers.

Changes

This lecture continues where the previous one ended. All of the slides from *lesson1*, starting from the slide about functions, were taken and incorporated into this lecture. Continuing with the *lesson2*, all slides until and including those covering structures were taken and appended to the new lecture. The section about points was also extracted and put behind the structures directly in the new lecture. Additionally, slides from the *lesson3* regarding topics like closures, lambda functions, and methods were also integrated into this lecture. Some examples were slightly changed to simplify them. Some examples felt too verbose or necessary, so they were discarded.

The new lecture now includes additional topics previously not covered, like enums, shadowing, struct embedding, or the infamous

init function, to name a few. The slides now additionally explicitly go over topics like high-order functions.

5.3.1 Exercise 02

This exercise lets students solve multiple katas that show commonly made mistakes. Some of the see katas are inspired by the mistakes covered in the book 100 Go Mistakes and How to Avoid Them by Teiva Harsanyi[34]. These katas all cover topics discussed throughout the lecture.

5.4 Lecture 03: Fundamentals #2

This lecture continues with the introduction to the fundamentals of Go. This one covers more interesting topics like errors, interfaces, and collections than the previous lecture.

Following this lecture, students will be able to understand the basics of error handling in Go. They will also be able to describe how interfaces are implemented, and they will be able to write idiomatic code using and iterating over the two common collection types, slices, and maps.

Outline

- Interfaces
- Errors
- Arrays
- Slices
- Maps
- Range

Content

The lectures start off with interfaces and their underlying implementations in Go. It also depicts some of the characteristics like implicit

interface satisfaction instead of explicit implementation or introduces the empty interface.

The second section follows up on interfaces with the introduction of errors. The section is dedicated to handling error values and their creation, either as in-lined, sentinel, or completely custom errors with additional metadata.

Arrays are introduced in the third section. These are mostly used just as a building block for slices which are explored next.

Slices are introduced next as the dynamic list type in the fourth section. The section further describes the slice's implementation and its implications.

The fifth section inspects the hash-map data structure native to Go. It goes over all of the common operations that can be performed on the map data type and also showcases how it can be used to implement a set.

The last section covers the range of keywords commonly used to iterate over collection items.

Changes

This lecture dramatically changes the order in which topics are covered. Basically, it takes all of the left slides in the original lectures called *lesson2* and *lesson3* and mixes them up. The lecture now covers topics like interfaces and errors, followed by collections like arrays, slices, and maps. Some examples, again, felt too verbose[2].

The new lecture now covers the topic of errors in much more depth, giving examples of creating custom errors, wrapping them, or using the errors package. It also adds an example of the new iteration over numbers range syntax.

5.4.1 Exercise 03

This exercise is split into two tasks:

- This first task, like in the previous exercise, lets students also solve multiple katas that show commonly made mistakes. Sim-

2. One concrete example would be the slides called or necessary, so they were discarded*More implementations of one interface* which spanned five slides without really adding anything to the table.

ilarly to the previous exercise, some take inspiration from the book 100 Go Mistakes and How to Avoid Them[34].

- The second task explores the options builder pattern commonly used in Go. This pattern is an alternative to the traditional builder pattern and is more idiomatic as it is less verbose and provides more flexibility.

5.5 Homework 01: EGG

The first homework lets students implement *Even Greater Grep* (EGG) grep-like command line tool. The assignment itself is simple; it specifies all possible variants of execution and arguments for the CLI application with expected behavior and output.

The motivation for this exercise is to let students start with a simple application that they most likely already know. As a result, they can focus solely on the implementation and do not need to solve complex algorithms or tackle business logic. The assignment itself also references some of the packages they will need to use without diving into their content. Hence, students will have to explore these packages using the documentation, which lets them get familiar with the Go packages[35] website and its format.

The submitted solution needs to cover all of the described usages and properly implement the expected behavior. The solution must also be written idiomatically and properly handle resources and errors.

5.6 Lecture 04: Concurrency & Parallelism

The lecture explores Go's concurrency model and dives into all of the synchronization primitives and principles commonly used in idiomatic Go code when building concurrent applications.

After attending this lecture, students will be able to understand the concept of goroutines and Go's runtime model. They will also be able to write concurrent applications leveraging all of the possible synchronization primitives. It is suggested that the students have some conception of concurrency and knowledge of synchronization primitives.

Outline

- Goroutines
- Runtime
- Channels
- Select
- Packages

Content

The lecture starts with exploring the differences between concurrency and parallelism and is followed by an introduction to the concept of goroutines. This knowledge is further built on in the second section, where Go's runtime model is described. One such state of the runtime environment can be viewed in fig. 5.1.

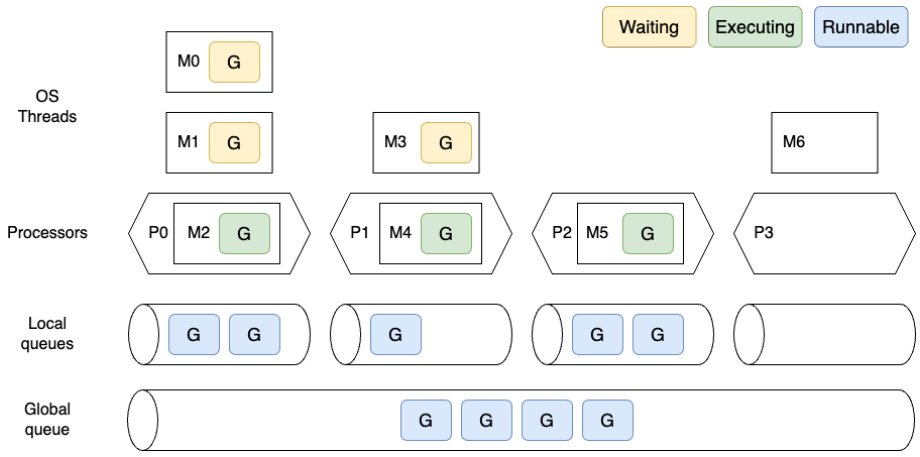

Figure 5.1: Possible runtime state.

The third section introduces the concept of channels as a means of communication for concurrent processes. Multiple diagrams were created to better showcase some of the most common channel interactions and edge cases, such as the one in fig. 5.2. The channels section also discussed various patterns commonly used in concurrent systems like fan-in, fan-out, or pipeline.

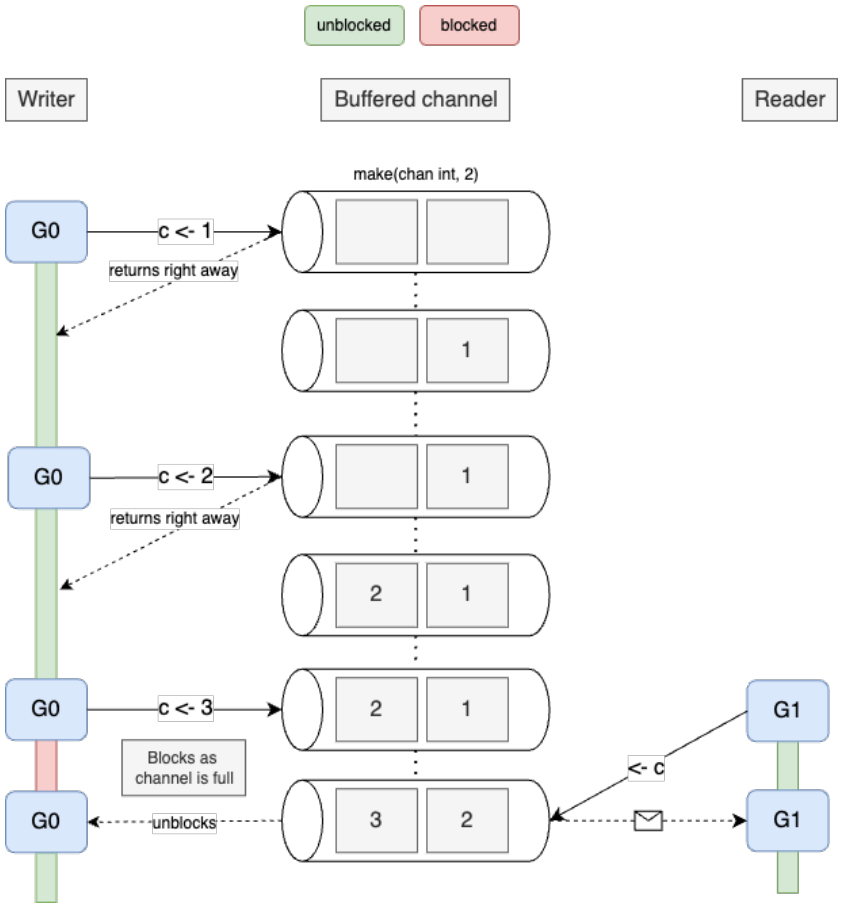

Figure 5.2: Two goroutines interacting using a buffered channel.

The last section mostly covers the sync and atomic packages that provide synchronization primitives, such as mutexes, conditional variables or wait groups, and atomic data types.

Changes

The original concurrency lecture was in a horrific state. It covered the whole topic of concurrency in just nine-teen slides. Moreover, some of the examples showcased out-of-date principles as they were around five years old. Ultimately, I believe I did not reuse any of these slides. However, some examples might share similarities.

5.6.1 Exercise 04

In this exercise, students are guided through implementing a producer-consumer problem in a service modeling invoice data messaging. The students will need to leverage the knowledge of all of the previously discussed topics like goroutines, contexts, and synchronization primitives. They are then further guided to optimize the application and implement a graceful shutdown for the service.

5.7 Lecture 05: Advanced #1

This lecture starts the exploration of advanced Go topics. The lecture begins with generics that were added to Go in version 1.18, followed by an overview of the majority of the commonly used standard library packages and finishes with the Go built-in testing framework.

Students will be able to implement all kinds of generic data types, test their applications with various types of tests, and use all of the described tools and packages. No prior knowledge is expected.

Outline

- Generics
- Packages
- Testing

Content

The first section explores the history of generics in Go and how generic data types were implemented before their addition to the language. It also explores some of their limitations and showcases possible use cases with the implementations of the map and filter functions.

In the second section, the lecture covers all interesting standard library packages that are often used in practice. These mostly include libraries for manipulating collections like maps and slices, libraries for interacting with the operating system and logging libraries.

Lastly, the lecture describes the different types of tests and testing models as shown in fig. 5.3 and further continues to explore Go's

testing toolchain. Apart from standard testing itself, it also covers fuzzing, which the testing framework supports. Finally, the section is finished with an overview of third party testing libraries that provide more high-level tooling or allow the generating of various types of fakes like stubs of mocks.

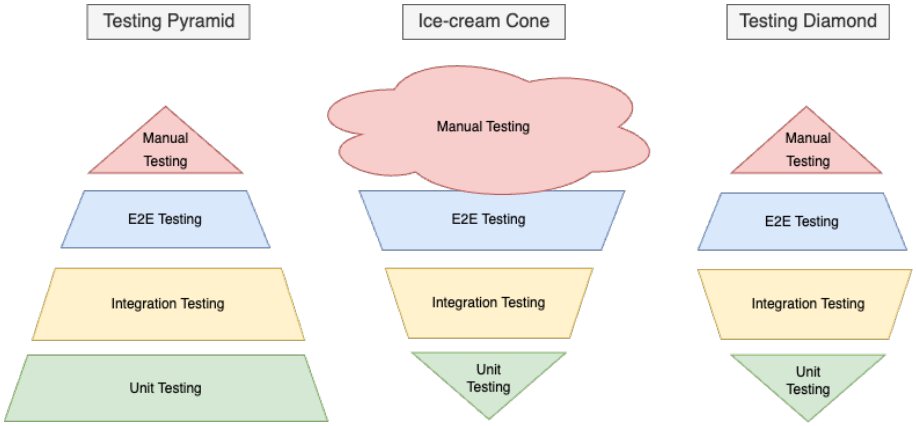

Figure 5.3: Testing models.

Changes

This lecture combines the original *testing*, *lesson5*, and *lesson8* lectures. Slides up to the go-carpet tool slides and slides regarding Gherking, GoConvey, and expect tools from the *testing* lecture are reused. From *lesson5*, the slides up to CGo were extracted. Lastly, the whole *lesson8* was reused, but some slides were removed, namely the generic list implementation slides or the Java and Python assembly examples.

Newly, the generics include slides about map and slice utility functions. After generics, the slides go over the most notable standard library packages; this was previously not covered at all. Finally, the lecture covers the topic of testing. Regarding this topic, the lecture goes much more in-depth into the standard library, thoroughly describing the T testing structure and giving examples of executing the tests while using additional arguments via the CLI, or it newly covers fuzzing and the F type.

5.7.1 Exercise 05

This exercise lets students implement a generic data structure of their choice, followed by the creation of multiple unit tests using the Go test suite. It also guides the student through creating a coverage profile and how it can be inspected using the go cover tool.

5.8 Homework 02: Barbershop

The second homework lets students implement a solution to the Sleeping barber problem described by Edsger W. Dijkstra in his Cooperating sequential processes publication[36]. The problem is described as a set of rules with a diagram showcased in fig. 5.4 to better visualize the problem.

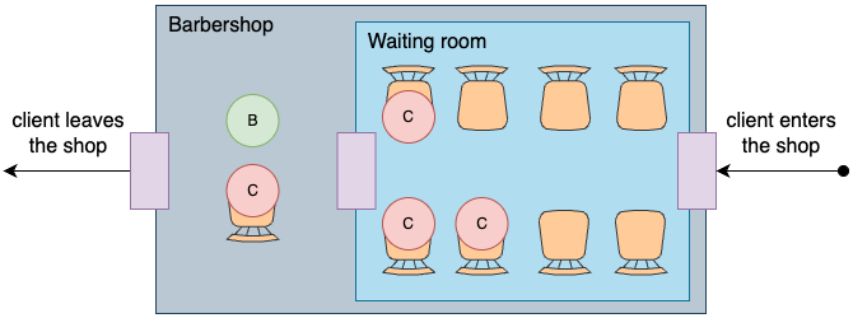

Figure 5.4: A visualization of the Sleeping barber problem.

The problem presents multiple synchronization challenges that students need to tackle. The motivation is to showcase the effectiveness of the concurrency model in Go, as the assignment can be implemented by only using channels and the select statement. No other synchronization primitives are required. However, the assignment intentionally references the sync package[37] that contains these primitives but hints that it may now be required to implement the solution. This should prompt the students to think through the means they will use to implement the solution. The problem is not hard to understand and, as such, is a nice way to introduce the concurrency model of Go.

The submitted solution needs to implement the problem as described by the set of rules. The solution must be race-free without the possibility of resulting in a deadlock.

Additionally, students can get two bonus points for implementing a solution that supports multiple barbers working in the barbershop.

5.9 Lecture 06: Advanced #2

This lecture introduces some of the additional advanced topics present in Go. It mainly focuses on performance and optimizations but also covers some of the additional topics like CGo, Unsafe, or Reflection, which could not be included in any other lecture as it either did not make sense or there was simply no space left.

Students will be able to understand common limitations regarding various data types and references and will be able to profile and optimize Go applications after completing this lecture. No prior knowledge is expected.

Outline

- Benchmarks
- Profiling
- Optimizations (References & Values, Maps, Slices, Loops, Strings)
- CGo
- Unsafe
- Reflection

Content

The lecture starts with introducing the benchmarking tooling in Go. This topic is similar to the testing presented as the last topic in the previous lecture. This was intentionally done to connect these two lectures, giving them a more natural transition.

Profiling go applications is then explored in the second section. The pprof profiling tool is introduced and a description is given about the various types of profiles and all methods for retrieving them.

The third section goes over optimizations commonly done in go and some common mistakes with specific data types that often cause low performance or high memory consumption.

The fourth section introduces CGo as a Foreign Function Interface (FFI) to the C programming language, which Go directly supports in the standard library.

The last two sections cover two standard library packages that leverage reflection, which allows one to examine types at runtime, or the unsafe package, which works around the Go's type system rules and allows such things as manipulating pointers directly.

Changes

This lecture combines the original *go performance*, *benchmarks*, *lesson5*, and *lesson9* lectures. Most notably, all of the benchmarks and slides covering the most commonly encountered performance errors were reused. However, the benchmark and all of their result were rerun with a newer version of Go. The version of Go and the machine used is now directly stated in the slides.

The new additions to this lecture include the reflect and unsafe packages that were previously not covered. Like with the testing and fuzzing type, it newly also explored the benchmarking B type in-depth. Also, a much better and more thorough introduction is given to the pprof profiling and the various types of performance profiles Go supports.

5.9.1 Exercise 06

This exercise presents students with a webserver that endlessly consumes memory resources. The students then must use the pprof profiling tool to examine the application's performance. They must generate the performance profiles using the exposed HTTP endpoints and visualize them using the appropriate graphs.

5.10 Lecture 07: REST APIs

Starting from this lecture, all of the materials are completely new additions to the course. It is worth noting that the previous course

did cover HTTP routing in one of its lectures, but it was outdated, like in the case of the fourth lecture described in section 5.6, and the new course does not build on top of those materials.

This lecture is dedicated to REST API design and to implementing such APIs in Go. The lecture covers the common fundamentals required to implement such APIs while also covering Go specifics and related technologies.

Following this lecture, students will be able to design and implement REST APIs using various Hypertext Transfer Protocol (HTTP) frameworks and routers. Only the fundamental knowledge of networking is expected.

Outline

- JSON
- HTTP
- REST API
- Go & HTTP (Standard library, Third-party libraries, Testing webservers)
- OpenAPI
- Templating

Content

In the first three sections, the lecture briefly goes over the fundamentals like JavaScript Object Notation (JSON), which represents a commonly used data format for data interchange, the HTTP protocol used for transmitting the data, and REST API principles.

The next section is dedicated to HTTP and its specifics in Go. It provides an overview of the most up-to-date and used routers and frameworks, highlighting both the advantages and disadvantages. A short part is also dedicated to testing HTTP servers and clients using Go's standard library packages.

The OpenAPI specification is described as the goto schema definition standard for REST APIs in the second to last section. Multiple

libraries that use the specification to create mocks or generate server or client code are also mentioned.

The last section explores the Go's built-in templating engine and showcases its usage. It is explored in this lecture as it can often be used to serve dynamic data over the REST API.

5.10.1 Exercise 07

This exercise starts with implementing an application that is further used in exercises described in sections 5.12.1, 5.13.1 and 5.16.1. The application is a simple ToDo Create-Read-Update-Remove (CRUD) app with no complex business logic. The OpenAPI[38] specification describes the application's API that the students will implement in this exercise.

5.11 Homework 03: ReelGoofy REST API & Testing

The third homework introduces the ReelGoofy application that students will implement throughout this and the next two following homework. The ReelGoofy application is a movie recommendation service. This homework aims to implement a REST API to expose four endpoints for ingesting data and recommending content. The API specification is given in the form of an OpenAPI specification.

The motivation behind the ReelGoofy application as a whole is to simulate building a realistic application from scratch. This first ReelGoofy homework gives students a lot of flexibility. They can choose a router or a web framework themselves. This gives them an opportunity to explore the Go ecosystem. After they choose a library, they are required to provide reasons for their choice in the form of a small text directly in the repository. This should prompt the students to think through their decisions as developers and the potential benefits or impacts of such choices. The implementation of the recommendation algorithm is, again, completely up to the students. This represents a form of a creative process. The students are also forced to study the OpenAPI specification that is often used in practice to document REST APIs.

The submitted solution must comply with the OpenAPI specification and implement the recommendation behavior. No complex algorithms are expected. The choice of the routing technology must be documented.

Additionally, students can gain up to five bonus points for implementing a simple test suite for the API. The tests were originally intended to be a part of the core assignment. However, even though the API was kept minimal, implementing it will most likely take a significant amount of time, so as a result, the tests were added as a bonus task.

5.12 Lecture 08: Containers

This lecture introduces containerization technology with a focus on practical usage. This knowledge is further required in the following lectures presented in sections 5.13, 5.15 and 5.16 and as it is currently not covered to the required extent as presented in section 3.2.3, it was incorporated into the course itself.

After the lecture, students will be able to containerize their applications and run their applications and their dependencies using container images. They are only expected to have a basic understanding of virtualization before the lecture.

Outline

- Containers
- Docker (Desktop, Hub, Build, Compose)
- Podman
- Testcontainers
- Kubernetes

Content

The lecture starts by introducing containerization as a concept while comparing it to virtual machines. This section also provides motivation

for using containers and briefly covers the Open Container Initiative (OCI) standards.

The second and most extensive section explores the Docker[16] platform in-depth. It provides information regarding its history, flows as depicted in fig. 5.5, architecture, and overviews related tools. Most of the time is spent on using the CLI, Dockerfile, and Compose tools. Lastly, a Podman[39] is mentioned in the third section as an open-source alternative to Docker.

The fourth section covers Testcontainers[40] as a tool that simplifies the testing process by running dependencies directly in code.

Lastly, the lecture briefly inspects the Kubernetes[41] platform that further builds on the containerization technology and solves numerous problems that are often faced when building distributed systems, such as horizontal scaling and fault tolerance. This section is not meant to provide thorough insight but to give a broad overview of how Kubernetes works and what it can be used for.

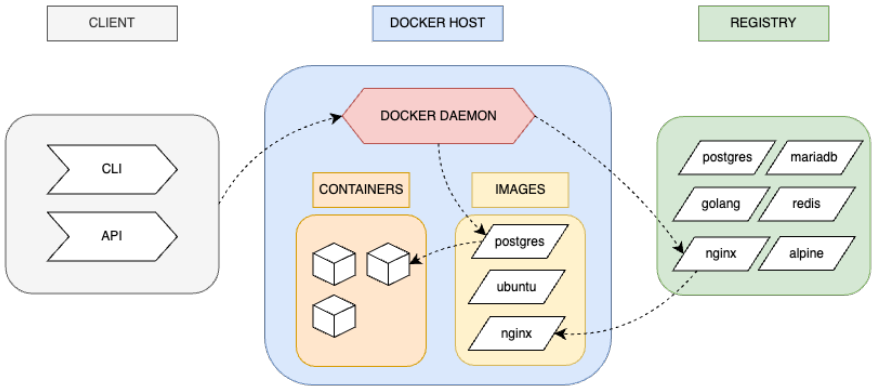

Figure 5.5: The Docker flow.

5.12.1 Exercise 08

During the exercise, students will containerize the ToDo application they started with in the previous exercise in section 5.10.1. The exercise is split into two steps. First, the students will create a custom container image using the Dockerfile, which they will then use in Docker Compose.

5.13 Lecture 09: Databases

This lecture focuses on Relational Database Management Systems (RDBMSs) and their integration with Go applications. The main goal of this lecture is to introduce the various approaches currently viable in the Go ecosystem, from low-level libraries to Object-relational Mapping (ORMs).

As the output knowledge, students will be capable of implementing persistent Go applications while levering RDBMSs. Students are only expected to be proficient in using the Structured Query Language (SQL).

It must be noted that the Red Hat Go Course also contains a lecture on databases. I did this lecture during the semester when we tutored the course. It mostly only shows excerpts from documentation of specific tools, some of these excepts are located in this lecture as well.

Outline

- SQL
- RDBMSs
- Database migrations
- Go & SQL (sql, sqlx, sqlc, GORM)

Content

The first two sections of the lecture give a brief refresher on SQL and RDBMS. It briefly overviews the most popular RDBMSs and their key points.

The third section introduces database migrations and their various aspects and challenges. It covers multiple database migration tools and showcases usage via both CLI and in-code.

Lastly, the fourth and biggest section discusses multiple approaches to integrating database access to applications leveraging modern Go libraries. It starts by covering the standard library's support and introduces compatible database drivers. As the standard library defines rather low-level interfaces, multiple libraries try to compensate for this; the most notable is sqlx[42], which is covered next. It provides more

high-level functions that make writing the code less verbose. Thirdly, the sqlc[43] library is showcased as an interesting and untraditional approach as it works by "compiling" the SQL queries and generating native Go code. The last presented approach is the GORM[44] library, which represents Go's most commonly used ORM.

5.13.1 Exercise 09

The exercise builds upon the ToDo application started in the previous exercise presented in section 5.10.1. The application in the assignment currently uses an in-memory repository. The exercise aims to extend the existing Docker compose file with a PostgreSQL[45] database and rewrite the repository to persist its data using the database instance.

5.14 Homework 04: ReelGoofy Containarization & Persistence

The fourth homework builds upon the ReelGoofy API implemented in the previous assignment. This assignment aims to containerize the application using Docker and implement a persistence layer using RDBMS and a library of choice.

The motivation is to futher simulate building a realistic application. This second ReelGoofy homework also gives students plenty of flexibility. Instead of a routing technology, they will now choose a library for interacting with the database system. This gives them an additional opportunity to explore the Go ecosystem. The choice will again have to be documented in the specified file. The choice of the relational database system is also given. However, PostgreSQL is recommended directly in the assignment, so it should not be expected that the students will deviate from this option.

The submitted solution must be runnable using the created Docker Compose file and must persist all of its data. Propper handling of the database connections and errors is also expected.

5.15 Lecture 10: Infrastructure

The motivation behind this lecture is to give students an insight into the continuous integration and continuous deployment/delivery (CI/CD) process and how this practice can be applied in the context of Go projects. The second goal is to give students a basic knowledge of how such projects can be deployed and maintained.

After attending the lecture, students will be able to set up a basic CI/CD pipeline, provision infrastructure using the Google Cloud Platform, and deploy simple applications. The students are expected to understand basic software engineering practices and elementary networking.

Outline

- Linters
- CI/CD (GitLab CI/CD, GitHub Actions)
- Infrastructure
- Cloud (Google Cloud Platform)

Content

The first section covers linters and expands on the knowledge acquired in the introductory lesson, where the built-in go-vet linter was introduced. Most of this section is dedicated to the golangci-lint tool, a linter aggregator incorporated in many projects and is the most complex tool today. Together with formatters, these tools are necessary in the continuous integration process introduced in the following section.

The second section introduces the CI/CD process as depicted in the fig. 5.6. The primary goal of this section is to provide students with the knowledge to implement the CI/CD pipeline using both GitLab CI/CD[46] and GitHub Actions[47].

As the faculty operates a self-hosted GitLab instance, almost all of the git-related work is done using GitLab. Consequently, most students do not interact with GitHub throughout their studies. The secondary goal of this chapter is to familiarize students with GitHub so they

can join the open-source community, as most open-source software is developed on the GitHub platform.

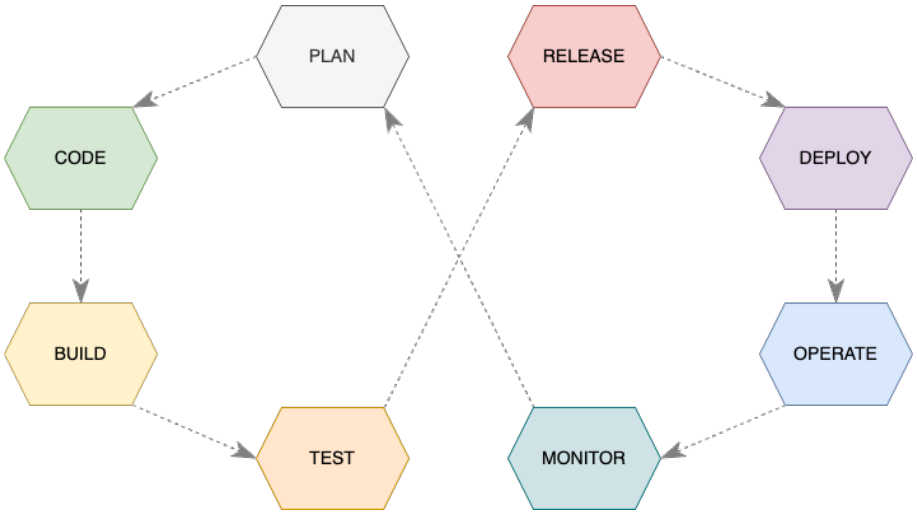

Figure 5.6: CI/CD pipeline.

The third section gives an introduction to infrastructure design. Introduces basic concepts such as proxying, load balancing, and application scaling and goes through multiple examples, one of which is depicted in the fig. 5.7. Two related technologies are introduced to leverage these concepts, namely Traefik Proxy[48] and Caddy[49]. This section aims to give students a basic understanding of how to provision basic infrastructure and tackle basic infrastructure design problems such as fault tolerance, scaling, or encryption.

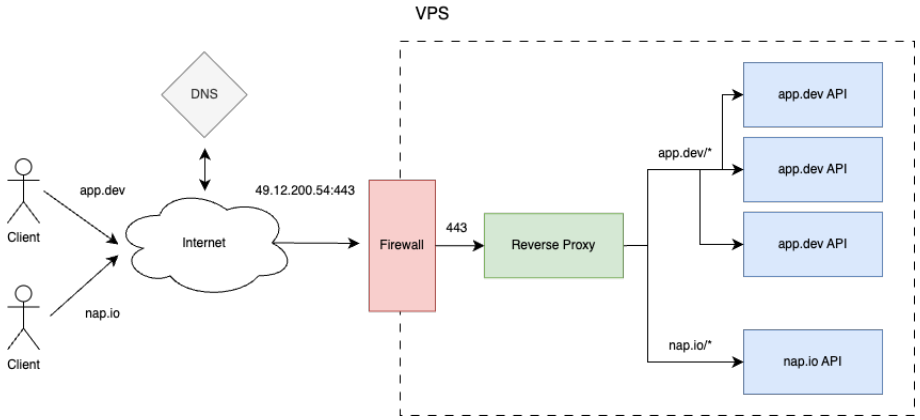

Figure 5.7: One of the infrastructure example diagrams.

The last section introduces all major cloud providers (Amazon, Microsoft, and Google) and further delves into Google Cloud Platform (GCP)[50]. As Google offers free credits thanks to the faculty program[51], it is a great opportunity for students to get their hands on operating cloud infrastructure. A set of basic services that GCP offers are introduced so students can leverage these in their final projects.

5.15.1 Exercise 10

The exercise comprises two tasks. The second task is more of a demo in which the tutor will guide the students through the GCP Console.

As students generally have a basic understanding of CI, thanks to their previous experiences, it is not practiced at the seminar. Instead, the seminar time is given to the GCP. The goal of the demo/exercise is to get students familiar with the GCP so they can provision their own infrastructure.

Tasks:

- The first task involves setting up a Caddy server instance as a reverse proxy/load balancing multiple ping-pong service instances. You can learn more about the ping-ping application in the appendix A.
- The second task involves setting up an automatic build and deployment using Cloud Run[52], provisioning a Compute

Engine[53] virtual machine, and a Cloud SQL SQL[54] server that can later be used for the student's projects.

5.16 Lecture 11: Observability

This lecture delves into the topic of observability[3]. The lecture covers all major types of telemetry signals, namely metrics, logs, and traces. Each signal is briefly introduced, after which specific state-of-the-art[4] technologies are introduced. In the last part, the OpenTelemetry[55] project is introduced.

Following the lecture, students should be able to set up the introduced technologies, instrument their applications with basic telemetry, and monitor their applications' health. No previous knowledge regarding observability is required.

Outline

- Health
- Metrics (Prometheus, Grafana)
- Logs (Loggers, Loki)
- Traces (Jaeger)
- OpenTelemetry

Content

The lecture starts off with an introduction to concepts such as health and heartbeat and shows how to export application status.

3. Even though it is an important topic that should be covered more throughout the studies, I personally encountered it only in the PA165: Enterprise Java course described in section 3.2.4. Moreover, it must be again noted that PA165 is a course designed for master's degree students, and bachelor's degree students do not generally attend it.
4. Many alternative technologies to the ones provided exist. The selection was made based on a general prevalence and on the fact that they are open-source and implemented in Go.

The next section covers the first of the three major signal - metrics. A brief explanation with multiple metric examples is given. The spotlight is then transferred to the Prometheus project.

Prometheus[20] is undeniably the leading project that revolves around retrieving, storing, and evaluating metrics. Apart from Prometheus itself, multiple third-party tools support the Prometheus format. The section introduces the format and the various metric types it supports.

Lastly, this section introduces Grafana[21] as the project of choice for visualizing metrics. The slides refer to the Granafa sandbox[5] that can be explored during the lecture shortly.

The second signal to cover is logs. This section puts logs into context and introduces additional structured logging libraries that offer additional features and focus more on performance compared to standard library packages. The technology of choice for logs is Loki[56]. Created by the Grafana Labs, it is a log aggregation system heavily inspired by Prometheus. The lecture covers the basic setup and summarizes possible clients and agents that can be used to retrieve the logs.

The final presented signal is traces. As traces are the most complex of all the mentioned telemetry types, a more thorough introduction is given. Then, a transition to Jager[57] is made.

Like with Loki, the slides introduce Jaeger architecture and core concepts. As Jaeger dropped support for all of its previous clients and now only supports OpenTelemetry API and SDK, the instrumentation part is left out and shown in the following section, where OpenTelemetry is explored in-depth.

The OpenTelemetry[55] project is an initiative that has gained much traction in recent years. Backed by the Cloud Native Computing Foundation (CNCF)[6], it is currently the leading project in defining telemetry standards. As of today, all major cloud providers support it. The last section gives insight into the OpenTelemetry Collector and its architecture showcased in fig. 5.8, explains how OpenTelemetry works, and how applications can be instrumented using its Go API and SDK.

5. `https://play.grafana.org/d/000000012/grafana-play-home`

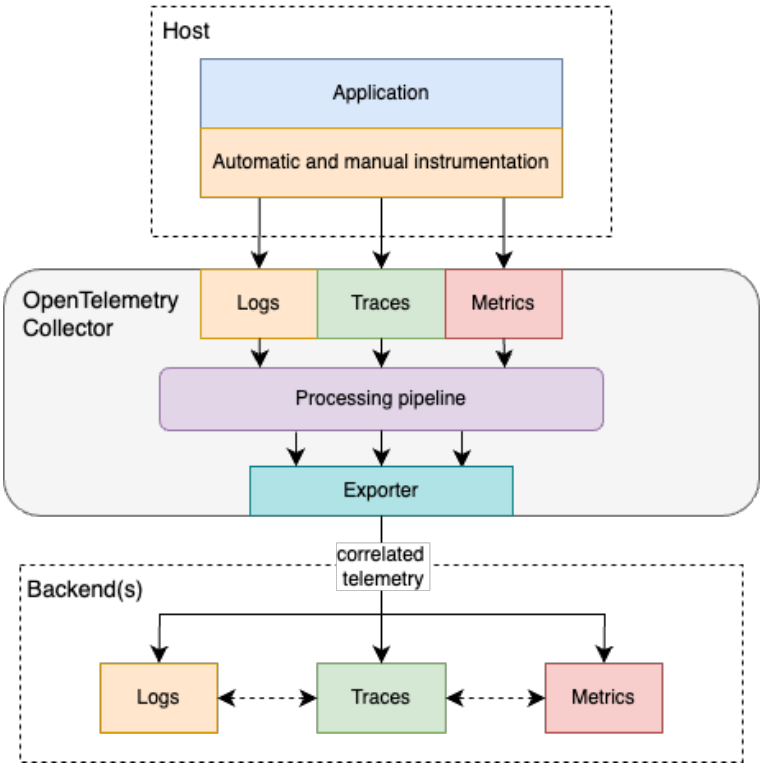

Figure 5.8: OpenTelemetry architecture inspired by the specification diagrams[58].

5.16.1 Exercise 11

Similarly to the previous exercise, this one comprises two tasks. The second task is also more of a demo where the students will explore the OpenTelemetry Demo[59] with the help of their tutors.

As I deemed metrics the most important to cover, the first task was dedicated to instrumenting and visualizing them. Logs are not really interesting to cover, and traces are rather complex to implement in a limited time span. Instead of trying to instrument an application with all of the related telemetry types, the exercise later moves on to the OpenTelemtry Demo in the second task, which has all of the mentioned things already implemented and better showcases real usage.

Tasks:

- The first task continues with the ToDo application that students previously worked on during the exercises described in sections 5.10.1 and 5.13.1. Their objective is to instrument the application with Prometheus metrics and create Grafana dashboards to visualize them.
- The second task involves running the OpenTelemetry Demo and interacting with the system. The system consists of multiple microservices implemented in multiple programming languages and instrumented with various types of telemetry while using the OpenTelemetry SDKs. This demo serves as a nice introduction to the OpenTelemetry ecosystem and also includes all of the presented technologies except for Loki, which is replaced with OpenSearch[60].

5.17 Homework 05: ReelGoofy CI/CD & Metrics

The fifth and final homework finishes the ReelGoofy service implemented throughout the two previous assignments. This assignment aims to create a CI/CD pipeline using GitHub Actions, instrument the ReelGoofy application with Prometheus metrics, and visualize these metrics using Grafana.

The motivation is to futher simulate building a realistic application. Proper monitoring setup is vital for production applications. The aim is to give students hands-on experience working with metrics, Prometheus and Grafana. The assignment only specifies that the HTTP request metrics need to be exported. Other custom metrics are left to be exported by the students as they deem fit. All this is to give students an insight into what it takes to create such graphs and set up a working monitoring system. CI/CD is also necessary for such applications as it eases the development process and collaboration.

The submitted solution must have a successfully running pipeline that implements all of the stages and operations described in the assignment. Additionally, the application must be properly instrumented while exposing Prometheus metrics and a Grafana dashboard has to be created for the service with at least six visualizations.

5.18 Bonus Lecture: Projects

The motivation for this lecture is to provide students with an overview of state-of-the-art libraries that can be used in the final Go projects. The technologies are split up into multiple sections depending on their use case.

Outline

- CLI
- TUI
- Desktop
- Web
- Open-source

Content

The Command Line Interface (CLI) section provides basic information about the most frequently used libraries to implement command-line applications. These libraries include the Cobra[61] and the CLI[62].

The lecture continues by introducing the Bubbletea[63] framework and its ecosystem for creating Terminal User Interface (TUI) applications.

The third section introduces possible libraries for implementing desktop applications. The first framework it introduces is Wails[64], an alternative to JavaScript Electron [65], or Rust Tauri[18] frameworks. The second mentioned library is Fyne[66], which, also from desktop environments, supports the iOS and Android mobile platforms.

The web sections mentions the possibility of creating a web application with a combination of Go backend and an optional frontend technology. The HTMX[17], an HTML extension, is mentioned as an alternative to traditional single-page applications (SPAs).

The lecture ends with giving multiple links to resources related to open-source development or possible Go projects.

6 Conclusion

The goal of this thesis was to design a comprehensive course on Go. This work reflected on the previous run of the Go course in the Red Hat format. The thesis also mapped the ground and the current state of similar courses at our faculty. Based on these observations, decisions were made that affected the overall course design.

6.1 Results

The thesis designed and implemented new materials that collectively constitute a comprehensive Go course. The materials include expanded and reworked lectures that now span the whole semester, some of which are based on the existing Go course maintained by Red Hat. Additionally, completely new homework assignments and exercises were created to promote active work throughout the semester. Furthermore, all of the newly created materials are open-sourced and publicly accessible on GitHub, so anyone is free to use them and adapt.

6.2 Future work

The course will have to evolve as the language and technologies evolve. I am currently dedicated to maintaining and developing the course to keep it up-to-date so it can be taught in the future. I plan to continue my studies at the faculty as a master's degree student, so the course should run in this format for at least the next two years. However, I will most likely continue tutoring it even in the following years.

Apart from keeping the course up-to-date, it could also be expanded with additional topics as described in the section 4.6. These topics will further improve the goal of the course being practice oriented.

The Go course will be run under the Domain-specific Development course the next semester. It could be later extracted into its own separate course to eliminate some of the limitations imposed by running under a parent course, such as the shared extent and intensity.

As discussed in the section 4.6.4, a new course could also be created in the future to cover some of the topics not necessarily tied with Go. This course could serve as a prerequisite to the Go course and other similar courses in the future.

Bibliography

1. GOOGLE LLC. *Go* [online]. [visited on 2024-05-19]. Available from: `https://go.dev`.
2. RED HAT. *GoCourse* [online]. [visited on 2024-05-07]. Available from: `https://github.com/RedHatOfficial/GoCourse`.
3. GITHUB INC. *GitHub* [online]. [visited on 2024-05-17]. Available from: `https://github.com`.
4. PIKE, Rob. *Using Go at Google* [online]. 2022. [visited on 2024-05-19]. Available from: `https://go.dev/solutions/google/`.
5. HOARE, C.A.R. Communicating Sequential Processes. *Commun. ACM*. 1977.
6. THE LINUX FOUNDATION. *Cloud Native Computing Foundation* [online]. [visited on 2024-05-17]. Available from: `https://www.cncf.io`.
7. GOOGLE LLC. *Go for Cloud & Network Services* [online]. [visited on 2024-05-19]. Available from: `https://go.dev/solutions/cloud`.
8. GOOGLE LLC. *Command-line Interfaces (CLIs)* [online]. [visited on 2024-05-19]. Available from: `https://go.dev/solutions/clis`.
9. FOUNDATION, Rust. *Rust* [online]. [visited on 2024-05-19]. Available from: `https://www.rust-lang.org`.
10. RITCHIE, Dennis. *C* [online]. [visited on 2024-05-19]. Available from: `https://en.wikipedia.org/wiki/C_(programming_language)`.
11. GOUY, Issac. *The Computer Language Benchmarks Game* [online]. [visited on 2024-05-19]. Available from: `https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html`.
12. GOOGLE LLC. *Go for Web Development* [online]. [visited on 2024-05-19]. Available from: `https://go.dev/solutions/webdev`.
13. GOOGLE LLC. *Development Operations & Site Reliability Engineering* [online]. [visited on 2024-05-19]. Available from: `https://go.dev/solutions/devops`.

14. CORPORATION, Microsoft. *TypeScript* [online]. [visited on 2024-05-20]. Available from: https://www.typescriptlang.org.

15. CORPORATION, Microsoft. *React* [online]. [visited on 2024-05-20]. Available from: https://react.dev.

16. DOCKER INC. *Docker* [online]. [visited on 2024-05-17]. Available from: https://www.docker.com.

17. BIGSKYSOFTWARE. *HTMX* [online]. [visited on 2024-05-08]. Available from: https://github.com/bigskysoftware/htmx.

18. TAURI-APPS. *Tauri* [online]. [visited on 2024-05-08]. Available from: https://github.com/tauri-apps/tauri.

19. VMWARE. *Spring* [online]. [visited on 2024-05-17]. Available from: https://spring.io.

20. PROMETHEUS AUTHORS. *Prometheus* [online]. [visited on 2024-05-17]. Available from: https://prometheus.io.

21. GRAFANA LABS. *Grafana* [online]. [visited on 2024-05-17]. Available from: https://grafana.com/grafana/.

22. INTERNET ENGINEERING TASK FORCE. *OAuth2* [online]. [visited on 2024-05-17]. Available from: https://oauth.net/2/.

23. OPENAI. *ChatGPT*. 2024. Available also from: https://www.openai.com/chatgpt. Accessed: 2024-05-19.

24. WIKIPEDIA. *GitHub Wikipedia* [online]. [visited on 2024-05-20]. Available from: https://en.wikipedia.org/wiki/GitHub#:~:text=As%20of%20January%202023%2C%20GitHub,host%20as%20of%20June%202023..

25. GOOGLE LLC. *Go Present* [online]. [visited on 2024-05-17]. Available from: https://pkg.go.dev/golang.org/x/tools/present.

26. MARP TEAM. *Marp* [online]. [visited on 2024-05-17]. Available from: https://marp.app.

27. GOLANG-STANDARDS. *project-layout* [online]. [visited on 2024-05-17]. Available from: https://github.com/golang-standards/project-layout.

28. INTERNET ENGINEERING TASK FORCE. *JWT* [online]. [visited on 2024-05-19]. Available from: `https://jwt.io`.

29. GRPC AUTHORS. *gRPC* [online]. [visited on 2024-05-17]. Available from: `https://grpc.io`.

30. THE GRAPHQL FOUNDATION. *GraphQL* [online]. [visited on 2024-05-17]. Available from: `https://graphql.org`.

31. NATS AUTHORS. *NATS* [online]. [visited on 2024-05-17]. Available from: `https://nats.io`.

32. THREE DOTS LABS. *Watermill* [online]. [visited on 2024-05-17]. Available from: `https://watermill.io`.

33. CENTRIFUGAL LABS LTD. *Centrifugo* [online]. [visited on 2024-05-19]. Available from: `https://centrifugal.dev`.

34. HARSANYI, Teiva. *100 Go Mistakes and How to Avoid Them*. Manning Publications, 2022.

35. GOOGLE LLC. *Go Packages* [online]. [visited on 2024-05-22]. Available from: `https://pkg.go.dev`.

36. DIJKSTRA, Edsger W. Cooperating sequential processes. In: *Programming languages : NATO Advanced Study Institute : lectures given at a three weeks Summer School held in Villard-le-Lans, 1966 / ed. by F. Genuys*. Academic Press Inc., 1968, pp. 43–112.

37. GOOGLE LLC. *Go Sync Package* [online]. [visited on 2024-05-22]. Available from: `https://pkg.go.dev/sync`.

38. THE LINUX FOUNDATION. *OpenAPI* [online]. [visited on 2024-05-21]. Available from: `https://www.openapis.org`.

39. PODMAN. *Podman* [online]. [visited on 2024-05-21]. Available from: `https://podman.io`.

40. ATOMICJAR, Inc. *Testcontainers* [online]. [visited on 2024-05-21]. Available from: `https://testcontainers.com`.

41. THE KUBERNETES AUTHORS. *Kubernetes* [online]. [visited on 2024-05-21]. Available from: `https://kubernetes.io`.

42. JMOIRON. *sqlx* [online]. [visited on 2024-05-20]. Available from: `https://github.com/jmoiron/sqlx`.

43. SQLC-DEV. *sqlc* [online]. [visited on 2024-05-20]. Available from: https://github.com/sqlc-dev/sqlc.

44. JINZHU. *GORM* [online]. [visited on 2024-05-20]. Available from: https://gorm.io/.

45. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL* [online]. [visited on 2024-05-20]. Available from: https://www.postgresql.org.

46. GITLAB. *GitLab CI/CD* [online]. [visited on 2024-05-17]. Available from: https://about.gitlab.com/topics/ci-cd/.

47. GITHAB. *GitHub Actions* [online]. [visited on 2024-05-17]. Available from: https://github.com/features/actions.

48. TRAEFIK LABS. *Traefik Proxy* [online]. [visited on 2024-05-17]. Available from: https://traefik.io/traefik/.

49. CADDY AUTHORS. *Caddy* [online]. [visited on 2024-05-17]. Available from: https://caddyserver.com.

50. GOOGLE LLC. *Google Cloud Platform* [online]. [visited on 2024-05-17]. Available from: https://cloud.google.com/?hl=en.

51. GOOGLE LLC. *Google Cloud for Faculty* [online]. [visited on 2024-05-17]. Available from: https://cloud.google.com/edu/faculty?hl=en.

52. GOOGLE LLC. *Google Cloud Run* [online]. [visited on 2024-05-17]. Available from: https://cloud.google.com/run?hl=en.

53. GOOGLE LLC. *Google Cloud Compute Engine* [online]. [visited on 2024-05-17]. Available from: https://cloud.google.com/products/compute?hl=en.

54. GOOGLE LLC. *Google Cloud SQL* [online]. [visited on 2024-05-17]. Available from: https://cloud.google.com/sql?hl=en.

55. OPENTELEMETRY AUTHORS. *OpenTelemetry* [online]. [visited on 2024-05-17]. Available from: https://opentelemetry.io.

56. GRAFANA LABS. *Loki* [online]. [visited on 2024-05-17]. Available from: https://grafana.com/loki/.

57. JAEGER AUTHORS. *Jaeger* [online]. [visited on 2024-05-17]. Available from: https://www.jaegertracing.io.

58. OPENTELEMETRY AUTHORS. *OpenTelemetry Specification* [online]. [visited on 2024-05-20]. Available from: `https://opentelemetry.io/docs/specs/otel/logs/`.

59. OPENTELEMETRY AUTHORS. *OpenTelemetry Demo* [online]. [visited on 2024-05-20]. Available from: `https://github.com/open-telemetry/opentelemetry-demo`.

60. OPENSEARCH CONTRIBUTORS. *OpenSearch* [online]. [visited on 2024-05-20]. Available from: `https://opensearch.org`.

61. SPF13. *Cobra* [online]. [visited on 2024-05-08]. Available from: `https://github.com/spf13/cobra`.

62. URFAVE. *CLI* [online]. [visited on 2024-05-08]. Available from: `https://github.com/urfave/cli`.

63. CHARMBRACELET. *Bubbletea* [online]. [visited on 2024-05-08]. Available from: `https://github.com/charmbracelet/bubbletea`.

64. WAILSAPP. *Wails* [online]. [visited on 2024-05-08]. Available from: `https://github.com/wailsapp/wails`.

65. ELECTRON. *Electron* [online]. [visited on 2024-05-08]. Available from: `https://github.com/electron/electron`.

66. FYNE-IO. *Fyne* [online]. [visited on 2024-05-08]. Available from: `https://github.com/fyne-io/fyne`.

67. BODNER, Jon. *Learning Go*. O'Reilly, 2024.

68. HOLMES, Joel. *Shipping Go*. Manning Publications, 2023.

A Course materials

All course materials are maintained under the course-go[1] GitHub organization. To preserve the current state of the materials, all relevant repositories were exported and compressed into an attachment to this thesis. Each top-level directory in the attached archive maps to a GitHub repository. The directories contain versions of materials that existed when submitting this thesis and are most likely out-of-date as you read this text. Inspect the respective GitHub repositories if you want to view the updated version.

The GitHub repositories for respective directories can be found under the following URLs:

- lectures[2]
- exercises[3]
- homework[4]
- code[5]
- course[6]
- ping-pong[7]

A.1 lectures

The lectures directory contains all slides and their respective assets (images, code samples, videos, etc.). The supplied README in the directory guides you on how to run the slides.

1. `https://github.com/course-go`
2. `https://github.com/course-go/lectures`
3. `https://github.com/course-go/exercises`
4. `https://github.com/course-go/homework`
5. `https://github.com/course-go/code`
6. `https://github.com/course-go/course`
7. `https://github.com/course-go/ping-pong`

A.2 exercises

The exercises directory contains all exercises with assignments and related code. Each exercise is in a separate directory containing a README specifying the assignment.

A.3 homework

The homework directory contains all homework assignments. Each homework assignment is located in its respective directory. The assignments are given in the form of a README.

A.4 code

The code directory contains source code that either implements solutions to some of the exercises or implements some form of functionality used throughout the exercises.

A.5 course

The course directory contains information that is the starting point when people want to learn more about the course.

A.6 ping-pong

The ping-pong directory contains a simple REST API project used for teaching deployments and their automation. It is split up from the code project as it simplifies the deployments.

B Catalog specification

This chapter specifies the common catalog used to describe courses at Masaryk University.

Extent and Intensity

As was already hinted in the section 4.1.4, the course is designed to the extent of four ECTS credits. This does not currently match the extent of the PB173: Domain-specific Development parent course and the project requirements will need to be relaxed until the course becomes independent.

The following expression notes the intensity of lectures, seminars, and homework, plus the type of completion, respectively.

(1/1/1) + 1 (colloquium)

Prerequisities

No hard requirements are enforced. However, students are expected be proficient in using SQL (PB154)[1], must possess programming fundamentals (IB111)[2] and have a basic understanding of networking (PB156)[3] and virtualization (PB152)[4] to the extent of the referenced courses.

Enrolment limitations

The course is intended to be run with up to 20 students. It can later be scaled to a higher number, but currently, the intent is to test the new course in practice and interact more with students, which does not scale well and would require opening a second seminar.

---

1. https://is.muni.cz/predmet/fi/podzim2023/PB154?lang=en.
2. https://is.muni.cz/predmet/fi/podzim2023/IB111?lang=en.
3. https://is.muni.cz/predmet/fi/jaro2024/PB156?lang=en.
4. https://is.muni.cz/predmet/fi/jaro2024/PB152?lang=en.

Course objectives

Students will understand the fundamentals of the Go programming language and its common use cases in practice. They will acquire the required knowledge for entry-level Go developer positions while writing idiomatic Go code, and they will be capable of applying this knowledge to real-life projects.

Learning outcomes

Students will be able to:

- Write idiomatic Go code.
- Understand the Go concurrency model.
- Profile and optimize Go applications.
- Develop REST API services.
- Containerize and deploy their applications.
- Implement persistence leveraging SQL databases.
- Instrument applications with various types of telemetry.

Syllabus

1. Introduction: (Introduction to Go, IDEs and editors, Installing Go, Running Go)

2. Fundamentals #1: (Packages & Visibility, Variables, Data types, Control flow, Functions, Pointers, User-defined data types)

3. Fundamentals #2: (Interfaces, Errors, Arrays, Slices, Maps, Range)

4. Concurrency & parallelism: (Goroutines, Runtime, Channels, Select, Related packages)

5. Advanced #1: (Generics, Packages, Testing)

6. Advanced #2: (Benchmarks, Optimizations, CGo, Unsafe & Reflect)

7. REST APIs: (JSON, HTTP, REST API, HTTP package, Routers & Web frameworks, OpenAPI, Templating)
8. Containers: (Containerization, Docker, Kubernetes)
9. Databases: (SQL, RDBMSs, Migrations, SQL, sqlx, sqlc, GORM)
10. Infrastructure: (CI/CD, Infrastructure, Google Cloud Platform)
11. Observability: (Health, Logs, Metrics, Traces, OpenTelemetry)

Literature

- BODNER, Jon. Learning Go. O'Reilly, 2024.[67]
- HARSANYI, Teiva. 100 Go Mistakes and How to Avoid Them. Manning Publications, 2022.[34]
- HOLMES, Joel. Shipping Go. Manning Publications, 2023.[68]

Teaching methods

In-person lectures with hands-on exercises and reviewed homework assignments and projects.

Assessment methods

Multiple homework assignments (fifty points) and a final project (fifty points), including its defense. Seventy out of the one hundred points are required to successfully complete the course.