

Projects & Clients

Overview

I've taken contract work since 2014 and worked on 13 projects for 11 different clients. Some of these have been multi-year, full time contracts while others have been advisory roles or short-time projects.

Most of my customers are startups or scaleups.

Interested in working together? [See this page](#).

See [my full CV in English](#) for something a little more formal =)

reMarkable

- [Built the first cloud-based backend system](#)
- [Document Synchronization prototype](#)

IntoLife

- [Built IntoFood, a food management system](#)
- [Built a REST API backend for IntoFood](#)

Hei Petter

- [CTO, Technical leadership](#)
- [Built prototypes and an MVP](#)

SHORTCUT

- [Prototyping a new secure payment solution](#)
- [Fantasy Football League - Built cloud backend system](#)



- [MVP Monkey - Nocode MVPs for early-stage startups](#)

VILLOID

- Backend Developer



- Mimiro - Backend development



- REST API design and implementation



- SRSLY - Prototype CMS and teach Go



- Prototyping new e-commerce system



- REMA 1000 - Æ - IDP backend development and operations



- Consultant on Google Cloud Platform, Golang and web technologies

Other Projects

- CMS / Blog engine, oyvindsk.com
- Simple web-based RSS reader
- A distributed email address checker

Project descriptions

MVP Monkey - Nocode MVPs for early-stage startups

2024

Skills: No-Code Development Platforms · No-Code Development · Customer validation Web Applications · Mobile Applications

MVP Monkey was my foray into a different customer segment and a new business model.

The idea was to help founders and early stage startups to get "unstuck" from the situation where they need a product but can't get funding without a product.

Spending a month or two to build a quick, but working, app is also a better way to validate ideas.

By using NoCode and AI tools, I could create web and mobile apps much quicker and cheaper.

To speed up development I did a few things very differently:

1. Use all the new No-Code and AI tools
2. Work with the founder to get to the core idea and Value Proposition.
3. Instead of building all imaginable features, we built only what was needed to get traction and test the idea (underlying hypothesis).

Results

MVP Monkey never got enough traction, so I shut it down. Basically, it was too hard to find anyone who wanted to buy the services.

What I learned

- Validating ideas without selling something is very hard (well, I already knew that, but still)
- No-code tools - I researched around 30 different No-code and AI tools. They are no longer my main focus, but they're nice to have in my tool-box.

Technologies: Multiple AI and No-code tools for rapid development, Webflow

[back to the top](#)

Mimiro - Backend development

2023

Skills: Software Development · Database design

Mimiro supports Norwegian agriculture and food production. Farmers are both their customer and partners.

I worked with Mimiro for 10 months and helped develop their new Eana360 product. As a part of a team, the work was straight up backend development in Go. Little to no architecture or Devops work.

Technologies: ArangoDB · Go (Programming Language) · PostgreSQL · Graph Databases · AWS · Git · SQL

[back to the top](#)

HeiPetter - CTO, Technical leadership

2020 - 2021

Skills: Technical Leadership

HeiPetter is a Norwegian startup connecting with a purpose of getting people in difficult situations back to work. They do that through a digital platform, focusing on the talent and building on individual strengths.

I have been involved both as a part of the team and as a contractor doing development work.

As the technical leader my job was to translate the goals and strategy into tangible technical deliveries. I also vetted potential partners, freelancers and contractors. My main contribution was technical know-how and a laser-focus on what we needed to get to the next milestone.

[back to the top](#)

HeiPetter - Built prototypes and an MVP

2020 - 2021

Skills: Nocode · Software Development · DevOps · Cloud

Nocode prototypes

Preceding the more complete MVP I created a couple of prototypes using No-Code tools. The first take was to use Bubble to create a semi-working web app, mostly to explore user signup and job creation. After hitting some problems, I switched to Adalo and created a more mobile-friendly prototype.

Technologies: Nocode, Bubble.io, Adalo

Backend for functional MVP

I worked with a freelance designer to build a prototype of the web-based platform. She did design, html and css. I wrote the backend, converted her HTML into templates, created data models and deployed the application.

Functionality: User signup and login, job registration and listing. Automatic matching (sorting) of jobs based on the user preferences.

Technologies: Go, Google Cloud Appengine and Datastore. Standard web technologies.

Results

Both the prototype and the MVP gave us something to test with potential users. It's vital to get user feedback as early as possible. They also gave us something to show to investors and other potential partners.

[back to the top](#)

REMA 1000 - Æ - IDP backend development and operations

2022

Skills: Cloud Computing · IDP · Firebase · Firestore · Go · Google Cloud Platform (GCP) · Digital Authentication · Multi-factor Authentication · Authentication Systems

Operation and development of the idp (login/authentication) part of the Æ app.

(On this project I was a subcontractor though Shortcut and 7n)

[back to the top](#)

Fantasy Football League - Built cloud backend system

2021

Skills: · Technical Leadership · Software Development · DevOps · Backend Architecture

Fantasy Football League (FFL) was a fantasy-football (soccer) mobile app. It had several new features and innovative game-play to set it apart from its competitors.

We worked in a small team of 3 people. The 2 others were the customer who set the overall direction and was the domain expert, and one developer who made the iPhone app

I made the backend system, which contained:

- A Go program that contained the game logic and exposed an API to the iOS client.
- Integrations with third parties, mostly real-time information about football matches, players, goals, substitutions etc.
- Database for storing simple things like customer information, but also complex schema's for making a real-time "view" of the real world.
- Servers to run the integrations, game logic and REST API. In this case a managed PaaS (Google Cloud Run), so there was little "devops".

This was a fun project with fast development, great teamwork and exiting technical challenges. Unfortunately, we developers were needed on other projects after finishing the beta. The development continued, but it never gained enough traction, and the app was later discontinued.

Challenges

- Real time synchronization: Unlike the competitors, FFL allowed substitutions while the game was in play.
- Complicated logic and rules for the game-play

- Anti cheats
- Novel and innovative game-play meant a lot of trial-and-error development
- Pre Product-Market-Fit
- Very fast development of new features
- 3rd party integrations

Results

The development and technical aspects was a great success. We created a fast, easy-to-use app, with novel game-play and real-time attributes, in record time.

However, FFL never found product-market-fit. My opinion, and I believe the customer would agree, is that not enough effort was put into marketing and customer research.

(On this project I was a subcontractor through Shortcut and 7n)

Prototyping a new secure payment solution

September 2020 - December 2020

Skills: R&D · Problem Solving · Secure development

This unnamed project was one of three greenfield projects I worked on for Shortcut customers in 2020.

Millions are lost every year because of fraudulent b2b payments. The banking infrastructure itself is secure, but money can still be sent to the wrong account number. This project would solve that by guaranteeing correct and unforgeable information.

This was an interesting project for me because it combined many of my interests:

- Security
- Research and prototyping
- Technical problems solving

We were a team of 2-3 people. Among other things, I created a Certificate Authority and helped the other developers with strong, on-device encryption and signing.

(On this project I was a subcontractor through Shortcut and 7n)

[back to the top](#)

reMarkable - Built the first cloud-based backend system

2016 - 2020

Skills: Open-Source Software · Technical Leadership · Software Development · DevOps · Backend Architecture

reMarkable is a very successful Norwegian startup. They have created a new type of device, a "paper tablet" to read, write and sketch on. I started working with them early on, when they were just eight people.

The company has grown to around 500 people and have been valued at 1 billion USD .

I was the only person working on the backend for the first few years and built a cloud based backend from scratch. Since then the system have expanded and more people joined the cloud team.

The main feature of the cloud service is syncing notes, drawings and documents from the users reMarkable device to other devices such as phones, tablets and computers. The backend also handles Authentication, Authorization and integration with third party services.

Challenges

- Real-time: Parts of the system require soft real time attributes.
- Novel: Parts of this system are quite novel. This, combined with the typical restricted resources of a startup, means we can not blindly follow "best practices".
- Scale: Potentially large amount of concurrent users

Results

The backend system got reMarkable from 0 to 1 and handled the very successful launch and the first few years of operations with only minor improvements.

Even though he worked as a consultant he immediately took responsibility for and drove the development of our entire cloud solution from the ground up. When Øyvind came in there wasn't a single line of code written, nor any architecture planned. He took on the responsibility for planning and executing what was necessary to go from high-level ideas about what our cloud solution should and could be, to what we have today.

Øyvind not only single-handedly wrote all the code for all parts of the cloud solution, he also designed the high-level architecture and drove the design of the interfaces and protocols between our devices, applications and servers.

After we launched and shipped Øyvind decided to keep working for us to ensure a smooth transition while we recruited more in-house talent. (..) the fact that Øyvind chose to stick around from 2016, through our pre-order

campaign in 2017, the crunch time before shipping in 2018 and continually maintaining, improving and on-boarding new developers into 2020 shows real dedication, integrity and ability to take ownership of what he creates.

Press

- [reMarkable sells 1M devices, closes funding at \\$1B valuation](#)
- [Ny enhjørning: Slik solgte Remarkable skrivebrett for 2,7 milliarder mens hele verden slet med å levere varene](#)
- [Gründersuksess: Solgte «papirnettbrett» for 3 millioner kroner – på under to døgn – E24](#)
- [Milliardhopp for norskdesignede skrivebrett: Innfører zen-dag for at ansatte skal reflektere | DN](#)

Other links

- [reMarkable Website](#)

[back to the top](#)

reMarkable - Document Synchronization prototype

2018 - 2019

Skills: Research and Development · Technical Leadership

I was part of a small team of 3 developers who prototyped a novel way of synchronizing files across devices (reMarkables, phones, computers etc).

Goals: Achieve fast and correct document synchronization while using as little bandwidth as possible.

We achieved this with known, but somewhat niche methods like Content-Addressable Storage and Merkle Trees.

(For a description of who reMarkable are and what they do, see above)

[back to the top](#)

Intolife - Built custom SaaS platform, IntoFood

Active development: 2014 - 2017

Hosting, support and operations: 2014 - 2023

Skills: Open-Source Software · Technical Leadership · Software Development · DevOps · Backend Architecture

Role: Sole developer, architect and devops person.

I built, support and host a "a sustainable food management system" for Intolife. The web-based application, called IntoFood, helps with data-input and report generation. It also exposes a HTTP API for data exchange with partners.

This webapp was built over several years. We started with basic data import and report generation and gradually added a few needed features. It is used by customers, but is no longer actively developed. I maintain and run it in Google Cloud.

Backend, frontend, architecture

About Intolife

More and more customers want healthy & sustainable food choices. IntoLife's toolkits will help you to do this by improving menu sustainability and cutting your food waste by half. Our technology solutions put sustainability into your business operations, allowing you to develop sustainable menus and reduce your food waste

Intolife works with restaurants, caterers and other players in the food industry. They help them cut their environmental impact. And to do it in a way that makes sense for their business. Intolife can also help them use this in their marketing. Intolife is a young and emerging company that innovates on several fronts. Few things are set in stone. As with most innovative projects, they have goals and know where they want to go, but not always how to get there. It's important for them to always learn and adapt to the market.

The Project

The workflow used by Intolife before this project was based on Excel and manual data input. This worked fine. But it was time-consuming and limited the possibilities for interacting with third parties. With this project we created a fully customized web-based application (so called Software as a Service, or SaaS) for Intolife. The goals were to cut down the time required for data entry and to automatically generate reports. We also wanted to allow for future expansion and integration with other software and services. We developed this project using lean startup methodologies. This gave us more flexibility. It also saved money by avoiding the development of unnecessary features.

Results

The result is a web-based application that helps with data-input and report generation. This helped Intolife use approximately 50% less time on each of their projects. The application is used mainly by Intolife, but it's also open to other partners and customers. It generates reports with 1 click. Since flexibility is important, we develop the software in phases, with their own milestones. This made it possible to quickly incorporate the lessons learned during development into the project. We meet the short-term needs while keeping the software open for future development.

The software also laid the groundwork for future expansions, and was later expanded with a REST API. This made it possible to automatically communicate with other systems. Examples are the customer's systems and third party systems.

What we learned

- Be uncompromising when it comes to prioritizing features and keeping things simple. These are, by far, the most important factors for keeping the development costs low.
- Prioritizing features and keeping things simple also creates a better product.
- Remember to account for hosting expenses. We host the service on a Norwegian cloud provider (2020 update: It's now in Google Cloud Platform). Since the number of users is low (it's not a product for the general public) this is not too expensive. Running the service requires operational work. These are things like database backups and software upgrades and maintenance. This adds to the costs. In technical terms it might make sense to move from Infrastructure as a Service (IaaS) to a Platform as a Service (PaaS) solution. This is to move more of the operational challenges to a third party.

Press

[TV - NRK Lørdagsrevyen 9. mai 2015 \(Norwegian\)](#)

Other links

- [Homepage](#)
- [Facebook page for Intofood](#)

Technologies

- Perl 5
- Nginx
- Mojolicious
- PostgreSQL
- Linux
- Docker
- Google Cloud Platform - Compute Engine (was Zetta.io, a Norwegian IaaS)

[back to the top](#)

Intolife - Built a REST API backend for IntoFood

Active development: 2016

Hosting and maintenance: 2016 - 2023

Skills: Open-Source Software · Technical Leadership · Software Development · DevOps

REST API design and implementation (Perl5). API client example (php)

We are proud to announce the forthcoming release of the integration

platform for IntoFood. This will allow existing food service management systems to automatically connect to IntoFood and receive sustainability metrics for menus, sales and purchasing.

By integrating with IntoFood you can see the climate change impact of your menu items, test new menus, and identify hotspots where you have the greatest opportunity to be more sustainable.

— [Intolife.no/news](https://intolife.no/news)

Project background

We launched this project to make it possible to integrate the *Intolife web application* with third parties. These third parties are typically customers and partners. They can use the API to include waste and emission data (GHG) in their own software and appliances. Using the API they can get this data automatically, without human interaction.

Results

The API is up and running and is used by IntoLife customers. Documentation was written to make it easier to implement the API.

The API opens up a whole lot of new possibilities. Use-cases that would otherwise involve too much human labor are now quick and easy.

What we learned

Moving forward in the face of uncertainty. This project faced some challenges that are in many ways quite typical for startup projects. The first of these is the question of exactly what we are making. We had a good sense of where we were heading and why. But neither we or Intolife's customers and partners had a concrete case in mind. We were treading new ground and the customers do not always know exactly what they want until they see it. This led to a "catch-22" situation. We needed to show something for people to understand the use-case. But, at the same time we needed customer feedback to make it in the first place. There's no easy, magical solution to this. The way through seems to be to learn as much as possible while spending as little time and money as possible. In this case we implemented a first version of the API in cooperation with one of the customers. We will use this first version to get feedback and drive customer engagement. Improve and iterate, or **build measure learn** as Eric Ries puts it.

Extending an existing system does not have to be hard. There was also a perceived challenge to fit this new API "on top" of the existing code and data model. Although it's certainly easier to start with a clean slate, this turned out to be quite manageable. The web application was made in a way that makes it easy to extend. The right level of flexibility and fairly clean and commented code makes this possible.

Writing documentation is time-consuming. The time and effort needed to write good API documentation surprised me. It was worth it though, as having this is crucial for adaptation of the API. I've previously experienced how missing or lacking documentation can make it unnecessary

difficult to implement external APIs.

Technologies

- Perl 5
- Nginx
- Mojolicious
- PostgreSQL
- Linux
- Docker
- Google Cloud Platform - Compute Engine (was Zetta.io, a Norwegian IaaS)

[back to the top](#)

Schlumberger - Consultant on Google Cloud Platform, Golang and web technologies

Helped a team at Schlumberger Norway getting up and running with web API's, Google Cloud Platform and Go.

Goals: Avoid the most common mistakes and get up and running quicker.

Challenges

- Legacy software: They were in the early phases of transforming some of their legacy systems to use the cloud. Since legacy systems are not made with the cloud in mind, this typically poses challenges.
- All new tech stack: Beginning with cloud and a new programming language means switching tech stack completely and therefore learning a number of new technologies at the same time. Luckily, there are some easy wins to be had.

[back to the top](#)

Villoid (Sobazaar) - Backend development

May 2015 - Aug 2015

Skills: Software Development · Database management and problem solving

Villoid (previously Sobazaar) was a social fashion and shopping app for Apple devices. They had a fairly large user-base in Norway and expanded to the US autumn 2015. Villoid later changed their business-model to focus on their web-shop.

At the time, Villoid was a startup and things moved fast. They needed someone with backend skills to join their team, but did not have the time to complete a normal hiring process. One of the

advantages of using consultants and freelancers is the short start-up time.

I implemented new features and fixed bugs in their backend. I also worked on scaling the backend and speeding up database queries. The expanding user-base and technical debt created some unique technical challenges.

Press

- [Forbes](#)
- [Reuters](#)
- [Dagens Næringsliv \(Norwegian\)](#)

Other links

- [Homepage](#)

Technologies

- Python
- Django
- Cloud Computing (IaaS): Amazon Web Services (AWS)
- Docker
- MySQL

[back to the top](#)

Picterus - REST API design and implementation

Feb 2015 - Mar 2015

Skills: Software Development · Prototyping · API design

REST API design and implementation. Backend (php)

Picturus is a medical app designed to diagnose Jaundice in newborns. Untreated jaundice in newborns is responsible for 114,000 deaths and 65,000 permanent brain damages each year. More than three quarters of these deaths occur in the poorest regions of the world, in sub-Saharan Africa and south Asia. Cheap treatment is available through e.g. sunlight, but the diagnostic devices in use today cost around 10,000 dollars, making them practically unavailable in low-resource settings.

Picturus therefore developed a smartphone app capable of diagnosing this condition.

As a sub-contractor I developed a small part of the MVP for this app. With my specialty in back-end systems I created a REST API to support features in the app that relies on something outside the device (phone) itself.

Links

- [Smartphone-based monitoring of jaundice in newborns](#)
- [Homepage](#)
- [LinkedIN](#)

Technologies: php, PostgreSQL, SQLite

[back to the top](#)

Shout Out Referral - Prototype design and implementation of a new social media engagement platform for e-merchants

January 2015 - March 2015

Skills: Research and Development · Software Development · Backend Architecture

Project background

A social media engagement platform for e-merchants

Shout out Referral is a referral system for web-shops that merges e-commerce with social media. It was a new project from individuals with success from other e-commerce ventures. They had a good idea and e-commerce experience. What they needed was someone with technical know-how and developer background to help them move forward from the idea phase.

Results

I helped draw up the technical architecture and the rest of the technology stack. After figuring out the core features, I implemented a simple prototype. The purpose was twofold: 1) to explore the possibilities and limitations of different social media providers, and 2) to show off the idea.

We ended up with a simple, but working prototype. It shows off a typical use-case for the product. Developing the prototype taught us much. We explored the different social media providers. The merging of different technologies unveiled some unexpected results. Also, we learned that authenticating with many social media providers makes identity handling challenging.

Developing a prototype was worth the time and effort. It raised questions that should be raised sooner rather than later. There are also many assumptions made early on. These assumptions do not always hold when theory meets real life. So it's important to check these assumptions as early as possible.

What we learned

This project depends heavily on social media integrations. We soon discovered that not all the providers have equally good APIs. Also, the different providers have different policies and guidelines. Since these factors are outside our control, these limitations can not be "fixed". We must

work around them. The advantage of following lean practices are clear here. We discovered these limitations early in the process, before wasting time on creating the wrong plans and unneeded code.

Using new technology can be unpredictable. There can be hidden advantages and disadvantages. Hidden disadvantages are likely more common, the advantages are often well promoted. In making the prototype we used somewhat new and unknown technology. In particular, the open source OAuth / OAuth2 library for Golang did not support App Engine. I therefore had to modify it. This was unexpected and made developing the social media log-ins ten times as time-consuming as expected. However, more often than not, the advantages new technologies bring will be worth the effort.

Implementation details

- Go (Golang)
 - “Goth” OAuth / OAuth2 library
 - Gorilla Web libraries
- Google App Engine (Cloud PaaS)
 - Datastore
- Social Media APIs: Facebook, Twitter, Pinterest

The prototype connects to social media like Facebook and Twitter. The full version would run in, and integrate with, a webshop.

I wrote it in Go and ran it on Google's App Engine Platform as a service (PaaS).

I really like the idea of PaaS, especially for projects that are going to grow big. I did feel the pain on working with App Engine though:

- Code must be written for especially for App Engine (note: No longer true for App Engine and Cloud Run)
- This, coupled with all the custom infrastructure really lock you in to App Engine, you can't easily quit
- Steep learning curve
- Many 3rd party packages do not work on the App Engine. It took me days, not minutes, to get Facebook and Twitter integration to work
- Can be expensive

But let's not forget the positives: * Scales totally automatically and indefinitely (If you use it correctly). This is how people often think all clouds work, but that's almost never the case. Especially for the database / nosql / datastore.

- The cost scales linearly with what you use (if I'm reading the pricing correctly). Unlike Heroku, for example. This makes a lot of business sense in many cases. Start out free or very cheap and pay more as the usage grows. Often the growing usage and expenses means more paying customers.

- Google cloud has so many cool tools to play with. Some of them give you the power of thousands of servers and can thus take a heavy task from 10 minutes to 10 seconds.

Technologies: Google Cloud Platform, App Engine (PaaS), Go

[back to the top](#)

Srsly - Prototype CMS and teach Go

2016

Skills: Go (Golang) · Prototyping · Instruction

Srsly is a Swiss media company and technology firm. I helped them get going with Go, as well as architecture and implementation of a new Content Management System / blog engine.

<https://www.srsly.ch/>

[back to the top](#)

Other projects

This blog, oyvindsk.com

I wrote my own website backend and blog engine in Go, using AsciiDoc and Tachyons css . It's a playground for testing new technologies and crazy ideas =)

<https://github.com/oyvindsk/web-oyvindsk.com>

[back to the top](#)

RSS reader

A simple, web-based and self-hosted, RSS reader I made for myself.

<https://git.sr.ht/~oyvindsk/rss-web-reader>

[back to the top](#)

A distributed email address checker

Like most of these projects it's partly for the usefulness, partly for learning and partly for the fun of it. This project chats with an SMTP server to find out if an email address is actually in use (works surprisingly well). Now, I'm planning to run a few million addresses through this, so even with go's concurrency, it's going to need more than 1 server. This is not going to be used for spam, I promise :). It has a REST API for submitting email addresses. It works, but it's not finished.

Challenges

- Avoid getting blocked by smtp servers. Solution: Smart throttle and fan out to multiple machines

(ip's)

- Distributing the work and gathering the results. I chose to use NSQ to communicate between the processes and machines. It's a distributed message bus made by bit.ly. In terms of learning, it's been great, I've learned a lot. But in retrospect, NSQ might be a little too "low level" for this project. Something like Resque/Sidekiq, Gearmand or one of the Go alternatives would have been **much** easier to work with. With a 1-way message bus like NSQ you are responsible for matching replies to requests, and other things a job system gives you for free. On the other hand NSQ does not have any single point of failures, it's fast and you can just hook up new parts to the stream to get messages on the fly.
- How should the throttling work? Per source IP? Per email domain (@gmail.com)? Can we save time by not re-connecting to the same smtp server all the time? How much traffic can you send to a server before you create problems for them? Or before they block you? Solution: Start simple and "slow". Gradually crank it up and incorporate what you learn.
- Running it in containers (Docker) changes things a little bit. The biggest reason to run it on multiple machines is to get many source ip's. But with containers it could place all the workers on the same machine. It was tested in something called Rancher, which "fixes" this, but has a bug that complicates the NSQ deployment.

[Github \(with code and even more text\)](#)

[NSQ](#)

[back to the top](#)