[Kevin Wallentin Carlsen]

04/03/2024

Space Inventor A/S

Processrapport

Udviklingen af serversadministrationsværktøjet ”Server Sentry”

Titelblad

Techcollege Aalborg,
Struervej 70,
9220 Aalborg

Elev:
Kevin Wallentin Carlsen

Firma:
Space Inventor A/S

Projekt:
Udviklingen af ”Server Sentry”

Uddannelse:
Datatekniker med speciale i programmering

Projektperiode:
18/01/2024 – 04/03/2024

Afleveringsdato:
04/03/2024

Fremlæggelsesdato:
11/03/2024

Vejledere:
Simon Hoxer Bønding

Læsevejledning

- ORM (Object Relational Mapping)
- API (Application Programming Interface)
- SQL (Structured Query Language)
- Repo ((GitHub) repository)
- DRF (Django-REST-Framework)

Forord

Denne procesrapport er skrevet i sammenhæng med det tværfaglige projekt på H5PD010124.
Denne rapport er en af to som tilhører projektet. Teknisk produktdokumentation forekommer i ”produktrapporten”

Procesrapporten dækker udviklingen af mit serveradministrationsværktøj kaldet "Server Sentry". Hermed indgår overvejelser vedrørende alternative teknologivalg.

Indholdsfortegnelse

Indledning
Det har fra starten været indtænkt at produktet skal implementeres på min server derhjemme, men samtidigt har det også været vigtigt, for mig, at implementationen er nem i flest mulige passende sammenhænge.

Den réele implementation skal som efter planen bruges af en vennegruppe på 5 mennesker, som skal have mulighed for lettere administration af vores spilservere, mens jeg selv er på arbejde eller ligeledes henholdt fra serveradministration.

Case beskrivelse
Problemformulering
Hvordan kan slutbrugere oprette og fjernadministrere spilservere på deres nemt tilgængelige hardware?

Afgrænsning
Jeg stiftede min kravspecifikation med flere nedprioriterede krav som, blandt andet, har som formål at lægge grundlag for fremtidig udvidelse af produktet efter projektets færdiggørelse.
Hermed er tiltænkt alle krav af prioritet 5, hvoraf følgende stadig mangler: push notifikationer, tilstrækkelig håndtering af brugerrettigheder og whitelist/blacklist.

Oprindeligt var det også tiltænkt at logging skulle kunne forgå vha. en tidsbaseret database (sandsynligvis Victoria Metrics), men sådan en væsentlig udvidelse af kompleksitet har jeg ikke vovet mig at introducere i projektet.

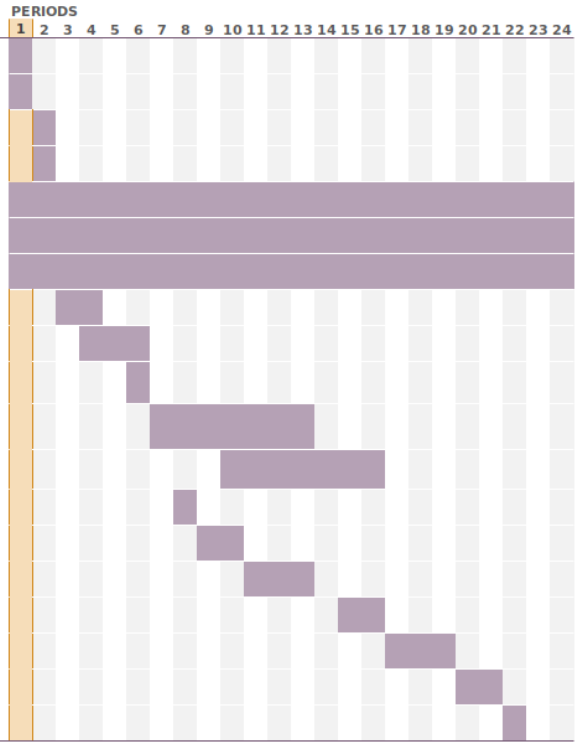
Projektplanlægning
Estimeret tidsplan
På den estimerede tidsplan forventes rapportskrivning og udviklingen af produktet parallelt.

Spilserver Administration

Enhed for tidsrum er angivet i dage.
Select a period to highlight at right. A legend describing the charting follows.

Period Highlight 1 | Plan Duration | Actual Start | % Complete | Actual (beyond plan)

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Case Beskrivelse	1	1	1	1	0 %
Problem formulering	1	1	1	1	0 %
Kravspecifikation	2	1	2	1	0 %
Estimeret Tidsplan	2	1	2	1	0 %
Real Tidsplan	1	24	1	24	0 %
Processrapport	1	24	1	24	0 %
Produktrapport	1	24	1	24	0 %
Opsætning af Docker/DB	3	2	3	2	0 %
WebApi opsætning	4	3	4	3	0 %
K1 - API til åbne/lukke Docker containere	6	1	6	1	0 %
K2 - Opsætning og forståelse af flutter	7	7	7	7	0 %
K2 - Simpel app til administration af servere	10	7	10	7	0 %
K3 – Logning	8	1	8	1	0 %
K4 – Visning af online spillere/servere	9	2	9	2	0 %
K5 - Live opdatering i app	11	3	11	3	0 %
K7 - Automatiske backups af save-filer	15	2	15	2	0 %
K8 – Web-frontend	17	3	17	3	0 %
K9 - Push notifikationer	20	2	20	2	0 %
K10 – Brugerrettigheder	22	1	22	1	0 %



Arbejdsfordeling

Hvordan har du fordelt arbejdet på programmet og rapporterne.
har du fyldt din tidsplan?
Er der noget du har brugt forlang tid på og hvordan har det påvirket din planlægning?

Metode- og teknologivalg

Produktet præsenteres af en frontend lavet med Flutter (Dart), som forbinder til en backend lavet med Django (Python). Databasen er på nuværende tidspunkt SQLite, men takket være Django's ORM er den nemt udskiftelig skulle der blive behov for det.

Projektets Dockerfiler (hermed også docker-compose.yml) har som formål at gøre det nemt at opsætte og afprøve produkterne.

Flutter

Flutter er et cross-platform frontend framework, hvormed den samme kode derfor kan præstere på: mobil, computere, web, mm.
Flutter håndterer derfor automatisk mange af de elementer nødvendige for skalering og præsentation på de forskellige platforme.
Alternativt kan kode udvikles til en specific platform, hvilket giver bedere mulighed for at udnytte platformens hardware og skræddersye udseende.

Indtil videre har jeg ikke mødt nogen frontend udviklingsværtøjer som jeg synes godt om.
Jeg er stor modstander af XML (eller lignende) til frontend, jeg oplever at det typisk knytter én til Visual Studio og dermed Windows.
Modsat set kan man beholde alt UI som kode, hvor jeg tidlige har brugt TKinter og Kivy (Python). Selv ser jeg størst potentiale med denne metodik.
Flutter udvikling forgår som den sidstnævnte, eksklusivt i kode.

Sjovt nok er Flutter alligevel stort set det stik modsatte af hvad jeg foretrækker inden for programmering:
- Meget af koden forgår med async/await, som kan siges at være ukompatibel men synkron programmering.
- Typisk instantierer man alle sine widgets som 'recursive' argumenter til retur værdigen. Det medfører en masse indlejrede paraneteser, af flere forskellige slags, som er svære at omstrukturere, trods hjælpen fra Visual Studio Code.

Alligevel er jeg åben for at vælge Flutter igen i fremtiden, grundet min store respekt for understøttelsen af mange platforme.

Django

Som backend framework bruger projektet Django, som er et Python web framework med fokus på hurtig udvikling med indbygget sikkerhed.
Django er dog større og mere kompleks end mange andre Python alternativer, såsom Flask, og ligner på mange måder Asp.Net Core (med Entity Framework) fra .Net verdenen.

Jeg har efterhånden brugt Django til en håndfuld projekter, og har derfor nemmere ved at forudsige kompleksiteterne dertil.
Selv sagt vil jeg også sige at jeg opbrugte min kvote på ukendte teknologier da jeg valgte Flutter som frontend, som viste sig at tage noget tid at lære.
Efterhånden kendetegner _jeg_ Django ved hvor nemt det er at lave sine modeller. Og med lidt viden om Django-REST-framework, er det også ret hurtigt at lave sit web-API.

Det primære formål med projektets Django backend er at tilbyde et web-API som muliggør at administrere spilserverne fra Flutter applikationen. Derfor skal backenden også tilbyde et fælles interface til diverse implementationer af spilservere.
Som den første konkrete spilklasse understøtter projektet servere til spillet Vintage Story, som i baggrunden kører vha. Docker-Compose. Men med Manager klassernes arvehierarki, kan udvikleren nemt udskifte hvordan spilserveren i virkeligheden køres.

Docker / Python(-on-whales)

Docker er et meget veletableret virtualiserings værktøj, hvortil mange spilservere derfor allerede understøttes.

Docker adskiller sig fra typiske virtualiserings værktøjer ved ikke at bruge virtuelle maskiner, i virkeligheden separerer Docker bare processer ved at bruge namespaces. Det giver forholdsvis lave omkostninger til virtualisering.
Samtidig gør Docker det også nemt, og systematisk, at 'Dockerisere' sine egne tjenester med deres 'Dockerfiler' og Docker-Compose.
Kombinerer man derfor Docker med en grafisk brugerflade, såsom Portainer, opnår man allerede meget af problemformuleringen til dette projekt.

Docker i sig selv er meget generisk, hvilket også afspejler sig i brugerfladerne dertil. Derfor ser jeg stadig hjemmel i en skræddersyet løsning som dette projekt forsøger at tilbyde.

Når man 'Dockeriserer' en tjeneste, laver man først et "image" med din "Dockerfile". Her fortæller man løser man sin tjenestes afhængerigheder ved at vælge hvilken virtualiseret platform/OS den skal køre på, hvorefter man yderligere tilpasser den til sit behov.
Herefter fortæller man også hvordan sin tjeneste skal starte på denne platform.
Et "image" i sig selv er bare en opskrift, når den virkeliggøres laver man en "container" (som man som programmør skulle kunne beskrive som en 'instans')
Tjenesten synliggøres for omverdenen når man vælger

Docker kan udvides med 'Docker Compose', som gør det nemt at administrere flere "Containere" som én helhed.

Jeg har tidligere brugt Dockers officielle Python SDK (API) med et godt indtryk, og planlagde derfor også at bruge det her.
Desværre fandt jeg ret tidligt ud af at det ikke understøtter docker-compose.
Jeg fortrækker docker-compose da det tillader mig at starte næsten alle servere med minimale ændringer til argumenter, så jeg ville nødigt undvære det.
Jeg søgte mig derfor frem til Python-on-whales, som er et uofficielt Docker/Python API med indbygget understøttelse. Pakken fungerer således at man instantierer en klient, ved at pege den i retning af en docker-compose.yml fil.
Herpå kan man derefter kalde metoderne: .up(), .down(), .start(), .stop() osv. præcis som man kender kommandoerne fra terminal.
Den eneste ulempe jeg oplevede ved Python-on-whales er det faktum

Database

Til database bruger jeg Django's ORM (Object Relational Mapping), som understøtter en række relationelle databaser. Heraf bruger jeg SQLite, som ikke kræver endnu en Docker container. Havde jeg haft større krav til databasen, ville jeg i stedet have valgt PostgreSQL af flere grunde:

- For at kunne installere Django-Polymorphism pakken, som ville tillade mig at gøre min GameServer model abstrakt, men fortsat lave forspørgeler derpå for at få konkrete nedarvede klasser til specifikke spil.

Hermed ville jeg kunne undgå mine 'Manager' klasser (og arvehierarkiet dertil).
Sagt således lyder Django-Polymorphism som et meget fornuftigt valg, men tager man i begtragtning at det ville koste en ekstra SQL JOIN per konkrete spilklasse, samt en ekstra Docker container, har pakken alligevel væsentlige ulemper.

- Jeg har tidligere Dockeriseret en PostgreSQL database til vores H3 Trello lignende projekt, og ved derfor at det nemt kan gøres med nogle enkelte ændringer til Django og docker-compose.yml filen.

Forespørgsler til relationelle databaser skrives i sproget SQL, som kan have mindre variationer afhængigt af database brugt. Disse variationer udviskes ved brugen af en ORM, som giver et abstraktionslag til databasen. Dette tillader udviklingeren at holde alle forespørgsler til databasen i samme sprog som resten af programmet og dets kode. Abstraktionslaget gør det typisk også nemt at udskifte hvilken database som bruges. Den fordel vil jeg gøre brug af, skulle jeg for behov for PostgreSQL.
Som ulempe er ORMe typisk langsommere end skræddersyede SQL sætninger.

Væsentlige elementer fra produktrapporten
Udviklingen af produktet har lært mig meget om Flutter, og givet mig stor respekt for dets multi-platform kompetencer. Det at jeg kan starte samme program som både en web-applikatation og hjemmehørende Linux applikation, og se live-opdateringer ske mellem dem, har været en stor successoplevelse.

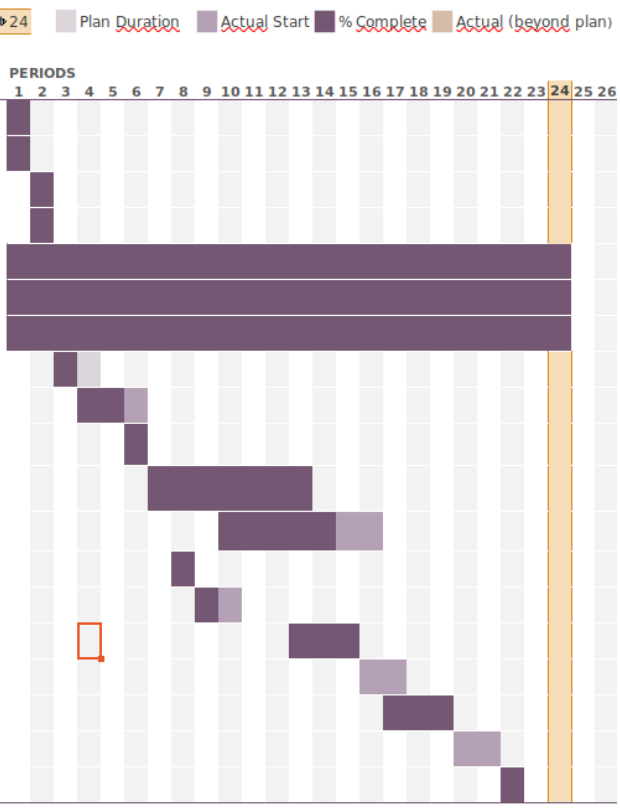
Realiseret tidsplan

Den realiserede tidsplan viser at visse punkter blev nedprioriteret og undladt.

Spilserver Administration

Select a period to highlight at right. A legend describing the charting follows. Period Highlight 24 Plan Duration Actual Start % Complete Actual (beyond plan)

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Case Beskrivelse	1	1	1	1	100 %
Problem formulering	1	1	1	1	100 %
Kravspecifikation	2	1	2	1	100 %
Estimeret Tidsplan	2	1	2	1	100 %
Real Tidsplan	1	24	1	24	100 %
Processrapport	1	24	1	24	100 %
Produktrapport	1	24	1	24	100 %
Opsætning af Docker/DB	3	2	3	1	100 %
WebApi opsætning	4	3	4	3	95 %
K1 - API til åbne/lukke Docker containere	6	1	6	1	100 %
K2 - Opsætning og forståelse af flutter	7	7	7	7	100 %
K2 - Simpel app til administration af servere	10	7	10	7	80 %
K3 – Logning	8	1	8	1	75 %
K4 – Visning af online spillere/servere	9	2	9	2	75 %
K5 - Live opdatering i app	13	3	13	3	100 %
K7 - Automatiske backups af save-filer	16	2	16	2	0 %
K8 – Web-frontend	17	3	17	3	100 %
K9 - Push notifikationer	20	2	20	2	0 %
K10 – Brugerrettigheder	22	1	22	1	70 %



Konklusion

Med mit program har jeg målrettet serveradministration til specifikke spil og implementationer. Herved har jeg kunne forenkle og forene opsætningen samt administration deraf.
Med denne målsætning kræves der mindre arbejde fra brugere, men mere fra udvikleren (per spil). Dog lettes det af kræfterne lagt i arvehierarkiet, hvor metoder og registering (af spil/klasser) kan håndteres af superklasser.

I virkeligheden kan spillere, samt deres spil, stille vidt forskellige krav til hvad serveradministration kræver.
Til mit projekt oplevede jeg ret tidligt nødvendigheden ved at håndtere versionering af serverene.
Heldigvis var denne problematik nem at løse med min GitHubVersionedManager klasse, som tildeler commits til spilversioner på et givent repo.
I fremtiden håber jeg også på at finde en god løsning til modding (tilføjelse af bruger-skabte udvidelser) af servere, da det er relevant for hvordan jeg selv kommer til at bruge

produktet. Dog ligger dette krav uden for min kravspecifikation, så manglen er ikke væsentlig her.

Mit største problem burde jeg have undersøgt, inden jeg startede projektet. Jeg lærte undervejs at 'URL' paths er HTTP specifikke, og derfor ikke kan bruges til viderestilling af TCP/UDP pakker.
Alternativt kunne jeg have brugt subdomains. Men da det ville have stilt større krav til deployment of HTTPS certifikater, besluttede jeg mig at det kostede for meget arbejdstid uden for kravspecifikationen.
I stedet endte jeg med den simple løsning, at eksponere spilservernes porte.
Som fordel er det simpelt; Docker tillader brugeren af styre hvilken container en given port skal viderestilles til.
Derfor ordnede jeg således at hver instans af en server finder den mindst inkrementeret port, som ikke er brugt af andre servere. Det er væsentligt da flere servere til samme spil ellers ville have brugt samme port.
Som ulempe stiller det krav til at en lang række potentielle porte skal åbnes på opsætterens/administratorens router, hvilket jeg havde håbet at undgå.

Trods jeg ikke nåede at tilpasse Flutter applikationen til at understøtte oprettelse af nye server, kan Django's autogenerede administratorside vise hvor nemt det er.

Diskussion
Dog jeg er tilfreds med kravspecifikationens nedprioriterede krav, har jeg følt at de 11 krav jeg har haft er for få, når jeg tillader mig selv __ikke__ at nå alle.

Med andre ord har mine højt prioriterede krav nok været for brede. Det er en situation som jeg vil forsøge at undgå til svendeprøven.

Jeg har længe forsøgt, eller i hvert fald overvejet, at fortsætte arbejdsmetodikken vi brugte på GF1. Her blev alle opgaver nedbrudt til max 5 dages længde, og dokumenteret med arbejdsplaner. Efterfulgt GF1 har min plan været at samle disse arbejdsplaner (til ét givent projekt) ved at skrive en rapport, som man løbende kan udvides efter projektets fremgang.
Jeg kan godt lide tanken om at dokumentere vilkårlige projekter til det fremtidige jeg, eller andre som ellers kan interessere sig for dem.
Modsat set har denne rapport givet mig større respekt for tiden det kræver, dog har en frivillig rapport ikke samme krav om længde og indhold.
Jeg holder fast i tanken og håber jeg får tid til flere rapporter i fremtiden.

Dette tverefaglige projekt har givet mig meget spekulation vedrørende omfanget til produktet til den kommende svendeprøve. De få obligatoriske krav jeg har stilt mig selv på kravspecifikationen, får mig til at tro at mit produkt er for småt, og dermed ikke passende til svendeprøven. Men tidsmæssigt tror jeg omfanget passer fint, nu hvor jeg har oplevet at skrive størstedelen af rapporterne dertil.

Jeg vil forsøge at være mere produktiv til svendeprøven, så jeg kan øge omfanget af produktet.
Hertil tror jeg at det er til megen hjælp at vi primært kommer kunne arbejde hjemmefra. På nuværende tidspunkt vil det spare mig 3 timer i transport dagligt, som jeg i stedet kan lægge på tidsplanen.
Motivationmæssigt oplever jeg dage hvor jeg opnår meget mere hjemmefra, men samtidig også dage som er fyldt med overspringshandlinger.
Som helhed er jeg næsten sikker på at jeg vil være mere produktiv til svendeprøven. På 2 dages hjemmearbejde fik jeg skrevet mellem 14000-17000 anslag til disse rapporter, hvor samme antal har varet projekttiden brugt på skolen.

Referencer

Inspiration til “ServerEvent Enum” som “TextChoices felt” I Django applikationen:
https://stackoverflow.com/questions/54802616/how-can-one-use-enums-as-a-choice-field-in-a-django-model
Her kunne “django-enum” pakken også være brugt

https://stackoverflow.com/questions/60869395/python-covert-enum-to-django-models-charfield-choices-tuple?noredirect=1&lq=1

Docker SDK i container
https://stackoverflow.com/questions/60054378/i-want-to-use-docker-sdk-inside-a-running-docker-container-and-want-to-build-an

At GameServer.server_name skal være ufølsomme for store og små bogstaver og unik:
https://stackoverflow.com/questions/7773341/case-insensitive-unique-model-fields-in-django

Flutter mørk tilstand:
https://stackoverflow.com/questions/60232070/how-to-implement-dark-mode-and-light-mode-in-flutter

Brug og opsætning af Django Channels til websockets:
https://channels.readthedocs.io/en/latest/installation.html

Brug og opsætning af Flutter/Dart channels:
https://pub.dev/packages/web_socket_channel/example