

DEEP FLIGHT

Eksamensprojekt i kurset '62597 Backendudvikling, drift og distribuerede systemer F20'

Gruppe 0



Rasmus sander Larsen
s185097



Erland Turmi
s132962



Malte B. Pedersen
s185139



Andreas B.G. Jensen
s130016

Indholdsfortegnelse

Indholdsfortegnelse	1
OBS:	3
Indledning	4
Spillet	4
Systemet	4
Projektstruktur	4
Links:	4
Arbejdsfordeling	5
Kravspecifikation	6
Systemets struktur (Deployment Diagram)	7
Backend Komponenter	7
Deployment	7
Implementering	9
Spillet: C#, RestSharp	9
GameAPI: Java, REST	9
UserAPI: Java, REST	10
UniverseUpdater: Python	12
GameDatabase: MongoDB	12
Hjemmeside	14
Monitorering	15
Test	16
UserAPI	16
Overordnet test af API	17
Diskussion	17
Stabilitet i Universe Updater	17
Deployment	18
Brug af Prometheus	18
Håndtering af Track Data	18
Konklusion	18
Bilag A: Skærbilleder af spillet	21
Bilag B: Teoriafsnt	23

OBS:

Selve spil applikationen er primært udviklet af Malte Pedersen (s185139) i sammenhæng med en eksamensaflevering i kurset '64213 Avanceret Objektorienteret Programmering med C# og .NET' i foråret 2020.'. Derfor er kun koden i pakken '*source.network*' i kildekoden, der indgår som del af afleveringen i dette kursus.

Indledning

Spillet

Deep Flight er et 2D spil udviklet i C#, hvor spillerne dystet online om at flyve hurtigst igennem spillets baner med sit rumskib. Banerne bliver tilfældigt generet og der dystes i en 'runde' af 4 baner af gangen over 24 timer. Når runden er slut bliver spillerne rangeres ift. deres tider på rundens baner.

Systemet

Det distribuerede system gemmer, opdaterer og udbyder adgang til spillets online data, herunder brugere, runder, baner og tider. Det er bestående af 7 komponenter:

- Spillet
- Hjemmeside
- Game API
- User API
- Universe Updater
- Database
- Prometheus (monitorering)

Projektstruktur

Projektet består af et underprojekt for hver komponent i systemet:

- DeepFlightGame: C#, Visual Studio 2019 (kræver MonoGame)
- WebSite: React
- GameAPI: Java, IntelliJ + Maven
- UserAPI: Java, IntelliJ + Maven
- UniverseUpdater: Python, PyCharm
- TrackGenerator: C, CLion + CMake. Denne indeholder ikke relevant backendkode og kan ses om en "black box".

Links:

GitHub:

- Overordnet: <https://github.com/maltebp/DeepFlight>
- Spillet: <https://github.com/maltebp/DeepFlightGame> (bemærk at der arbejdes videre på projektet efter aflevering. For at se en version ved afleveringstidspunktet, se tagget 'v1.1': <https://github.com/maltebp/DeepFlightGame/tree/v1.1>)

Komponenter:

- Hjemmeside: <http://maltebp.dk/>
- UserAPI: <http://maltebp.dk:7000/>
- GameAPI: <http://maltebp.dk:10000/gameapi/>
- Prometheus: <http://dist.saluton.dk:13018/graph>
- UptimeRobot: <https://stats.uptimerobot.com/M7XpNh5x0q>

Arbejdsfordeling

	Malte	Erlend	Rasmus	Andreas
Hjemmeside		X		
GameAPI	X	X	X	X
UserAPI		X		X
Database	X		X	X
Universe Updater	X			X
Prometheus				X
Game API TUI			X	
Spillet (netværkskommunikation)	X			
Deployment (Docker)	X	X		X

Kravspecifikation

Flere bruger skal kunne spille det samme map på samme tid.	Opfyldt
Der skal genereres en runde som der spilles pr. dag	Opfyldt
Spil/bruger informationer skal lagres i en database	Opfyldt
Deepflight spillet skal kunne downloades fra en hjemmesiden	Opfyldt
Der skal genereres forskellige tracks	Opfyldt
En bruger der er logget ind skal have udleveret et token	Opfyldt
Der skal indgå brugervalidering på Game API'en	Opfyldt

Systemets struktur (Deployment Diagram)

Systemets struktur, samt kommunikationsveje og former (protokoller) kan ses af deployment diagrammet herunder.

Backend Komponenter

Systemets backend består af 3 primære komponenter:

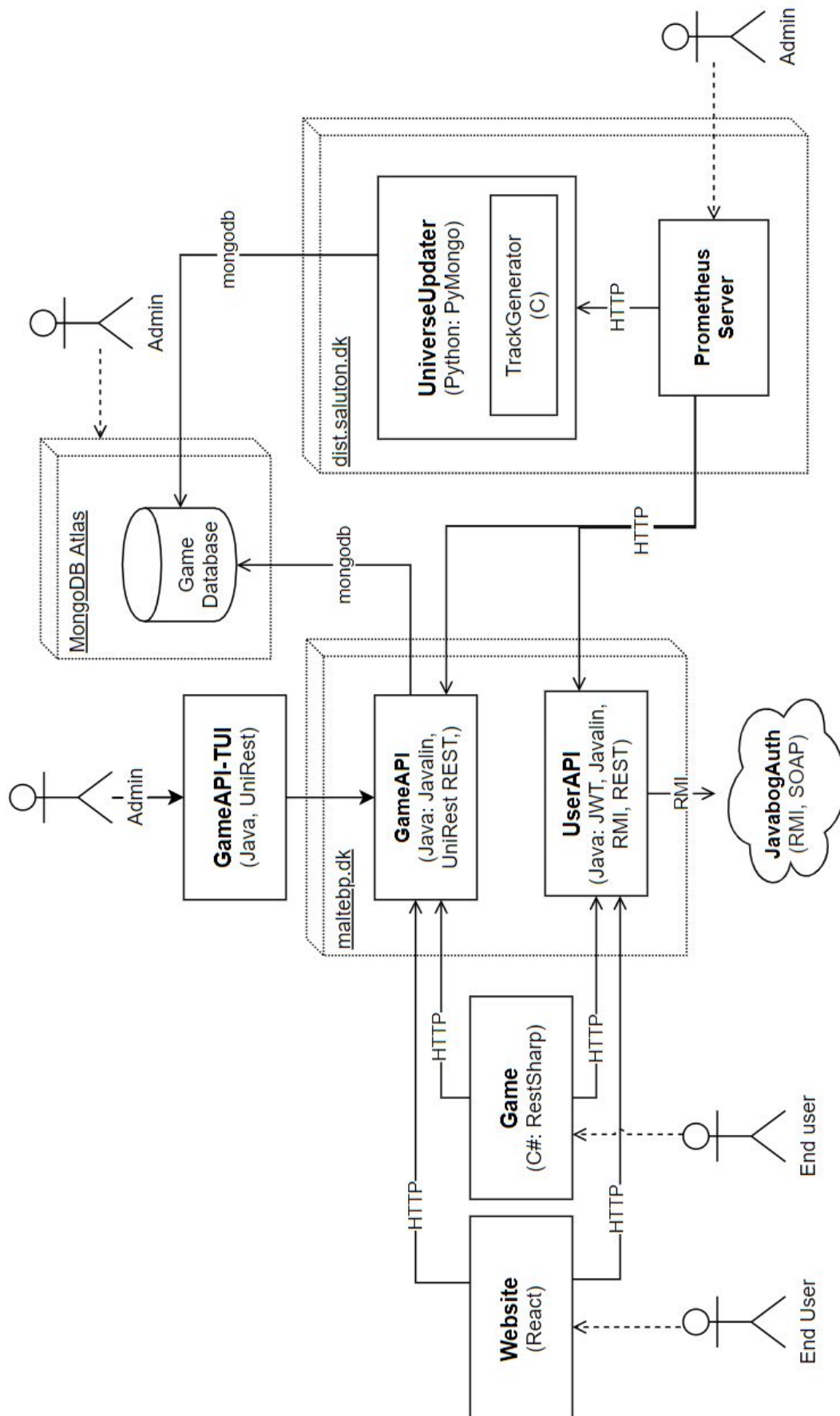
- *Game API*: Spillets primære API, der udbyder adgang til spillets resurser, herunder runder, baner, brugere og rankings. Flere ressourcer kræver verificering af brugeren, og den kontakter derfor UserAPI'en
- *User API*: Modulet der håndtere authentication af brugere og fungerer som et interface mellem JavaBogs brugerautorisation, vores egne prædefinerede brugere, og systemets brugergrænseflader (spillet og hjemmesiden).
- *Universe Updater*: Står for at genere nye runder, herunder rundens baner, samt at opdatere brugeres ranking efter en runde er afsluttet.

Herudover eksisterer også en Prometheus klient, der blot hentes som et Dockerimage, der står for at indsamle informationer og de forskellige komponente og udbyde dem via en webserver.

Deployment

Alle backend komponenter bliver deployet i hver deres Docker container. Databasen er hostet hos MongoDB Atlas, der sikrer automatisk backup, samt udbyder praktisk brugergrænseflade ift. administrering af databasen.

GameAPI og UserAPI er deployet på Maltes egen server cloud server (Amazon EC2) under domænet "maltebp.dk", mens UniverseUpdateren og Prometheus-serveren er deployet på *dist.saluton.dk*. Årsagen til at Universe Updateren ikke er deployet på maltebp.dk skyldes at dets Python image fylder for meget..



Implementering

Spillet: C#, RestSharp

Relevant kode: Alle klasser i pakken `'source.network'`

*NB: Ikke hele spillets kildekode er en del af dette projekt - se bemærkning ved 'Indledning'.
Den relevante kode ligger i pakken `'source.network'`.*

Spillet skal kontakte både GameAPI'en og UserAPI'en, og her benyttes biblioteket *RestSharp*, som gør det nemt at implementere kald til en REST API.

Exceptions

Da spillet er det slutbrugeren primært benytter, er det vigtigt at håndtere forskellige fejl scenarier, og kommunikere dem videre. Derfor er opstillet en række Exceptions, der kastes i afhængigt af resultatet fra API'erne, såsom fejl i forbindelsen og manglende authentication.

Asynkront kald

Da hoved tråden i programmet skal tegne GUI'en og spillet, er det nødvendigt at netværkskommunikationen foregår asynkront. Derfor benyttes C#'s `'Task'` og `'await'` keyword, der gør det muligt at starter en stykke kode som baggrundstråd.

GameAPI: Java, REST

Relevant Kode: Hele modulet

Netværksteknologier

- Javalin (Jetty) som server
- UniRest

Kommunikationsprotokoller

- HTTP(TCP)

REST - endpoints:

Alle endpoints starter på `/gameapi/...`

- `/round/current`, `round/previous`
- `/planet/<id>`, `/planet/all`
- `/rankings/universal`
- `/track/<id>`, `track/<id>/trackdata`, `track/<id>/times/<userid>` (POST)
- `/user/<id>`, `/user/<id>/private`, `/user/all`

Brugerautorisering

Der anvendes brugerautorisation for tre resurser på GameAPI'en: når der hentes en privat bruger (benyttes til login), når der uploades en ny track tid, samt når spillet downloades. Når én af de nævnte resurser kaldes, kontrollerer der for om requesten har en "Authentication" header, der indeholder en token. Denne videresendes til UserAPI'en, der valideres der den. For at forhindre at en bruger manuelt kan poste en ønskværdig tid på en bane i gennem GameAPI'en, indeholder requesten en "updateKey" der skal matche en tilsvarende på GameAPI'en. Det er hensigten at updateKey'en for så vidt holdes skjult for klienten.

Track Data

En bane består af to dele: metadata (navn, planet osv), samt 'track data' (hvordan banen skal tegnes). Da Track dataen er relativt store filer (ca 2 mb), eksisterer de som en separat resurse, man eksplicit skal bede om (via. resursen `/track/<trackid>/trackdata`). Dette gør det f.eks. muligt at se de eksisterende baner uden at skal hente dataen for dem.

UserAPI: Java, REST

Relevant kode: Hele modulet

Brugervalidering:

Fra UserAPI'en er der er der oprettet forbindelse til javabog.dk autorisering modul vha. RMI (Remote Method Invocation).

Server:

- Javalin/jetty

Kommunikations protokoller

- HTTP(TCP)

REST -tendpoints:

- "/login"
- "/jwt/changeLogin"
- "/jwt/exchangeUser"
- "/loginRequest": En respons besked, der returneres hvis at en request ikke slipper igennem auth filteret.

Kommunikation med Javabog.dk

- RMI (Remote Method Invocation)

AuthFilter:

```
app.before("/jwt/*", ctx ->
    Optional<DecodedJWT> decodedJWT =
        JavalinJWT.getTokenFromHeader(ctx)
            .flatMap(JWTHandler.provider::validateToken);
    if (!decodedJWT.isPresent()) {
        System.out.println(source+": No/or altered token");

        //Redirection to a responsemessage, providing with information
        on how to post a login request.
        ctx.redirect("/loginRequest", 302);
    }
});
```

Alle requests (bortset fra "/login") der sendes til UserAPI bliver filtreret af et authentication filter. I dette filter vil en given request bliver undersøgt for, om der findes en 'Authentication' header, der indeholder en JWT. Hvis denne header ikke findes eller ikke kan valideres, vil klienten bliver nægtet adgang med en statuskode 401.

Som det fremgår af Auth Filteret bliver tokens claim ikke pakket ud. Dette skyldes at denne API ikke har implementeret nogen funktionalitet, der vil gøre brug af brugerens informationer.

Brugerens informationer skal anvendes på Game API og det er derfor valgt at en Game API'en kan udveksle en gyldig token for brugerinformationer. Dette sker vi endpointet "/jwt/exchangeUser"

Token

En token har en varighed på 12 timer, for at sikre at eventuelt stjålne tokens ikke kan bruges for evigt. Dog er en høj varighed nødvendig, da det er u hensigtsmæssigt hvis man bliver logget ud mens man spiller, da en tokens benyttes for at uploade nye tider.

UniverseUpdater: Python

Relevant kode: Pakken 'database', samt filen 'universeupdater.py' ("main" klassen)

Universe Updateren skal genere baner og "rate" spillerne ud fra deres tider på rundens baner. Dette er noget der kun skal ske hver én gang i døgnet, da en runde varer 24 timer.

Opdateringsprocessen

Da en runde varer 24 timer, er det ikke nødvendigt at Universe Updateren kører hele tiden. Derfor tjekker den kun om en opdatering er nødvendig hvert 30 sekund, og resten af tåden sættes tråden til at sove, hvilket sparer resurser på serveren.

Selve programmet der generere banerne ("TrackGenerator.exe") er en generisk program til at generere baner til spillet, og startes som en "subprocess" af Universe Updateren, når der skal generes baner.

Sikring mod nedbrud

Informationen der styrer om universet skal opdateres ligger *ikke* i programmet, men er i stedet inkorporeret i database. Dette betyder at selvom Universe Updateren crasher, startes runden ikke forfra.

Derudover, generes der altid én runde ud i fremtiden, så der programmet kan være gået ned i minimum 24 timer uden at universet går i stå.

Transaction

Et nyt Track skal indsættes i databasen i to dele: metadataen og trackdataen. For at sikre at den ene ikke kan eksistere uden den anden, benyttes "transactions" for at indsætte dem, hvilket sikrer at enten går det hele i gennem ellers går intet i gennem.

GameDatabase: MongoDB

Det er valgt at anvende den skemaløs dokument database; MongoDB, eftersom at den er lettere at arbejde med end en relationsdatabase og fordi at den meget let kan hostes hos Atlas der er en Cloud-hosted MongoDB service hos AWS.

Alle informationer der vedrører rating af brugere er en essentiel del af Deep Flight's game play, hvor det jo handler om at brugere konkurrere på tid. Det er derfor valgt at outsource forvaltningen af databasen til en tredjepart, som vi mener, kan sikre konsistent data bedre end vi kan på nuværende tidspunkt.

Databasen består af følgende collections og documents:

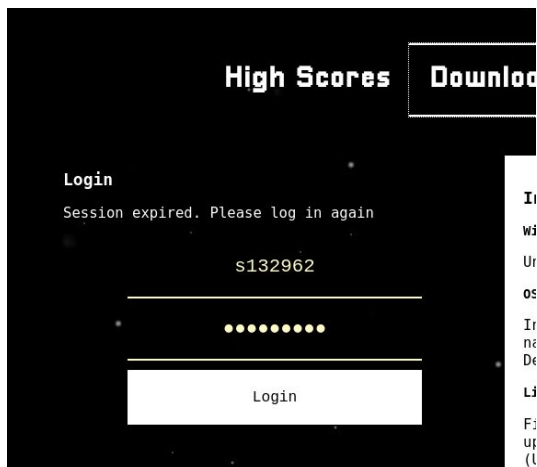
Collections	Documents
planets	<code>_id</code> <code>name</code> <code>color</code> : Array <code>lengthFactor</code> <code>curveFactor</code> <code>stretchFactor</code> <code>widthFactor</code> <code>widthNoiseFactor</code>
rounds	<code>_id</code> <code>trackIds</code> : Array <code>roundNumber</code> <code>startDate</code> <code>endDate</code> <code>rankings</code> : Object
tracks	<code>_id</code> <code>name</code> <code>planetId</code> <code>seed</code> <code>times</code> : Object
trackdata	<code>_id</code> <code>data</code> : Binar
users	<code>_id</code> <code>username</code> <code>rank</code> <code>rating</code>

Hjemmeside

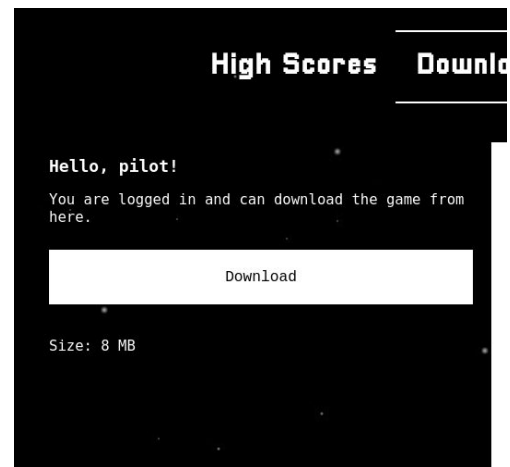
Universal ratings (top 5)			Last round ratings (top 5)		
1	kingkong	3.5	1	s132962	9.9
2	dennis	2.9	2	s343434	4.0
3	hogrid123	1.7	3	Team Liquid	2.3
4	admin42	1.5	4	s339988	2.1
5	s132952	1.3	5	Astralis	0.0

Hjemmesiden er lavet i react, og bruger både GameAPI og UserAPI. Forsiden henter data fra GameAPI og viser rekorder for spillet, som vist over. De fleste 'game stats' fra spillet vises, men ikke de personlige.

Login-siden henter et JWT-token, og gemmer det i browserens local storage. Hvis token er udløbet, vises login-menuen med en besked om at token er udløbet. Hvis man har et gyldig token, vil siden vise download-menuen. Gyldigheden er sat til 12 timer af hensyn til spillet. Download-siden har også en kort installationsguide til forskellige operativsystemer, der aflæser browserens nuværende operativsystem.

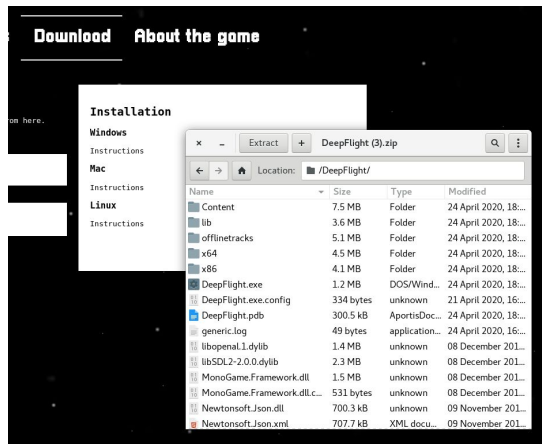


Login



Download

Så snart man har et gyldig token, kan man downloade spillet som en zip-fil. Der kommer en download-prompt fra browseren. Hvis for eksempel token er udløbet i mellemtiden, eller der er problemer med forbindelsen får brugeren besked om, at download fejlede.



Når et download sættes i gang, sker det ved at JWT sendes til GameAPI, der verificerer den via UserAPI (hvor nøglen ligger). HTTP-responsen indeholder en binær fil som stream, der åbnes via en midlertidig link genereret i JavaScript. Metoden er testet i Firefox og Chrome.

Hjemmesiden har også en 'about'-side med skærbilleder og en kort introduktion af spillet og projektet.

Hjemmesiden er deployed i en docker container med nginx på en AWS-server.

maltebbp.dk

TUI

TUI'en er et kommandolinje program, hvorfra 7 af GameAPI'et REST endpoints kaldes og brugeren ser resultatet af kaldene. TUI'en er skrevet i java og der er gjort brug af følgende dependencies; Unirest og Jackson. Unirest benyttes til at foretage de udvalgte 7 kald til vores GameAPI. Jackson bruges til at parse resultatet af REST kaldet fra json-streng til java klasser.

Alle de 7 udvalgte REST kald er "get"-metoder, som enten kræver ekstra information fra brugeren eller som blot kan kaldes. Dette er valgt fordi TUI'en er tiltænkt som et redskab, som hurtigt og enkelt skal give adgang til nogle af nøgle informationerne i spillet.

Monitorering

Systemet bliver monitoreret med Prometheus. UserAPI, GameAPI og UniverseUpdater udstiller en række statistikker (metrics), som Prometheus kan hente.

Som metric er der anvendt en standard metric, der trækker informationer ud fra 'Jetty' vha en StatisticsHandler. Metrics udstilles via en HttpServer der kan tilgås via en anden port end den javalin lytter på.

UniverseUpdater service udstiller ingen API og det er derfor valgt blot at starte en Prometheus client når servicen starter op. Der er ikke implementeret nogen metric eftersom at det blot ønskes at se om den er oppe.

Figuren nedenfor viser der de service der bliver scrapet af Prometheus.

Targets

All Unhealthy					
Game_API (1/1 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://www.maltebp.dk:10001/metrics	UP	instance="www.maltebp.dk:10001" job="Game_API"	454ms ago	228.4ms	
Prometheus (1/1 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://dist.saluton.dk:13018/metrics	UP	instance="dist.saluton.dk:13018" job="Prometheus"	3.941s ago	11.32ms	
Universeupdater (1/1 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://dist.saluton.dk:13216/metrics	UP	instance="dist.saluton.dk:13216" job="Universeupdater"	9.275s ago	8.321ms	
UserAPI (1/1 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://www.maltebp.dk:7001/metrics	UP	instance="www.maltebp.dk:7001" job="UserAPI"	5.984s ago	221.6ms	

Hvis maltebp.dk serveren går ned vil det være muligt at kunne identificere dette eftersom Prometheus er deployet på dist.saluton.dk serveren.

Uptime Robot

Uptime robot monitorere lige så opptiden på alle komponenter. Hvis begge servere går ned vil vi blive informeret herom.

Linket nedenfor viser dashboard for opptiden af deploys.

- <https://stats.uptimerobot.com/M7XpNh5x0q>

Test

GameAPI

Til at teste GameAPI'et er der lavet en indirekte test, hvilket vil sige at en række af DatabaseDAO'ets metoder testet via junit tests, i stedet for at selve REST API'et er testet. DatabaseDAO klassen indeholder de metoder som hvert REST endpoint benytter sig af. Testen af GameAPI'et, tester udvalgt nøglefunktioner i DatabaseDAO.

Der er testes følgende metoder:

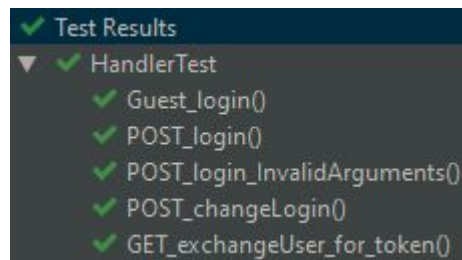
<ul style="list-style-type: none">- getPreviousRound- getCurrentRound- addUser- getUser- getPlanet- updateTrackTime	<div>▼ ✓ TestDatabaseDAO 341 ms<ul style="list-style-type: none">✓ getPreviousRound 97 ms✓ getCurrentRound 29 ms✓ addUser 95 ms✓ getUser 65 ms✓ getPlanet 25 ms✓ updateTrackTime 30 ms</div>
--	--

UserAPI

Til at teste UserAPI er alle endpointenes funktioner givet til en handler der kan kaldes fra en test suite. Når der testes startes der en server, hvor kun det endpoint der testes stilles til rådighed.

Testene på UserAPI, går primært ud på at teste de responses der finder sted under forskellige scenarier. Der er testes følgende metoder:

- POST_Login()
- POST_changeLogin()
- POST_changeLogin()
- GET_exchangeUser_for_token()
- Guest_login()



```
✓ Test Results
▼ ✓ HandlerTest
  ✓ Guest_login()
  ✓ POST_login()
  ✓ POST_login_InvalidArguments()
  ✓ POST_changeLogin()
  ✓ GET_exchangeUser_for_token()
```

Overordnet test af API

Postman er benyttet til at opstille en test suite, der automatisk tester begge API'er, og sikrer at systemet er oppe at køre. Der er opstillet en collection af requests, samt testscripts, der er grupperet i nogle foldere og tester følgende:

- *Authentication Test*: Tester authentication processen ved først at kontakte UserAPI'en og derefter GameAPI'en
- *Post Time Test*: Tester opdateringen af brugers 'track time'
- *Game API Test*: Generel test af størstedelen af GameAPI'ens endpoints

Test *collection* kan findes i github repoet her:

https://github.com/maltebp/DeepFlight/blob/master/DeepFlight_PostMan_Tests.postman_collection.json

og kan importeres i Postman og køres vha. *Collection Runneren*.

Diskussion

Stabilitet i Universe Updater

Fra et spillemæssigt synspunkt foretager Universe Updateren nogen meget vigtige opdateringer i databasen. På trods af dette er applikationen ikke særligt stabilt, og uventede nedbrud kan betyde at der opstår inkonsistent data. Dette ses f.eks. under rangeringen af en runde, hvor runden kan blive rangeren, men den universelle ranking ikke bliver opdateret. Samtidigt kan den uploade baner, uden at runden når at blive lavet hvis applikationen skulle fejle, hvilket betyder baner kan eksisterer uden at blive brugt.

Til at løse disse problemer, kunne der været gjort yderligere brug af *transaktioner* i til at skrive til databasen. Dette ville dog kræve en fundamental omstrukturering af Universe Updateren som der desværre ikke var tid til i denne omgang.

Der bliver ikke håndteret exceptions i Universe Updateren eftersom at vi gerne vil se at applikationen stopper, hvis der sker en fejl. På sigt skal dette håndteres men på nuværende tidspunkt finder vi det hensigtsmæssigt at applikationen går ned ved eventuelle fejl.

Deployment

Vi ønsker helst at spillet skal kunne spilles 24/7, 365 dage om året. Hvis dist.saluton.dk går ned, kan spillet stadig spilles, da Universe Updateren kun behøver at være aktiv hver 24. time. Dette er dog ikke tilfældet med serveren maltebp.dk. Hvis den går ned er der hverken muligt logge ind eller hente- /og eller uploade highscores.

På nuværende tidspunkt bliver skal serverne startes manuelt, hvilket vil sige at de potentielt kan være nede i længere tid (selvom serveren blot genstartede). Her kunne det være brugbart at implementere automatisk opstartning af containeren i tilfældet af at serveren går ned.

Brug af Prometheus

Der er på nuværende tidspunkt ikke lavet nogen vurdering af, hvilket data prometheus skal samle ind. For fremtiden ville det være spændende at man eks kunne se, hvor mange tokens der var udstedt og hvor man spillere der er logget ind på GameAPI'en.

Der bliver ikke indsamlet nogen information om docker miljøet som de forskellige applikationer befinder sig i. Det ville være hensigtsmæssigt at indsamle disse informationer så vi kan få kendskab til, hvor belastet docker containerne er.

Håndtering af Track Data

Dataen der bruges af spillet til at vise banen, er skræddersyede binære filer (navngivet .dftbd). Disse filer bliver lagt op på databasen som Base64. Dette er ikke hensigtsmæssigt, da databasen er hostet i Frankfurt, mens GameAPI'en ligger på maltebp.dk der er hostet i USA. Det kan derfor snildt tage 8 sekunder at hente en lille bane på 2 megabyte, hvilket kan være problematisk. For fremtiden kan det være en fordel at hoste filerne på en dedikeret filserver, som brugeren kan hente track dataen direkte fra.

Konklusion

Dette projekt omhandler et distribueret system der er opstillet i DTU kurset; 62597 - *"Backend development, operations and distributed Systems F20"*.

I projektet er det lavet et distribueret system, der kan understøtte en online udgave af spillet Deep Flight.

Det endelige projekt indeholder følgende:

- Et UserAPI der fungere som et interface imellem javabog.dk's brugerautoriseringsmodul og understøtter JWT.
- Et GameAPI der understøtter en online version af spillet "*DeepFlight*", hvor brugere dyster på tid. Styringen af spillets online udgave finder sted backend vha. "Universe Updater"en der generere runder, fremkalder tracks og opdatere spillernes ranks.
- Alle spil informationer bliver lagret i en tredjeparts database.

Det distribuerede system, er deployet i alt 3 servere:

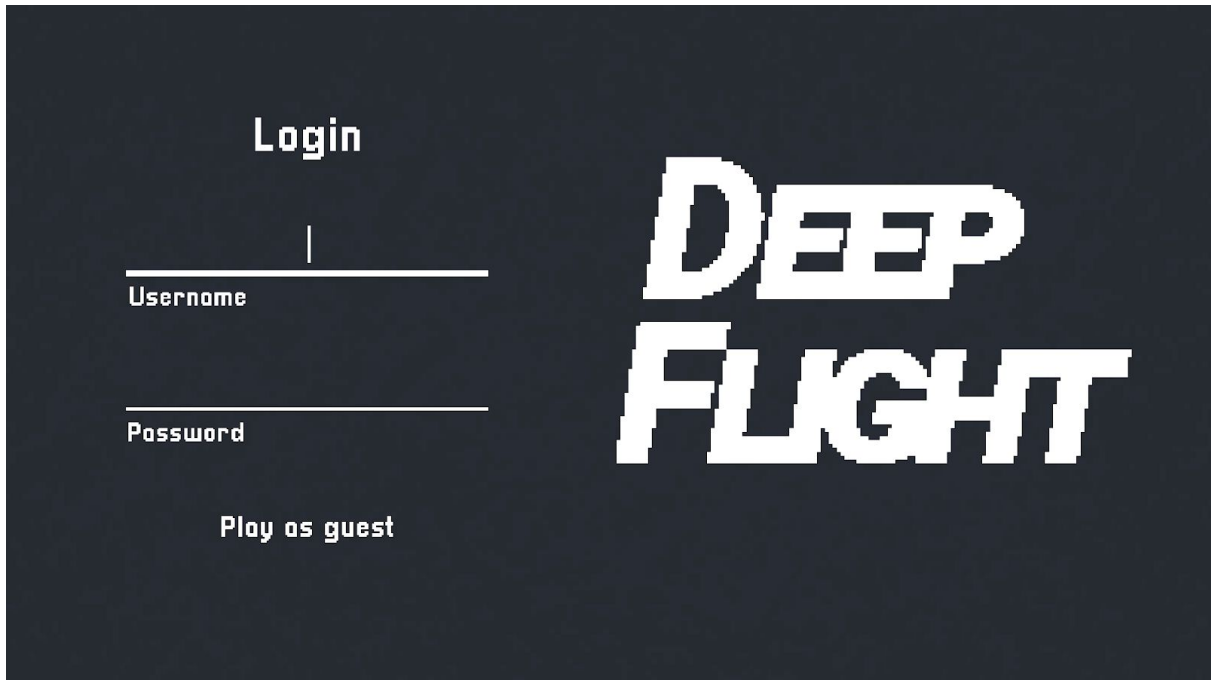
- MongoDB: Atlas server
- GameAPI, UserAPI, Webpage: Maltes egen server cloud server (Amazon EC2) under domænet 'maltebp.dk'.
- Universer Updater og Prometheus: dist.saluton.dk

Vi håber at I får god fornøjelse med spillet og at I kommer på highscoren

Med venlig hilsen
Gruppe 0

Bilag A: Skærbilleder af spillet

Spillets grafiske udforming og funktion er en del af en aflevering i et andet kursus. Billederne er kun med for at vise hvad der bliver downloadet fra serveren. Spillets kommunikation med

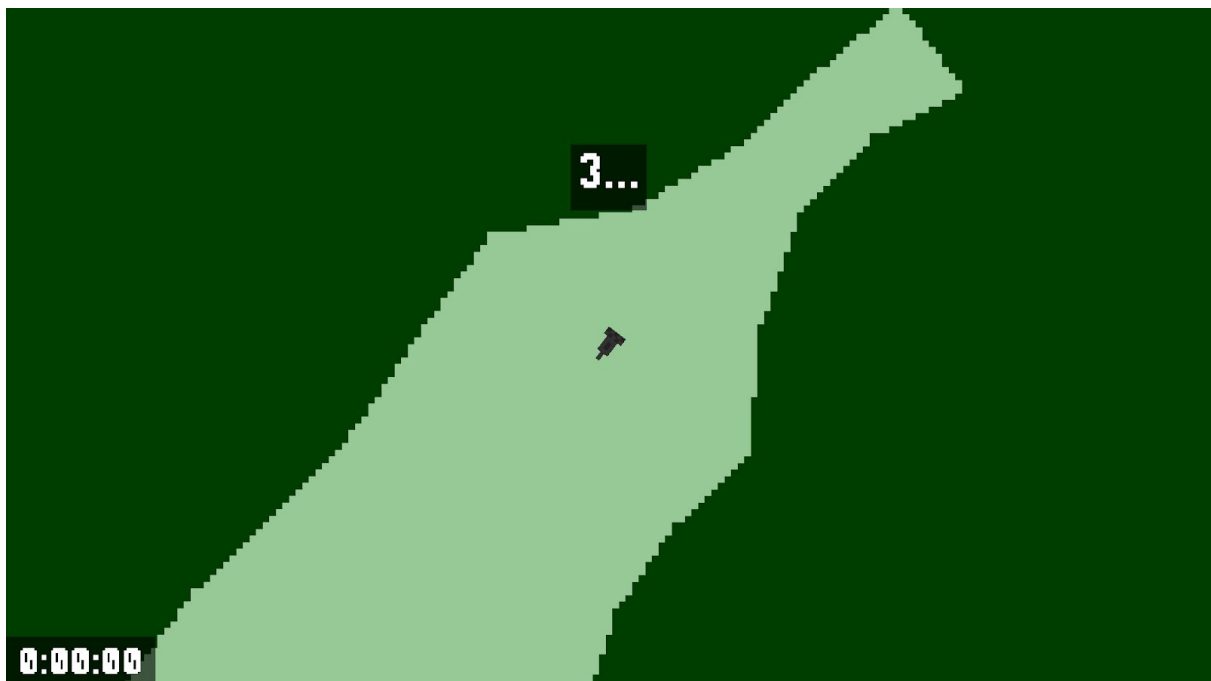


serveren er en del af afleveringen i dette kursus.

Loginsiden



Loadingssiden der vises mens en banes data hentes



Online Rankings

Universal Rankings (Top 5)

Rank	Username	Rating
#1	kingkong	7.00
#2	dennis	5.88
#3	hogrid123	3.46
#4	admin42	2.92
#5	s132952	2.50

Your Rank:
#7

Your Rating:
2.00

Round #1 rankings

Rank	Username	Rating
#1	kingkong	7.00
#2	dennis	5.88
#3	hogrid123	3.46
#4	admin42	2.92
#5	s132962	2.50

Your Rank:
#7

Your Rating:
2.00

Bilag B: Skærbillede af TUI

```
~~~~~ DEEPFLIGHT - GAMEAPI ~~~~~  
~~~~~ MENU ~~~~~  
  
1) Find Track by ID  
2) Find Planet by ID  
3) Find ALL Planets  
4) Find Current Round  
5) Find Previous Round  
6) Find ALL Rounds  
7) Find Universal Rankings  
  
666) Exit  
  
Enter the function you wish to use:  
1  
Enter TrackID:  
5eb81b8096418b4354f930fe  
Track information:  
Track{id=5eb81b8096418b4354f930fe, name=UBMJ-619, planetId=5eb8179bdc749d4b22bf55c5, seed=1507990126, times{s185139=1234,
```

Øverst ses Menuen. Nederst ses brugen af funktion 1 "Find Track by ID" og indtastningen af et trackID og resultatet af kaldet.

Bilag C: Teoriafsnit

Teoriafsnit af Gruppe 0, 62597 Backend & DIST DTU, Forår 2020

Erlend Tyrmi, Studienummer s132962.

Andre må distribuere, remix, tweake, og bygge videre på teoriafsnittet, med kildeangivelse.



<http://creativecommons.org/licenses/by/4.0/>

Kom i gang med React

En kort guide til at komme i gang med en hjemmeside i React. Når man arbejder i React bruger man terminalen til at starte og bygge projektet. Det er ikke mange kommandoer man skal lære.

Produktionsmiljø	1
Grundlæggende koncepter i React	2
Props	2
Virtual DOM	2
Component	2
State	3
Function (functional component)	3
Hvordan give argumenter til 'component' og 'function'.	4
Deployment	4
Npm start	4
Build	5
Konklusion	6

Produktionsmiljø

Først skal man have installeret node.js: [<https://nodejs.org/en/>].

For at starte et nyt projekt skriver man

```
npx create-react-app my-app  
cd my-app  
npm start
```

React skaber en projekt-folder, i dette tilfælde 'my-app'.

Du burde nu se en demo-side med blå baggrund, og en link til følgende guide, som jeg kan anbefale alle at følge. Den forklarer de grundlæggende elementer af React: <https://reactjs.org/tutorial/tutorial.html>. Guiden er kort og ret simpel, men hjælper dig hurtigt at forstå hvad Components, Functions, props og state er i React.

Man kan åbne folderen i en IDE, for eksempel VS Code. (Det er smart at bruge en ide der både kan formatere js og har en god terminal).

Grundlæggende koncepter i React

Props

Vi nævner props kort allerførst, for det kan forvirre en smule: props står for properties, og er ikke et reserveret ord, men en konvention. Når en funktion eller komponent modtager argumenter, er det i form af et objekt, lidt på samme måde som 'char *argv[]' i c. Dette objekt hedder props, og har public members. Man kan altså kalde det noget andet, men det kan være en god ide at holde sig til 'props'.

Virtual DOM

DOM er som kendt Document Object Model, og er måden en HTML side er opbygget af objekter på i JavaScript. [\[w3schools DOM\]](#) React har sin egen virtual DOM, der er mere effektiv og lidt mere skjult for programmøren. Det er dette objekt React bruger til at opdatere DOM mens siden vises. [\[reactjs.com VDOM\]](#)

Component

'Component' er måske det mest grundlæggende React-element. En komponent er en klasse, der repræsenterer en del html + javascript (jsx), der sendes til Virtual DOM.

```
class Hello extends React.Component {  
  render() {  
    return <h1>Hello</h1>;  
  }  
}
```

Hvis komponenten har fået argumentet 'name' vil det kunne bruges som nedenunder. Teksten i variablen 'name' bliver her skrevet til VDOM som jsx:

```
class Hello extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

En komponent har *altid* en render-metode, der returnerer JSX-kode. JSX er et sprog der kombinerer HTML og JavaScript, eller XML og Javascript. [\[react.js JSX\]](#). Kodeeksemplet over viser hvordan JSX ligner almindelig html, men med muligheden for at angive JavaScript-kode i Tuborg-klammer {}. Hvis return metoden indeholder flere linjer, bruger man

parenteser. Bemærk, at render altid skal returnere et jsx-element. Hvis man har flere, skal de pakkes i for eksempel en `<div>`.

```
class Hello extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, {this.props.name}</h1>;
        <p>Goodbye.</p>
      </div>
    )
  }
}
```

State

Klassen kan også have en state. Dette betyder at de enkelte dele af siden, for eksempel et spørgeskema, kan holde styr på brugerens valg, uden at andre dele af dokumentet ved noget om det. En komponent har en constructor, der ikke er påkrævet hvis man ikke bruger state eller lokale funktioner.

```
class Hello extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: 'Peter' };
    this.handleClick = this.handleClick.bind(this);
  }
  render() {
    return (<h1>{this.state.name}</h1>);
  }
}
```

I dette kodeafsnit erklæres også en lokal funktion 'handleClick'. Dette er ikke det samme som en 'function' i React. Lokale funktioner bruges som almindelige funktioner inde i klassen. Her bruges også 'bind', som binder funktionen til dette klasse. Vi går ikke i dybden med bind i denne guide. [[freecodecamp bind](#)].

Hver gang du opdaterer state, sørger React for at tegne Virtual DOM igen, og siden genopfriskes. Bemærk, at det ikke er nok at bare opdatere enkelte variabler i state, men at metoden 'setState' skal kaldes.

```
this.setState({name: 'Peter'})
```

Function (functional component)

React Function kan anses som en letvægts komponent. Den kaldes også 'function component'. Den har ingen constructor, ingen state, men skal have en return-metode. Den defineres sådan her:

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

Hvorfor bruge funktions-komponenter: Der er mindre kode der skal skrives, og små elementer i Virtual DOM, der ikke behøver state bør skrives på denne måde.

Hvordan give argumenter til 'component' og 'function'.

JSX har en lidt speciel syntax for argumenter [[reactjs JSX](#)]. Når man angiver argumenter til en funktion eller komponent, ser det sådan ud:

```
function Greeting(props) {  
  return (  
    <MorningGreeting name={props.name}/>  
  );  
}
```

Her forestiller vi os, at der allerede er en komponent der hedder MorningGreeting, og at den behandler props.name. Greeting har også fået name som argument, og giver det videre som argument til MorningGreeting. Læg mærke til Tuborg-klammerne, de angiver at vi går fra HTML til JavaScript.

Deployment

Der er mange alternativer, når man skal deploye hjemmesiden.

Npm start

Man kan i teorien deploye en hjemmeside med

```
npm start
```

Dette er ikke et optimeret build, og fylder lidt mere på disken. Alle ændringer i filer på disken vil blive opdateret øjeblikkelig på hjemmesiden, som er smart ved test men kan være farlig i produktion. Udover det er den ret stabil.

Serve

Den officielt anbefalede måde at deploye på, er at installere 'serve'

```
npm install -g serve
```

Og så skrive:

```
serve -s build -l 80
```

Argumentet '-' står for listen, og angiver, at serveren skal lytte på port 80. Dette er en statisk server, det vil sige at den er egnet til hjemmesider, hvor frontend er skrevet i html/javascript, og ikke dynamisk indhold med for eksempel PHP.

[https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server]

Build

Man kan bygge projektet med kommandoen

```
npm run build
```

Dette genererer en folder, der hedder 'build', og indholdet af 'build' kan så for eksempel deployes med Apache Server. Man kopierer helt enkelt indholdet af 'build' til www-folderen på serveren. Detaljer er beskrevet her:

[<https://gist.github.com/ywwwtseng/63c36ccb58a25a09f7096bbb602ac1de>]

Vi har valgt at bruge docker i vores projekt, og har brugt følgende dockerimage:

```
# build environment
FROM node:10-alpine as build
WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH
COPY package.json ./
COPY package-lock.json ./
RUN npm ci --silent
RUN npm install react-scripts@3.4.1 -g --silent
COPY . ./
RUN npm run build

# production environment
FROM nginx:stable-alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Her bygges projektet i en container og kopieres så over i en anden. Dette er for at slanke containeren så meget som muligt.

Der er mange guides til at deploye react på diverse blogger, men rigtig mange af dem bruger npm start eller er dårlig forklaret. Dette er den vi har brugt til vores projekt:

[<https://mherman.org/blog/dockerizing-a-react-app/>]

Konklusion

Der er mange flere ting at lære i react med hensyn til mappestruktur og setup. Her har vi forsøgt at forklare et par af de ting, der umiddelbart kan virke fremmede, når man sætter sig ind i React.

React har en ret begyndervennlig dokumentation, og det anbefales på det varmeste at starte med [deres tutorial](#).