

# Modern Software Development

## A Simple Guide to Today's Technology

Technical Concepts Explained Simply

December 15, 2025

# What We'll Learn Today

- **Docker:** Shipping software like shipping boxes
- **SSR (Server-Side Rendering):** Where should your code run?
- **SvelteKit:** Modern web framework
- **Go vs Python:** Different approaches to writing programs

# Think of Docker Like Shipping

## Traditional Shipping Problem

Before containers, shipping goods was messy:

- Books packed differently than clothes
- Different sizes required different trucks
- Items could get damaged during transport
- No standard way to pack things

**Solution:** The shipping container (standardized box)

- All items fit in same sized box
- Easy to stack and transport
- Protected from damage
- Works on ships, trucks, trains

# Docker Works the Same Way

**Docker Container** = A box that contains everything your software needs:

- Your code
- Required libraries
- Operating system basics
- Configuration files

**Why this matters:**

- Works the same on your computer, my computer, or a server
- No more “It works on my machine” problems
- Easy to update and ship out

# Docker: The Real-World Benefit

## WITHOUT Docker:

- Developer installs software locally
- Moves to production server
- Server has different setup
- Software breaks
- Hours spent debugging

## WITH Docker:

- Developer creates container
- Container works everywhere
- Server runs exact same container
- Everything works
- Deploy in minutes

**Think of it:** Like mailing a self-contained lunch box that works anywhere, instead of just mailing ingredients.

# Docker Vocabulary Made Simple

**Image** A template or blueprint (like a recipe)

**Container** A running copy of that template (like the cooked meal)

**Dockerfile** Instructions for building the image (the recipe itself)

**Docker Hub** Library of pre-made images (cookbook)

## Analogy:

- Image = Recipe
- Container = Meal you cooked from that recipe
- Dockerfile = How you write the recipe
- Docker Hub = Book of recipes

# The Server vs Browser Dilemma

Modern websites need to decide: **Where does the work happen?**

## Browser (Client)

- Your computer
- User's phone
- Software runs here
- Faster response
- More processing power needed on device

## Server (Cloud)

- Powerful computer far away
- Software runs here
- Works on slow devices
- Requires internet
- Less power needed locally

# Server-Side Rendering (SSR) Explained

## What is SSR?

The server does the heavy work BEFORE sending data to your browser.

## Real-World Analogy:

**Restaurant Kitchen (Server)** Prepares your meal completely

**Your Dining Room (Browser)** Receives a finished, hot meal

**Result** Fast experience, ready to eat immediately

## Without SSR:

- Browser receives empty plate
- Browser must cook the meal
- Takes longer
- User waits more

# SSR Benefits for Different Users

## Who benefits from SSR?

- **Slow internet users:** Less data to download
- **Mobile users:** Less work for phone battery
- **Old devices:** Don't need powerful hardware
- **SEO (Google search):** Better for search engines

## Why companies like SSR:

- Users see content faster
- Better search engine ranking
- Works better globally
- Better for all devices

# What Are Web Frameworks?

**A Framework:** A set of ready-made tools for building websites

**Think of it like building a house:**

- Without framework: You make bricks, shape wood, mix cement
- With framework: Pre-made walls, doors, windows ready to assemble

**SvelteKit:**

- Helps you build modern, fast websites
- Includes SSR capabilities
- Provides reusable components
- Handles routing (page navigation)
- Optimizes for users

# SvelteKit: The Simple Framework

**SvelteKit's Philosophy:** Write less code, do more

## What makes it simple:

- Smaller file sizes
- Less boilerplate
- Reactive by default
- Clean syntax

## Best for:

- Quick prototypes
- Small teams
- Performance-critical apps
- Learning web dev

**Analogy:** Like a small, nimble sailboat - fast and responsive

# What is Go?

**Go:** A modern programming language designed for servers and APIs

**Go's Philosophy:** Simplicity, speed, and reliability

## Key characteristics:

- **Super fast** - Runs at incredible speeds
- **Simple syntax** - Easy to read and understand
- **Compiled** - Turns into fast machine code
- **Perfect for servers** - Handles thousands of users
- **Small programs** - Easy to package

## Why we chose Go:

- Perfect for building APIs (like our Fibers backend)
- Containerizes beautifully with Docker

# How Go Works: Compiled vs Interpreted

## Three different approaches:

- Go (Compiled)**
- Converts code to machine language BEFORE running
  - Like baking a cake completely before serving
  - Super fast when it runs
  - Must rebuild for each operating system

- Python (Interpreted)**
- Reads and runs code line by line
  - Like cooking while guests watch
  - Slower execution
  - Works on any computer without rebuilding

- Java (Compiled + Interpreted)**
- Compiles to middle format
  - Runs in special environment (JVM)
  - Medium speed

# Go Compilation Process

## How Go turns code into a fast program:

### ① You write Go code

- Simple, readable syntax

### ② Go compiler reads it

- Checks for errors
- Optimizes the code

### ③ Converts to machine code

- Native instructions for the CPU
- Specific to the operating system

### ④ Creates executable file

- Standalone program
- No extra dependencies needed
- Ready to run

# Go vs Python: Speed Comparison

## Why Go is faster:

### Go (Compiled)

- Code pre-processed
- Direct machine instructions
- Runs at CPU speed
- Minimal overhead
- **VERY FAST**

### Python (Interpreted)

- Code read line by line
- Decisions made while running
- Translation overhead
- Extra interpretation layer
- **SLOWER**

## Real world example:

- Go handles 10,000 users: Still fast
- Python handles 1,000 users: Gets slow
- Go uses 50MB RAM: Light and lean
- Python uses 500MB RAM: Heavy

# Understanding Composition: Go's Way

## Go Example: Building a Dog

```
1 type Animal struct {
2     Name string
3 }
4
5 func (a Animal) Speak() string {
6     return "Some sound"
7 }
8
9 type Dog struct {
10    Animal // Include Animal in Dog
11    Breed string
12 }
13
14 // Override the Speak method
15 func (d Dog) Speak() string {
16     return "Woof!"
17 }
```

Listing 1: Go Composition Example

# Go Advantages for Backend Development

## Why Go is perfect for our backend:

- **Lightning Speed** - Compiled code runs super fast
- **Low Memory** - Uses minimal resources
- **Easy Concurrency** - Handles many users simultaneously
- **Simple Syntax** - Easy to write and understand
- **Cross-Platform** - Runs on Linux, Windows, Mac
- **Docker Friendly** - Creates tiny, fast containers
- **Single Binary** - One file to deploy

## Fibers Framework:

- Built in Go for maximum speed
- Perfect for building APIs
- Minimal overhead

# How Everything Works Together

## A typical modern web application:

### ① Backend (Server): Written in Go or Python

- Handles business logic
- Stores data safely
- Runs in Docker containers

### ② Frontend (Website): Built with SvelteKit

- Uses SSR for fast loading
- Beautiful user interface
- Runs in Docker containers

### ③ Deployment: Docker containers everywhere

- Same containers on developer's computer and server
- Easy to scale
- Easy to update

# SvelteKit + Go Fibers: A Real Example

## What is Fibers?

Fibers is a lightweight, fast web framework for Go - perfect for building APIs.

## How they work together:

### Go Fibers Backend

- Receives requests from SvelteKit
- Processes business logic
- Talks to databases
- Returns data as JSON
- Very fast and lightweight

### SvelteKit Frontend

- Displays beautiful interface
- Sends user requests to Fibers backend
- Receives JSON data
- Shows updated content instantly

# SvelteKit + Go Fibers Architecture

## Communication Flow:

- ① **User clicks button in SvelteKit**
- ② **SvelteKit sends request to Go Fibers API**
  - Example: “Give me user data”
  - Sent as HTTP request
- ③ **Go Fibers processes the request**
  - Checks permissions
  - Queries database
- ④ **Fibers sends back JSON response**
  - Pure data, lightweight and fast
- ⑤ **SvelteKit updates display instantly**
  - User sees fresh data
  - No page refresh needed

# Why SvelteKit + Go Fibers Works Great

## Perfect Combination: SvelteKit Strengths

- Beautiful UIs
- Small bundle size
- Fast rendering
- Reactive updates
- SSR support

## Go Fibers Strengths

- Lightning fast
- Handles many users
- Uses little memory
- Easy to write
- Great for APIs

## Real-World Benefits:

- **Speed** - Both are optimized for performance
- **Scalability** - Can handle thousands of users
- **Simplicity** - Both use simple, clean code
- **Docker Friendly** - Both containerize perfectly

# Architecture Diagram: SvelteKit + Go Fibers

## How the pieces fit together:

### User's Browser

- SvelteKit App (beautiful interface)
- Shows data, handles clicks
  - ↓ HTTP Requests ("Get my data") ↓

### Go Fibers Server

- Processes requests
- Talks to database
- Returns JSON
  - ↓ JSON Responses (data) ↓

### User's Browser

- SvelteKit updates display
- User sees changes instantly

# Why This Stack Makes Sense

## Benefits of modern development:

- **Speed** - Users see content fast (SSR)
- **Reliability** - Everything works the same everywhere (Docker)
- **Flexibility** - Choose the right tool for each part
- **Simplicity** - Clean languages like Go
- **Performance** - Optimized frameworks like SvelteKit
- **Easy Updates** - Change and redeploy instantly

## For non-technical people:

- Software works on all devices
- Websites load super fast
- Updates happen instantly
- No more software compatibility issues

# Key Takeaways

- ① **Docker** - Like shipping containers for software. Works everywhere.
- ② **SSR** - Server does work first, sends finished product to you. Faster loading.
- ③ **SvelteKit** - Simple, lightweight, modern framework for websites.
- ④ **Go Language** - Compiled, super-fast language. Perfect for servers.
- ⑤ **Go Fibers** - Fast backend API framework. Built in Go.
- ⑥ **Go's Speed** - Compiled code is much faster than Python or Java.

**Together:** These create fast, reliable software that works everywhere.

Questions?

Thank you!