CHALMERS

SCONES: Ett incitamentssystem för delning av lösningar baserat på blockkedjeteknik

Examensarbete inom Data- och Informationsteknik

Jonathan Uhre
Anton Eliasson Gustafsson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2020

EXAMENSARBETE

SCONES: Ett incitamentssystem för delning av lösningar baserat på blockkedjeteknik

Jonathan Uhre
Anton Eliasson Gustafsson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2020

SCONES: Ett incitamentssystem för delning av lösningar baserat på blockkedjeteknik
Jonathan Uhre
Anton Eliasson Gustafsson

Examinator: Peter Lundin, Institutionen för Data- och informationsteknik
Handledare: Sakib Sistek, Institutionen för Data- och informationsteknik
Handledare: Henrik Fagrell, Diadrom Systems AB

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik
Göteborg 2020

Sammandrag

På internet finns forum där lösningar på problem eftersöks. Ofta kan användare på dessa forum avgöra lösningars legitimitet med hjälp av uppskattningsmekanismer. Syftet med detta projekt är att skapa incitament för kvalitativ problemlösning på forum där lösningar kan valideras.

För att åstadkomma detta behövdes informationsinsamling ske innan systemets delar planerades mer ingående. Därefter delades projektet upp i två parallella delar. I den ena delen utvecklades smarta kontrakt. I den andra delen låg fokus på att implementera smarta kontrakt mot en applikation med ett användargränssnitt.

Resultatet är ett smart kontrakt som kan köras på Ethereums blockkedja, kopplat till en applikation med ett användargränssnitt. Kontraktet är inte bundet till specifika forum eller specifika forumanvändare. Vem som helst kan utnyttja kontraktet och deklarera sig som forumägare i kontraktet. Denna forumägare kan lägga till Ethereum-adresser som forumanvändare. Dessa forumanvädare ges uppskattningspoäng av forumägaren, vilket de kan utnyttja för att få ersättning i valutan. En forumägare i detta sammanhanget kan själv avgöra vilka uppskattningsmekanismer som bidrar till mest kvalitet vad gäller lösningar på problem, för att sedan använda dem. Kontraktets funktionalitet implementeras i en applikation med ett användargränssnitt.

Applikationen demonstrerar en användares koppling till blockkedjan genom smarta kontrakt. Den har även ett användarsystem som sköts av en databas. Det är tänkt att visa hur ett decentraliserat och centraliserat system kan samverka i en applikation. Applikationen har övergripande funktionalitet som gynnar både forumanvändare och forumägare.

Systemet utvecklades lokalt. Genom vidareutveckling och samarbete med existerande forum kan systemet implementeras i verkligheten. Systemet kan som helhet användas för att skapa incitament för att delge kvalitativa lösningar på problem. Däremot kan kontraktets funktioner användas brett, och kan användas på andra typer av forum där inte problemlösning är centralt. Systemet kan gynnas av en vidare diskussion om förbättringar.

Nyckelord: Blockkedja, Smarta kontrakt, Ethereum, Kryptovaluta, Decentraliserad applikation, Node.js, Solidity

Abstract

On the internet there are forums where solutions for problems are sought. Users on these forums can often determine the legitimacy of solutions with the help of appraisal systems. The purpose of this project is to create additional incentive for qualitative problem solving on forums where the solutions can be validated.

In order to achieve this, information gathering was required before planning of the system's parts. After completed planning, the project got divided into two parallel development branches. The first branch's aim was to focus on smart contract developing. The other branch's focus was to deliver a user application where the smart contract could be implemented.

The result is a smart contract which can run on the Ethereum blockchain, connected to an application with a user interface. The contract is not limited to specific forums nor specific forum users. Anyone can utilize the contract and declare themselves as forum owner in the contract. The owner can add Ethereum addresses as forum users. These users are given appraisal points by the forum owner, which can be exchanged to the created currency. As a result, the forum owner can solely determine and adapt to what appraisal systems that contribute most in terms of quality in problem solving. The functionality of the contract is implemented in an application. It demonstrates a users' connection to the blockchain through a smart contract. It also has a user login system connected to a database. This is intended to demonstrate how a decentralized and centralized system can cooperate using an application. The application has functionality that could support both a forum owner and a forum user.

The system was developed locally. Through further development and cooperation with existing forums, the system can be implemented in reality. The system as a whole can be used to create incentives to provide qualitative solutions to problems. However, the smart contract's functions can be used widely and thus on other types of forums where problem solving is not central. The system can benefit from a further discussion of possible improvements.

Keywords: Blockchain, Smart contract, Ethereum, Crypto Currency, Decentralized application, Node.js, Solidity

Förord

Rapporten utfördes som examensarbete hos Institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola. I detta projekt utforskades hur smarta kontrakt kan utnyttjas för att skapa en digital valuta kopplad till uppskattningsmekanismer på forum.

Vi vill tacka våra två handledare:

Hos Diadrom Systems AB, Henrik Fagrell, PhD.
Tack för att vi fick utforska din ide, och framförallt - tack för all din vägledning.

På Chalmers Tekniska Högskola, Sakib Sistek.
Tack för din enorma entusiasm, inspiration och vägledning.

Vi skulle också vilja tacka alla främlingar på Stack Overflow.

Begreppslista

Blockkedja - Decentraliserat nätverk för lagring med noder som delar samma data
Ethereum - En plattform för utveckling av smarta kontrakt som kan köras på en blockkedja
Smart kontrakt - Programkod som kan köras på en blockkedja
Karma - Uppskattningspoäng
API- Application Program Interface (på svenska: applikationsprogrammeringsgränssnitt)
CLI - Command Line Interface (på svenska: kommandotolk)
HTTP - Hypertext Transfer Protocol

Innehållsförteckning

1 Inledning

På internet finns det forum där individer eller organisationer eftersöker lösningar på problem. För att avgöra huruvida bra eller dåliga lösningar är kan man se på uppskattningsmekanismer, exempelvis uppvoteringar och medaljer. Att skapa mer incitament för att dela med sig av lösningar på problem kan tyckas vara av intresse eftersom det främjar problemlösning generellt. Att ge ersättning i form av en valuta kopplad till uppskattningsmekanismer är ett sätt att skapa mer incitament. I denna rapport undersöks detta genom att utveckla en valuta i ett smart kontrakt som kan köras på blockkedja, samt demonstrera detta på applikationsnivå. Syftet med detta projekt är att skapa incitament för kvalitativ problemlösning på forum där lösningar kan valideras. Det teoretiska bidraget blir då en valuta med ett nischat syfte. Det praktiska bidraget blir målet att utnyttja smarta kontrakt och blockkedjeteknik. Detta för att skapa en digital valuta kopplad till uppskattningsmekanismer som existerar på forum, samt en applikation där smarta kontrakt kan implementeras.

1.1 Bakgrund

På internet finns många forum för diskussion om problem. Stack Overflow och Reddit är bara två exempel. Dessa forum kan ha trådar där användare efterlyser lösningar på problem. Andra användare kan sedan läsa dessa trådar i jakt efter lösningar på samma eller liknande problem. För att avgöra huruvida bra eller dåliga användares lösningar är, existerar uppskattningsmekanismer på forum. Dessa kan exempelvis vara uppvoteringar, medaljer, karma etc.

En fråga man dock kan ställa sig, är vad för sorts incitament det handlar om när till synes främlingar hjälper andra främlingar med problem. Oavsett svaret på den frågan kan det tyckas vara av intresse att skapa mer incitament. Dels för att fler skulle vilja dela sina lösningar, och dels för att det främjar problemlösning generellt.

Ett sätt att skapa mer incitament för problemlösning skulle kunna vara att ge individer en möjlighet till ersättning i form av någon valuta som har en koppling till uppskattningsmekanismer på forum. Att utforma en valuta med regler kan göras genom smarta kontrakt, vilket är programkod som kan köras på blockkedjor [1].

1.2 Syfte

Syftet är att skapa incitament för kvalitativ problemlösning på forum där lösningar kan valideras.

1.3 Mål

Målet är att utnyttja smarta kontrakt och blockkedjeteknik för att skapa en digital valuta kopplad till uppskattningsmekanismer som existerar på forum. Användare ska också kunna utnyttja en applikation för att intereagera med valutan.

1.4 Frågeställning

Hur kan man skapa ett system, baserat på blockkedjeteknik, som ger incitament att dela lösningar på problem, samt premierar lösningar av god kvalitet?

1.4.1 Delfrågor

- Är det möjligt att utveckla systemet genom ett eller flera smarta kontrakt som kan köras på en blockkedja?
- Är det möjligt att utforma en koppling som binder samman användare på forum med en digital, decentraliserad valuta?
- Är det möjligt att i det smarta kontraktet, ha funktionalitet som låter användare få ersättning genom redan existerande uppskattningsmekanismer?
- På vilket sätt kan uppskattningsmekanismer användas för att premiera lösningar av god kvalitet i systemet?
- Är det möjligt att utforma det smarta kontraktet på sådant vis att; det smarta kontraktet inte ska vara bundet till ett specifikt forum eller specifika användare, utan ska kunna utnyttjas av vem som helst?
- Är det möjligt att applicera det smarta kontraktet på en applikation med ett användargränssnitt?
- Är det möjligt att i applikationen, koppla centraliserat data med decentraliserad data?

1.5 Avgränsningar

Val av plattform vad gäller utveckling av blockkedjsteknik är Ethereum. Detta eftersom det verkar vara väletablerat och information tyckas vara relativt enkel att hitta. För att underlätta testning och genomförandet av systemetkonstruktionen körs all kod lokalt. Blockkedjan är en lokal simulering av Ethereums blockkedja. Alla noder på blockkedjan är upplåsta. Detta innebär att noderna i blockkedjan, som normalt måste signera transaktioner med en privat nyckel, inte behöver signera någon transaktion alls. Noderna är alltså konstant upplåsta. Detta underlättar testningen av systemet.

2 Bakomliggande tekniker och tjänster

Arbetet grundas i blockkedjeteknik, som simpelt sett är en decentraliserad lista med transaktioner [2]. Tekniken har bland annat lett till skapandet av *Ethereum*, vilket är en programmerbar distribuerad maskin och blockkedja [3]. För att utveckla system kopplat till Ethereum krävs vissa verktyg och bibliotek.

2.1 Blockkedjeteknik

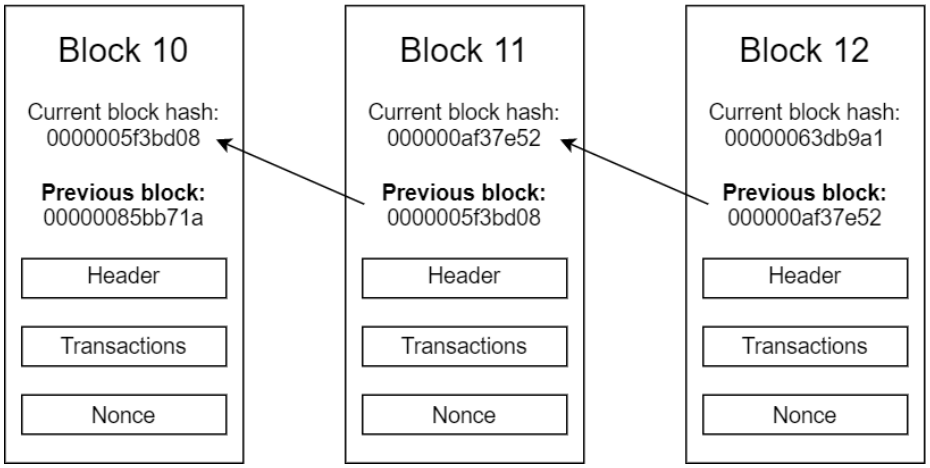
En blockkedja är praktiskt taget en växande lista av transaktioner som är sammanlänkad med kryptografi. Den är distribuerad mellan många datorer och kan köra verifierbar mjukvara för att kontrollera hur data läggs till. Målet är att datan som lagras ska vara oföränderlig [2].

En kopia av hela blockkedjan lagras hos varje nod i blockkedjenätverket, vilket innebär att blockkedjan är decentraliserad. En annan egenskap hos en blockkedja är dess distribuerade peer-to-peer-nätverk. Detta innebär att alla noder kan kommunicera direkt med varandra, utan att gå via en annan nod [2].

2.1.1 Uppbyggnad

I grunden är en blockkedja sammanlänkade block med data, vilket rent abstrakt kan liknas med en kedja av datablock. Varje block används för att lagra transaktioner. En transaktion innebär att ny data skrivs på blockkedjan, där datan kan representera nästan vilken information som helst. Det kan exempelvis vara finansiella transaktioner, körbar kod eller dokument [2].

Blocken länkas samman genom att lagra ett kryptografiskt hash-värde av föregående blocks metadata, som lagras i dess header. Ett hash-värde är ett unikt, ofta hexadecimalt nummer som representerar ett block med data. Detta beräknas med en matematisk funktion [2]. *Figur 2.1* visar hur blocken är sammanlänkade och vilken data som lagras.



Figur 2.1: Uppbyggnad av blockkedja

Varje block har en header som innehåller en tidsstämpel, ett blocknummer och annan beskrivande information. Den innehåller även metadata om allt innehåll i blocket [2].

När ett hash-värde ska beräknas finns det ofta vissa krav för hur resultatet ska se ut. Ett exempel är i *Figur 2.1* där de sex första siffrorna i varje hash är 0. För att kunna anpassa hash-värdet används en så kallad *nonce*. Det är bitar i ett block som kan sättas till ett frivilligt värde. En hash-funktion ger ett tillsynes slumpmässigt värde som är oförutsägbart. Att ändra en nonce så att de sex första siffrorna i hash-värdet ska bli 0 är därmed upp till slumpen. Det gör att beräkningen av hash-värden tar lång tid och behöver utföras av många datorer. Att beräkna hash-värden är även känt som *mining*. Detta förhindrar enstaka noder från att kontrollera blockkedjan. Tanken är att noderna ska vara beroende av ett stort nätverk med datorer som utför beräkningar för att lägga till block [4].

2.1.2 Lägga till block

Noder på blockkedjan måste komma överens om hur nya block ska tillåtas. Eftersom alla noder har samma auktoritet måste det finnas konsensus om hur block ska tillåtas att läggas till. De flesta strategier kräver att en majoritet av noderna godkänner det nya blocket. Anledningen att det är viktigt med konsensus hos noderna är för att ett block som lagts till på blockkedjan aldrig kan ändras [2].

Innan ett block kan läggas till på blockkedjan måste en kryptografisk hash till föregående block beräknas. Det är länken till föregående block och en garanti att det aldrig kommer ändras. Eftersom hash-värdet beräknas på metadatan i föregående blocks header, inkluderas även det föregående hash-värdet. Så om ett block ändras har alla efterliggande block en felaktig hash [2].

För att validera blockkedjan granskas samtliga block för att försäkra att de lagrar korrekta hash-värden. Om alla värden stämmer är blockkedjan giltig. Så fort datan hos ett block ändras blir den kopian av blockkedjan ogiltig. Det är anledning att blockkedjor anses vara oföränderliga [2].

2.2 Ethereum

Iden om *Ethereum* kom i samband med *Bitcoins* växande intresse. 2013 publicerade Vitalik Buterin en vitbok som beskrev en ny ide om vad som skulle kallas för Ethereum. Vitboken fick både tekniskt och finansiellt stöd av en schweizisk ideell organisation som idag heter *Ethereum Foundation* [2]. Ethereum är en programmerbar distribuerad maskin och blockkedja. Noderna i Ethereums nätverk exekverar bytekod mot blockkedjan, vilket genererar nya tillstånd. Bytekoden definieras av programmerbara smarta kontrakt. Dessa smarta kontrakt definierar vilken data och vilka funktioner som ska köras mot blockkedjan. Stödet för smarta kontrakt i Ethereums blockkedja gör det möjligt för utvecklaren att skapa flera olika decentraliserade applikationer med varierande syften. Invävt i Ethereum, tillkommer också kryptovalutan *ether* [3]. Varje nod i Ethereums blockkedja har en egen unik adress, vilket gör det möjligt att avgöra

vilken nod en avsändare är. Vidare har varje nod en egen privat nyckel, vilket en nod använder för att signera transaktioner [2].

2.2.1 Smarta Kontakt i Ethereum

Konceptet om smarta kontrakt är inte exklusivt för Ethereum. Däremot kan Ethereum tyckas vara den största och mest kända plattformen där konceptet också används. Ett smart kontrakt i Ethereum är programkod som kan köras på Ethereums blockkedja. Ethereum garanterar att den kod som körs på blockkedjan får vissa egenskaper. Dessa egenskaper är bland annat oföränderlighet, deterministiska körningar, ett distribuerat nätverk och ett verifierbart tillstånd. Varje smart kontrakt på Ethereums blockkedja får en egen unik adress för att andra noder ska kunna kommunicera med kontraktet. Utvecklaren kan definiera vilken data som kan skrivas på blockkedjan genom smarta kontrakt. Vidare kan kontraktet innehålla funktioner som ändrar på eller läser datan [1].

2.2.2 Gas

Smarta kontrakt exekveras av alla fulla noder i nätverket. En full nod är en sådan som har hela kopian av blockkedjan. Detta skapar redundans, men också en kostnad i form av energi. För att kompensera energiåtgången tillkommer en avgift då en nod i nätverket vill exekvera kod som ändrar på tillståndet i blockkedjan. Avgiften för exekvering mot blockkedjan kallas *gas*. Vid läsning av blockkedjan tillkommer dock ingen avgift, alltså ingen gaskostnad. Gas betalas med ether. En miner i Ethereum (alltså en nod som processar och verifierar exekveringar), är den nod som belönas med motsvarande gasavgift [5].

2.2.3 ERC-20

För att utveckla en digital valuta (också kallat för *token*) i Ethereum, följs ofta en standard som kallas för *ERC-20*. Denna standard är också ett gränssnitt för variabler och funktioner som ska finnas med i utvecklingen av en valuta. Genom gränssnittet försäkras att valutan kan skickas mellan adresser, och att varje adress har ett saldo [6]. För att se ett fullständigt gränssnitt, se Bilaga C.

2.3 Mjukvara

I avsnittet beskrivs mjukvaran som använts i systemet. Det består av verktyg, programmeringsspråk och JavaScript-bibliotek.

2.3.1 Truffle Suit

Truffle Suit är en samling av verktyg som underlättar utvecklingen av smarta kontrakt på Ethereums blockkedja, och decentraliserade applikationer [7]. Truffe Suit består av tre olika verktyg, varav två av dessa används i detta projekt. Det första verktyget som används är *Truffle* (inte att missförstå med Truffle Suit). Truffle är ett verktyg och CLI med flera olika funktioner som assisterar utveckling av smarta kontrakt för Ethereums blockkedja [8]. Några av dessa funktioner är bland annat kompilering av smarta kontrakt, testing av smarta kontrakt och *migrering* av smarta kontrakt på en blockkedjan. Migrering av kontrakt

på blockkedjan innebär att kontraktet utplaceras och blir körbart på blockkedjan [2]. Det andra verktyget som används, *Ganache*, är en lokalsimulerad ethereum-blockkedja [9].

2.3.2 Solidity

Det vanligaste programmeringsspråket för att skriva smarta kontrakt, är språket *Solidity*. Detta språk är influerat av bland annat *JavaScript* och *Python* [10].

2.3.3 HTML

HTML (Hyper Text Markup Language) är ett sidbeskrivningsspråk som används för att skapa webbsidor. Det använder olika element för att förklara för en webbläsare hur en sida ska visas. Elementen kan exempelvis representera en rubrik, brödtext eller hyperlänk [11].

2.3.4 JavaScript

JavaScript är ett skriptspråk som kan köras i en webbläsare eller på en server. Det kan användas tillsammans med HTML och kan då köras automatiskt när en HTML-sida laddas, alternativt vid interaktioner på sidan [12]. Programmerar kan dela kod genom JavaScript-bibliotek. Det tillåter andra programmerar att ta del av redan existerande JavaScript-kod [13].

2.3.5 Node.js och NPM

Node.js är en asynkron JavaScript exekveringsmiljö för nätverksapplikationer. Den kan exempelvis användas för att skapa en HTTP-server eller en databasserver. Det går även att skriva JavaScript-kommandon till en Node-server genom ett CLI [14].

NPM (Node Package Manager) är en pakethanterar för Node-paket, som egentligen är JavaScript-bibliotek. Kommandot *npm install <paketnamn>* används för att ladda ner paket. Om Node-projektet har dependencies kan samtliga dependencies laddas ner genom att köra kommandot *npm install* [15].

2.3.6 web3.js

web3.js är en samling JavaScript-bibliotek som används för att kommunicera med Ethereum-blockkedjan. Kommunikationen kan ske genom alla noder som tillåter åtkomst via HTTP. Det fungerar med både lokala och offentliga noder [16].

2.3.7 React

React är ett JavaScript-bibliotek som används för att bygga användergränssnitt. Det är komponentbaserat och varje komponent kan hantera sina egna variabler som kallas för *states*. När en komponent anropas kan variabler skickas med, de kallas för *props*. Komponenter använder funktionen *render()* för att returnera det som ska visas i applikationen. Den renderar alltså komponenten från backend till front-end [17].

För att använda React på flera sidor används biblioteket *React Router*. Egentligen används inte flera sidor, utan allt innehåll renderas till en HTML-fil. Router tillåter olika komponenter att laddas beroende på hemsidans URL. Webbläsaren kan ändå navigera som vanligt med fram- och bakknappar [18].

2.3.8 MySQL

MySQL är ett databassystem som kan köras på en server. Det använder standard SQL-queries. För att MySQL ska kunna köras som en Node-server används ett MySQL JavaScript-bibliotek [19].

3 Metod

På grund av coronapandemin utfördes arbetet till stor del hemifrån med online-kommunikation genom kommunikationsprogrammet *Discord*. Arbetet skulle ursprungligen utförts på företag. Där hade det funnits möjlighet för mer konsultation än den som genomförts online. Arbetet hemifrån resulterade dock i mer ostört arbete.

Först planerades en abstrakt idé om systemets syfte och delar. Eftersom kännedomen av de planerade verktygen som skulle användas inte var hög, kunde inte all funktionalitet planeras från start. Först behövdes alltså en relativt omfattande informationsinsamling genomföras, för att sedan lyckas planera mer funktionalitet i systemet.

3.1 Arbetssätt

I de flesta fall präglades arbetet av en cykel; undersökning, implementation och testning. Informationsinsamlingen bestod till stor del av att studera lösningar till liknande problem. Det gav en förståelse för möjliga lösningar, som vidare kunde analyseras och anpassas till systemet. Om ytterligare problem uppstod fortsatte informationsinsamlingen, fast för ett mer specifikt problem. När implementationen var färdig genomfördes tester. Det ansågs viktigt för att försäkra kvalitet och förhindra framtida problem i systemet. Implementationen kunde då integreras i systemet.

Till en början låg fokus på att undersöka, implementera och testa ett enkelt smart kontrakt kopplat till ett enkelt skript som i sin tur kunde kopplas till en blockkedja med ett smarta kontraktet. Det föreföll sig då lämpligt att först konstruera en slags *hello-world-applikation*. Efter implementation och test av hello-world-applikationen delades arbetet upp i två delar som kunde köras parallellt. I den ena delen låg fokus på det smarta kontraktet. I den andra delen låg fokus istället på en applikation med ett användargränssnitt. De två parallella delarna sammanfogades vid ett flertal tillfällen.

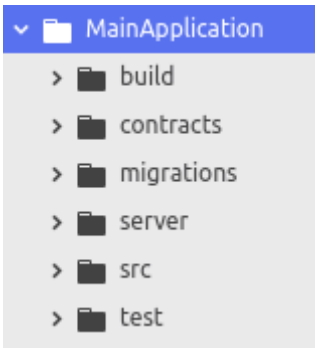
4 Systembeskrivning

I bakgrunden av systemet körs en lokal simulering av Ethereums blockkedja. På blockkedjan finns smarta kontrakt, skrivna i Solidity, med egna unika adresser. De smarta kontrakten har definierade variabler och funktioner, vilka är åtkomliga för alla noder i nätverket. Det mest relevanta kontraktet heter *Forum.sol*. I kontraktet definieras en digital valuta som kallas *SCONES*, samt forumhantering. En nod på blockkedjan kan antingen agera som en forumanvändare, eller forumägare.

Genom en applikation demonstreras en klients kopplingar till blockkedjan och det smarta kontraktet. Applikationen är till stor del byggd i demonstrativt syfte. Den synliggör hur det kan se ut för både forumanvändare och forumägare. Den första vyn av applikationen är ett inloggningssystem kopplat till en centraliserad databas. En användare kan således logga in och göra olika val för att intereagera med blockkedjan och det smarta kontraktet. Se avsnitt 4.4 för en detaljerad beskrivning.

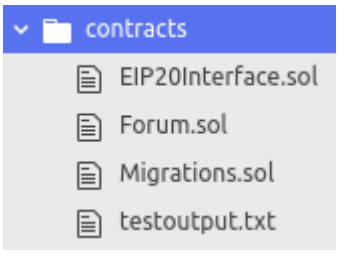
4.1 Övergripande struktur

En förenklad beskrivning av systemetkonstruktionen är en applikation som samverkar med en blockkedja genom ett smart kontrakt. För att åstadkomma detta behövs ett antal delar. I Figur 4.1 syns mappstrukturen för hela systemkonstruktionen. I detta avsnitt beskrivs en översiktlig bild av systemkonstruktionen utifrån mappstrukturen.



Figur 4.1: Mappstruktur för MainApplication

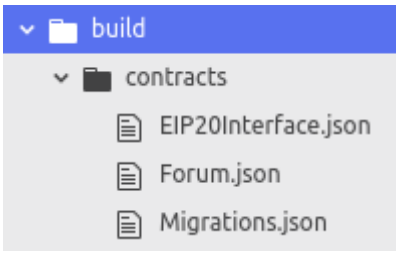
- **contracts**
  Mappen *contracts* innehåller de smarta kontrakten. Se Figur 4.2 för filstruktur.



Figur 4.2: Filstruktur i contracts

- **EIP20Interface.sol**
  Kontraktet är ett gränssnitt för en token (eller valuta) till Ethereums blockkedja. Gränssnittet innehåller funktionssignaturer och variabler som måste ingå för en valuta. Detta är hämtat från [20]. Se Bilaga C.
- **Forum.sol**
  I kontraktet definieras funktionalitet för valutan SCONES och forumhantering. *Forum.sol* implementerar *EIP20Interface.sol*. Se Avsnitt 4.2 för en fördjupning av *Forum.sol*.
- **Migrations.sol**
  Kontraktet är autogenererat genom truffle. Syftet med kontraktet är att lyckas pusha alla smarta kontrakt till en ethereum-blockkedja. Genom *migrations.sol* skapas kontraktreferenser. Detta för att veta vilka kontrakt som körs på det lokala nätverket. Se Bilaga D.

- **build**
  Mappen build och dess innehåll genereras genom truffle då de smarta kontrakten kompileras utan felaktigheter. Innehållet är JSON-representationer av de smarta kontrakten. Se Figur 4.3 för filstruktur.



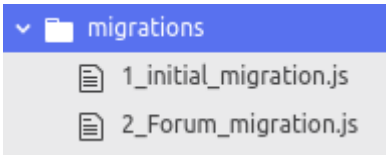
Figur 4.3: Filstruktur i build

JSON-representationerna blir en del av kopplingen mellan applikationen och blockkedjan. JSON-filerna innehåller bland annat:

- Kontraktets namn.
- Kontraktets ABI (Application Binary Interface), vilket är en lista av alla funktioner och dess signaturer.
- Kontraktets bytekod.
- En lista med nätverk där kontraktet finns utplacerat. I detta fallet, en lokalsimulerad blockkedja via ganache.

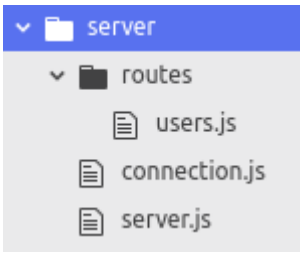
  - Kompilatorversion.
  - Kontraktets adress.

- **migrations**
  Mappen genereras automatiskt via truffle. Filen, *1_initial_migration.js* likaså. Däremot är inte *2_Forum_migration.js* autogenererad. Innehållet är två JavaScript-filer vars syften är att funktionsanropa migrering av de smarta kontrakten till blockkedjan. Se Bilaga E, respektive Bilaga F för fullständig kod. Se Figur 4.4 för filstruktur.



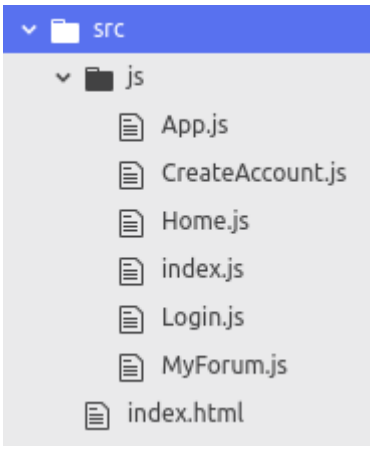
Figur 4.4: Filstruktur i migrations

- **server**
  Mappen innehåller filer för att skapa en databas-server. Filen *server.js* hanterar hela servern. Den importerar *connection.js* för att skapa en anslutning till databasen och *users.js* för att tillåta kommunikation via HTTP-requests. Mer om filerna och dess funktioner förklaras i avsnitt 4.3.1. Se Figur 4.5 för filstruktur.



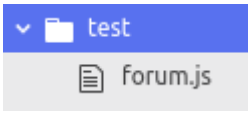
Figur 4.5: Filstruktur i server

- **src**
  Mappen *src* innehåller alla filer som används för applikationen. I filen *index.html* renderas alla React-komponenter, vilket bestäms i *index.js*. Resterande filer skapar olika komponenter som interagerar med databasen och det smarta kontraktet. Mer om filerna och dess funktioner förklaras i avsnitt 4.4.3. Se Figur 4.5 för filstruktur.



Figur 4.6: Filstruktur i src

- **test**
  Mappen har en fil, *forum.js* som innehåller 59 olika test av kontraktet *Forum.sol*. Testkoden finns att se i Bilaga G.



Figur 4.7: Filstruktur i test

4.2 Smart kontrakt

Kontraktet som beskrivs i detta avsnittet är *Forum.sol*. Det kan ses i Bilaga B. De två andra kontrakten, *EIP20interface.sol* och *migrations.sol* finns att läsa i Bilaga C respektive Bilaga D.

Det smarta kontraktet *Forum.sol* stödjer en valuta, SCONES, som finns definierad i kontraktet. När en nod migrerar kontraktet till blockkedjan för första gången bestäms totala antalet SCONES som initialt är i omlopp. Varje adress är mappad till ett heltal, *balances*. Detta representerar hur mycket respektive adress har av valutan. Initialt äger den som migrerar kontraktet alla SCONES. I kontraktet finns även funktioner som låter en nod skicka SCONES från sig själv till en annan nods adress. *Forum.sol* implementerar ERC-20-standarden. Utöver funktionaliteten enligt ERC-20-standarden, finns också en koppling mellan valutan och forumhantering i samma kontrakt. Funktionerna för forumhanteringen i kontraktet stödjer såväl forumägare som forumanvändare. För att undvika missförstånd hos läsaren listas följande begrepp:

- **ägarnod** - en ethereum-adress som har deklarerat sig som en forumägare genom det smarta kontraktet. Detta innebär nödvändigtvis inte att en ägarnod är en faktisk forumägare, utan enbart vill agera som en sådan genom det smarta kontraktets funktioner.
- **användarnod** - en ethereum-adress registrerad som användare hos en ägarnod.

- **forumnod** - representation av ett forum enligt det smarta kontraktet. En ägarnod äger en forumnod.

I kontraktet har varje användarnod en positiv heltalsvariabel, *karma*. Denna variabel justeras manuellt av den som äger forumet enligt det smarta kontraktet. Tanken med variabeln är att den ska spegla hur uppskattad en användare är.

Kontraktet *Forum.sol* är tänkt att appliceras för redan existerande forum. Användare på dessa forum kan agera näst intill precis som vanligt, givet att de på något sätt kan kopplas till en ethereum-adress på blockkedjan. För att intereagagera med valutan mer direkt (exempelvis se sitt saldo och begära ersättning) kan användaren dock ha nytta av en applikation. Ur ett perspektiv som användarnod kan klienten bland annat se sin användardata enligt det smarta kontraktet. Detta innefattar en användaradress, ett användarnamn och karma - för varje forumnod denne är medlem i på blockkedjan. Klienten kan också, för varje forumnod denne är medlem i, begära utbetalning (eller ersättning).

Vilken nod som helst kan deklarera sig som en ägarnod genom det smarta kontraktet. Detta innebär att en ägarnod kan vara vem som helst, och är alltså inte bunden till något specifikt forum. En ägarnod kan också lägga till ethereum-adresser som inte heller nödvändigtvis måste vara användare på faktiska, existerande forum.

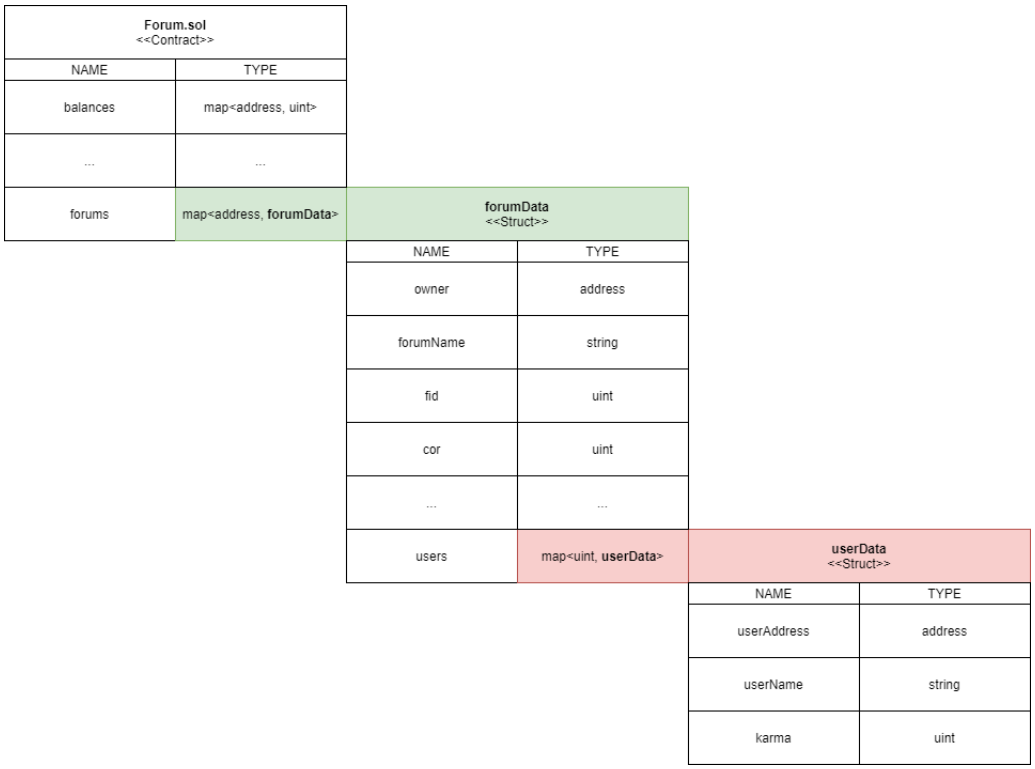
Ur egenskap för någon som vill agera som en ägarnod är tanken att data som finns på centraliserade databaser för redan existerande forum ska kunna användas. Exempelvis forumanvändares data. Funktionaliteten för en ägarnod i det smarta kontraktet är något annorlunda jämfört med funktionaliteten för en användarnod. Bland annat kan klienten då lägga till användarnoder i sin deklarerade forumnod på blockkedjan.

Utbetalningen användarnoderna kan begära bestäms med hjälp av två variabler som ägarnoden bestämmer. Den första variabeln kallas *cor* (cash out ratio), vilket är en ratio i SCONES per karma. Den andra variabeln är *karma*, för varje användarnod som är medlem i forumnoden. När en användarnod begär utbetalning skickas SCONES från ägarnoden till användarnoden.

En viktig del av funktionaliteten bakom utbetalningen är att en ägarnods saldo alltid ska kunna hantera ett scenario där alla användarnoder begär utbetalning. En ägarnods saldo får ju aldrig nå ett värde mindre än 0. Det finns felhantering för detta i kontraktet. Varje gång en ägarnod ändrar på en användarnods karma, eller läggar till en ny användarnod med en given karma, finns en check för detta. Dessutom finns en liknande check då en ägarnod justerar utbetalningsration, *cor*.

4.2.1 Komponenter

Figur 4.8 visar några av komponenterna (eller variablerna) och några av dess kopplingar. Nedanför figuren beskrivs dessa komponenter mer ingående. Detta avsnitt förklarar inte alla variabler. Däremot beskrivs de som är av mest relevans. För en intresserad läsare finns alla variabler att se i Bilaga B.



Figur 4.8: Variabler och structer i Forum.sol

Varje ethereum-adress har ett ett saldo i SCONES. För att göra detta möjligt finns en variabel, *balances*, deklarerad i *Forum.sol*. Variabeln mappar varje ethereum-adress till ett heltal. *balances* är implementerad via EIP20Interface.sol.

För att representera ett forums data i det smarta kontraktet (alltså en forumnod) finns en struct deklararerad i kontraktet, *forumData*. En variabel, *forums*, kan för varje unik ethereum-adress hålla en unik samling av variabler enligt *forumData*. Variabeln *forums* tar en adress som nyckel och mappas sedan till data enligt *forumData*. Alla adresser som kan mappas till *forumData* är alltså ägarnoder.

Följande lista innehåller några av variablerna i *forumData*:

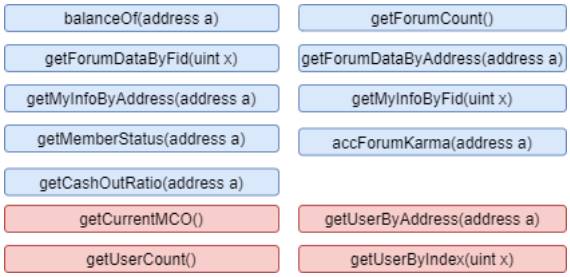
- ***owner*** - ägaradress.
- ***forumName*** - namn på forumet.
- ***fid*** - ett identifikationsnummer för forumet.
- ***cor*** - en utbetalningsratio, i SCONES per karma.
- ***users*** - en samling av användare. Variabeln kan för varje unik ethereum-adress hålla data som är representativ för forumanvändare, alltså en användarnod. Variabeln mappar adresser till data enligt structen *userData*.

Datan i structen *userData* är bestående av en användaradress, ett användarnamn och en användares karma.

4.2.2 Funktioner

Alla funktioner är publika och kan anropas av vilken nod som helst. En viktig aspekt hos ett flertal av funktionerna i det smarta kontraktet, är att funktionsanrop kan skilja sig beroende på vem avsändaren är, alltså vilken nod som anropar funktionen. Alla funktioner finns att se mer ingående i Bilaga B.

De flesta funktionerna i *Forum.sol* är läsfunktioner. De läser befintlig data på blockkedjan. I Figur 4.9 syns bland annat alla läsfunktioners namn. Ingen enskild läsfunktion kommer att beskrivas mer ingående. Istället beskrivs de mer allmänt. Alla variabler som beskrivs under avsnittet 4.2.1 kan på något sätt läsas med hjälp av en läsfunktion i det smarta kontraktet. Oavsett nod, är det viktigt att på ett enkelt sätt kunna läsa viss data på blockkedjan. Exempelvis en nods egna saldo i SCONES, eller vissa delar av den data som ska representera ett forum - alltså delar av *forumData*. Vissa läsfunktioner är skrivna att användas ur egenskap som antingen ägarnod eller användarnod. Läsfunktioner kan kasta fel beroende på vem avsändaren är. Figur 4.9 visar alla läsfunktioners namn, samt varje läsfunktions inparametrar. Blå färg innebär att den kan anropas av alla typer av användare. Röd färg innebär att den enbart kan anropas och ge lyckat resultat av ägarnoder.



Figur 4.9 Läsfunktioner i Forum.sol

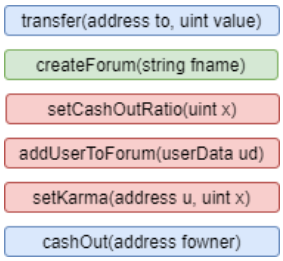
Skrivfunktionerna i det smarta kontraktet *Forum.sol* skriver ny data på blockkedjan. De mest relevanta funktionerna beskrivs kortfattat och simplifierat här. Alla finns dock att läsa mer ingående i Bilaga B. Följande lista inleder med namn och inparametrar på en skrivfunktion, följt av en kort beskrivning.

- **transfer(address to, uint value)**
  Skickar SCONES från avsändarens adress till en annan adress. Funktionen är tagen från EIP20Interface.sol.
- **createForum(string fname)**
  Avsändaren deklarerar sig som en ägarnod och får data enligt *forumData* förknippad med sig själv, alltså en forumnod. Lägger till sig själv i *users*.
- **setCashOutRatio(uint x)**
  Avsändare justerar sin forumnods utbetalningsratio.
- **addUserToForum(userData ud)**
  Lägger till en användarnod i avsändarens forumnod.

- **setKarma(address a, uint x)**
  Avsändaren justerar en användarnods *karma* i avsändarens forum.
- **cashOut(address fowner)**
  Avsändaren begär utbetalning av en ägarnod. Detta lyckas om:
  - avsändaren finns registrerad som en användarnod i ägarnodens forumnod.
  - *cor* i forumnoden är större än 0.
  - avsändaren har en *karma* i forumnoden större än 0.

Figur 4.10 förtydligar vilka skrivfunktioner en nod kan anropa. Följande gäller i figuren.

- Blå - Stödjer alla typer av användare.
- Grön - Stödjer enbart användare som inte har deklarerat någon forumnod. Alltså inga ägarnoder.
- Röd - Kan enbart anropas och ge lyckat resultat av ägarnoder.



Figur 4.10: Skrivfunktioner i Forum.sol

4.2.3 Tester av smart kontrakt

Testerna skrivs i Node.js och är skrivna för att säkerhetsställa att funktioner i *Forum.sol* fungerar sig som väntat. Alla funktioner testas minst en gång. Körningen av testen sker med hjälp av truffle-kommandot *truffle test*. Kommandot kör alla test, ett i taget. Se Bilaga H för rubriker av varje test, i korrekt ordning.

4.3 Databas

En MySQL-databas används för att skapa ett användarsystem. Den använder endast en tabell, *users*, som lagrar användare. Tabellen visas i Figur 4.32. *id* och *reg_date* genereras automatiskt. *username*, *address* och *password* anges manuellt, där *address* är en Ethereum-adress. Databasen hanteras av en Node.js MySQL-server, som skrivs i JavaScript.

Users	
Name	**Type + attributes**
id	INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
username	VARCHAR(30) NOT NULL UNIQUE KEY
address	VARCHAR(255) NOT NULL UNIQUE KEY
password	VARCHAR(255) NOT NULL
reg_date	TIMESTAMP DEFAULT CURRENT_TIMESTAMP

Figur 4.11: Tabellen Users i databas

4.3.1 Filer och funktioner

Koden kan ses i Bilaga I.

- **server.js**
  Hanterar hela databasen med hjälp av olika importerade filer och paket. De är bland annat *Express.js*, *connection.js* och *users.js*.

- **connection.js**
  Skapar en anslutning till MySQL-databasen.

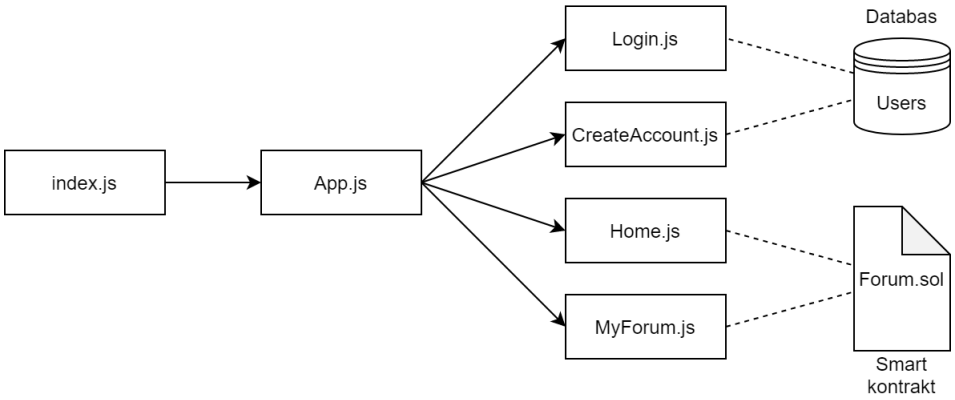
- **user.js**
  Skapar routes som tillåter applikationen att kommunicera med databasen. Kommunikationen sker via HTTP-requests. I funktionerna specificerar första parametern var en sådan request ska skickas. Ett exempel är om *Router.post('/new', function(request, response)* ska anropas från applikationen måste en post-request skickas till *http://localhost:3000/users/new*.

  - **Router.post('/new', function(request, response)**
    Funktionen används för att skapa en ny användare. Den kräver ett objekt med *username*, *address* och *password*, som hämtas via *request*. Lösenordet krypteras med en hash-funktion och skickas tillsammans med resterande information till databasen via en query. Ett resultat eller felmeddelande skickas sedan tillbaka via *response*.
  - **Router.post('/auth', function(request, response)**
    Funktionen används för att autentisera användarinformation vid inloggning. Den kräver ett objekt med *username* och *password*, som skickas via en post-funktion. Om informationen stämmer skickas användardata tillbaka via *response*, som funktionen hämtat med en query. Om informationen inte stämmer skickas istället ett felmeddelande tillbaka.

4.4 Applikation

Applikationen körs på en Node.js HTTP-server, som skrivs i JavaScript. Användargränssnittet är skapat med React, som renderar de HTML-komponenter som ska visas för användaren. En koppling till en nod i blockkedjenätverket är

skapad med hjälp av web3.js. Det gör att funktioner från ett smart kontrakt kan anropas i applikationen. Kopplingen till databasen är skapad av databasservern. Den tillåter requests som skickas med JavaScripts Fetch API. I Figur 4.33 visas hur filerna interagerar med varandra, samt om filerna kommunicerar med databasen eller det smarta kontraktet.



Figur 4.12: Illustration av applikationslogik

4.4.1 Användarperspektiv

En användare möts först av en login-sida där den väljer att logga in eller skapa ett konto. Väl inne på hemsidan visas information om de forum som användaren är med i. Användaren har där valet att använda en *cashout*-knapp för att få betalt för sin karma i SCONES. Om användaren istället vill agera forumägare kan ett forum skapas. Användaren kan då ses som en forumägare och har möjlighet att sätta en *cash out ratio* och lägga till användare. Hur applikationen ser ut visas i Bilaga A.

4.4.2 Anropa funktioner från smart kontrakt

Beroende på om en funktion från det smarta kontraktet skriver eller läser data, måste de anropas på olika sätt i applikationen. En funktion som skriver data anropas med *send()* och kräver en användaradress och ett värde i gas, som betalas för transaktionen. Om en funktion läser data används istället *call()*, den har inte samma krav, men kan behöva en användaradress för att hämta användarspecifik data.

4.4.3 Filer och funktioner

Koden kan ses i Bilaga J.

- **Funktioner i flera filer**
  Vissa funktioner används i mer än en fil.
  - **constructor()**
    Konstruktorn i React anropas innan komponenten monteras. Den används främst för att initiera state och för att binda funktioner. Att

binda en funktion tillåter funktionen att anropas från andra funktioner i klassen [21].

- **componentDidMount()**
  React-funktion som körs direkt efter komponenten är monterad. I den här applikationen anropas ofta setup-funktioner från componentDidMount() [21].
- **render()**
  React-funktion som används för att rendera Html-kod i applikationen. Funktionen krävs av en React-klasskomponent [21].
- **logChange(e, key)** Funktionen används för att spara input-värden till state-variabler. Parametern *e* används för att hämta input-värdet och *key* är namnet på variabeln som ska sättas.

- **index.html**
  index.html är en simpel Html-fil där meta-data och title sätts i head. Eftersom applikationen använder React blir body relativt tom. I body skapas ett div-element som React kan rendera element i och build.js importeras som script.

- **build.js**
  En fil som genereras av JavaScript-biblioteket Webpack. Filen kombinerar alla JavaScript filer i applikationen, men hämtar bara de filer som är nödvändiga för att ladda hemsidan.

- **index.js**
  Skapar en *BrowserRouter* som renderas i *index.html*. BrowserRouter är en del av React Router som möjliggör rendering av flera olika sidor. I routern renderas komponenten *App* som skapas i *App.js*.

- **App.js**
  Skapar komponenten *App*. I komponenten renderas en Switch med Routes, vilka är en del av React Router. Det specifierar vilken komponent som ska renderas beroende på URL. Ett exempel är att */Home* ska rendera komponenten *Home* från *home.js*.

- **Login.js**
  Skapar komponenten *Login*. Den renderar en form som accepterar ett användarnamn och ett lösenord. Med hjälp av dess funktioner autentiseras login-informationen mot databasen.

  - **handleSubmit(event)**
    Funktionen anropas vid submit av form. Den skickar sedan det inmatade användarnamnet och lösenordet till databasfunktionen *Router.post('/auth', function(request, response)* som autentiserar login-informationen.
  - **HistoryHook(props)**
    Funktionen skickar användaren till *Home* och anropas när användaren blivit autentiserad. Parametern *props* är ett objekt som skickar med data.

- **CreateAccount.js**
  Skapar komponenten *CreateAccount*. Den renderar en form som accepterar

ett användarnamn, en address och ett lösenord. Med hjälp av dess funktioner skickas informationen till databasen för att skapa en användare.

  - **handleSubmit(event)**
    Funktionen anropas vid submit av form. Den skickar sedan det inmatade användarnamnet, addressen och lösenordet till databasfunktionen *Router.post('/new', function(request, response, next)* som autentiserar login-informationen.

- **Home.js**
  Skapar komponenten *Home*. Den renderar information om användaren och en lista med samtliga forum användaren är med i. Listan innehåller information om varje forum och en knapp som låter användaren få betalt för sin Karma i SCONES.

  - **constructor(props)**
    I konstruktorn initieras en koppling till det smarta kontraktet med hjälp av web3.js. Det gör att samtliga funktioner från det smarta kontraktet kan anropas.
  - **setup()**
    Funktionen hämtar information som ska visas på hemsidan från det smarta kontraktet. Den anropas från *componentDidMount()* när komponenten renderas.
  - **cashOut(forumAddress)**
    Funktionen betalar användaren för sin *karma* med hjälp av det smarta kontraktet och anropas vid knapptryck. Parametern *forumAddress* skickas med vid funktionsanropet och skiljer beroende på forum i listan.

- **MyForum.js**
  Skapar komponenten *MyForum*. Den renderar olika element beroende på om användaren äger ett forum eller inte. Om användaren inte äger ett forum renderas en form som accepterar ett forumnamn, för att skapa ett forum. Om användaren äger ett forum renderas istället information om forumet och två olika forms. Den första formen accepterar ett heltal för att sätta *cor* (cash out ratio). Den andra formen accepterar ett användarnamn och en adress för att lägga till en användare till forumet.

  - **setup()**
    Funktionen kollar om användaren äger ett forum och hämtar i så fall relevant information. Den anropas från *componentDidMount()* när komponenten renderas.
  - **setCashOutPrice(event)**
    Funktionen skickar det inmatade värdet till en funktion i det smarta kontraktet som ändrar*cor* (cash out ratio).
  - **addUser(event)**
    Funktionen skickar det inmatade användarnamnet och adressen till en funktion i det smarta kontraktet, som lägger till användaren i forumet.

– **createForum(event)**
Funktionen skickar det inmatade namnet till en funktion i det smarta kontraktet som skapar ett nytt forumobjekt.

5 Genomförande

Systemet byggdes inte i exakt den här ordningen, men om systemet ska återskapas är det ett rimligt tillvägagångssätt.

Först av allt behöver nödvändig programvara finnas på plats. Alltså truffle, ganache, och Node.js. Genom att starta ganache simuleras en blockkedja på det lokala nätverket. Innan utveckling av det smarta kontraktet påbörjas kan truffle generera en basstruktur genom kommandot truffle init. Nästa steg är att påbörja utvecklingen av de smarta kontrakten. Dessa kan kompileras med hjälp av truffle-kommandot truffle compile. Vid kompilering skapar truffle en build-mapp med JSON-representationer av de smarta kontrakten. För att testa funktionerna i de smarta kontraktet skrivs tester i test-mappen. Dessa tester kan köras med hjälp av truffle genom kommandot truffle test. För att lyckas kommunicera med smarta kontrakt på blockkedjan måste de först migreras till blockkedjan. Detta sker med hjälp av truffle-kommandot truffle migrate. När kontrakten migreras får de varsin egen unik adress, vilket kommer användas av applikationen för att kommunicera med kontrakten på blockkedjan. Då någonting ändras eller läggs till i kontrakten behöver både truffle compile och truffle migrate anropas på nytt. För en att skapa en simpel applikation behövs endast en HTML-fil och en JavaScript-fil.

För att skapa en databas krävs endast att MySQL är installerat, men en databasserver har fler krav. Servern behöver vissa Node-dependencies, där de viktigaste är MySQL och Express.js. Databasen kan skapas i terminalen genom MySQL. Servern behöver skapa en koppling till databasen och routes för att kommunicera med applikationen.

Innan applikationen kan byggas behövs vissa Node-dependencies. De viktigaste är React och web3.js. De anges i filen *package.json* och installeras i projektet genom att köra *npm install* i CLI. En koppling till de smarta kontrakten kan då skapas med hjälp av web3.js. React-funktioner kan sedan användas för att koppla ett användargränssnitt till det smarta kontraktet. Applikationen kan sedan vidareutvecklas till en flersidig applikation med React Router.

6 Resultat

Ett belöningssystem till forum har skapats. Det körs smarta kontrakt på en lokal simulering av Ethereums blockkedja. En digital valuta, SCONES, finns definierad i kontraktet *Forum.sol*. SCONES kan belönas till användare som delar lösningar av god kvalitet. Värdet av variabeln *karma* varje användarnod har kan kopplas till uppskattningsmekanismer, vilka kan reflektera lösningars kvalitet. Systemet har även ett användargränssnitt i form av en webbapplikation.

6.1 Smarta kontrakt

Systemet är byggt med tre stycken smarta kontrakt. Det mest relevanta kontraktet är *Forum.sol*.

Koppling mellan forumanvändare och en digital valuta finns utformad och kan användas. I *Forum.sol* finns bland annat en valuta, SCONES, definierad. Valutan är decentraliserad eftersom den lagras på en blockkedja. Valutan följer ERC-20-standarden, vilket bland annat innebär att alla noder i Ethereums blockkedja kan ha ett saldo av valutan, och att noder kan skicka valutan mellan varandra. Utöver ERC-20-funktionaliteten finns också funktionalitet för forumhantering i samma kontrakt. Noderna kan agera som användarnoder och ägarnoder, vilka representerar faktiska forumanvändare respektive forumägare. Givet att en användarnod finns registrerad som användare i en ägarnods forumnod, kan användarnoden begära utbetalning från ägarnoden genom det smarta kontraktet. Utbetalningen sker i SCONES från ägarnoden till användarnoden.

Användare kan få ersättning genom redan existerande uppskattningsmekanismer. Genom en variabel, *karma*, i det smarta kontraktet, kan användarnoder tilldelas ett uppskattningsvärde i varje forumnod de är registrerad i. Det är alltid ägarnoden som justerar sina användarnoders *karma*. Variabeln *karma* kan kopplas till redan existerande uppskattningsmekanismerna på faktiska forum. Ägarnoden bestämmer också en utbetalningsratio, i SCONES per karma. Variabeln *karma* har en direkt koppling till SCONES genom utbetalningsration och funktionen *cashOut*. En användarnod kan genom cashOut-funktionen begära utbetalning baserat på hur mycket karma användarnoden blivit tilldelad, samt forumnodens utbetalningsratio.

För att premiera lösningar av god kvalitet måste en ägarnod själv anpassa och ansvara för hur redan existerande uppskattningsmekanismer på forum ska utnyttjas.

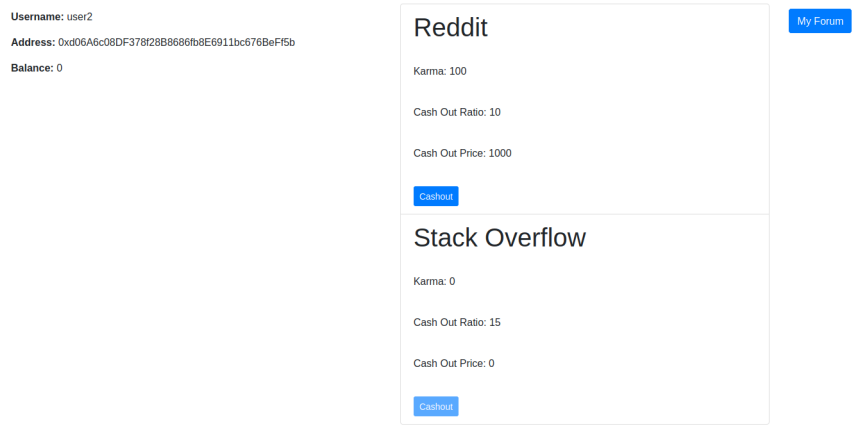
När kontraktet migreras till blockkedjan kan det tillgås och användas av vem som helst. Vem som helst kan då genom det smarta kontraktet deklarera sig som en ägarnod och registrera ethereum-adresser som användarnoder. *Forum.sol* är alltså inte på något sätt kopplat till ett specifikt forum eller specifika användare.

6.2 Applikation

En applikation har skapats för att tillföra ett användargrässnitt till systemet. Applikationen körs på en Node.js HTTP-server. För att interagera med det smarta kontraktet används Web3.js, som skapar en HTTP-anslutning till en lokal nod. Applikationen kan då anropa funktioner från det smarta kontraktet. För att koppla det till ett användargränssnitt används React, som från servern renderar HTML-kod i en HTML-fil. Filen representerar det som visas i webbläsaren, samt styr vad användaren kan interagera med.

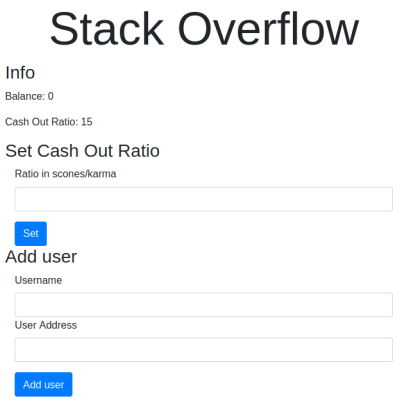
Applikationens användarsystem hanteras av en Node.js MySQL-server kopplad till en MySQL-databas. Express.js används för att skapa routes. Dessa routes kan sedan användas av applikationen för att interagera med databasen. Databasen lagrar bland annat användares ethereum-addresser, vilka sedan används för att interagera med det smarta kontraktet. På så sätt är centraliserad data kopplad med decentraliserad data.

I applikationen går det att agera forumanvändare och forumägare. En forumanvändare kan se information om de forum den är med i och har möjlighet att få betalt för sin karma. Figur 6.1 visar hur denna vy ser ut.



Figur 6.1: Vy för forumanvändare

En forumägare har en annan vy. Ägaren kan se information om sitt forum och har möjligheten att sätta en *Cash Out Ratio*, samt lägga till en användare. Vyn kan ses i Figur 6.2.



Figur 6.2: Vy för forumägare

7 Diskussion

I följande kapitel diskuteras systemets val, förbättringsområden, vidareutvecklingsmöjligheter och hållbarhet.

7.1 Blockkedjor

Valet av plattform för utveckling av smarta kontrakt blev Ethereum. Att utveckla liknande system med hjälp av andra plattformar kan vara av intresse då man vill göra en jämförelse mellan systemens för- och nackdelar beroende på vilken plattform som används.

En annan jämförelse som kan vara av intresse är valet att implementera ett liknande system utan att använda blockkedjeteknik över huvud taget. Nyttan av det faktum att systemet utnyttjar blockkedjeteknik har inte varit i något särskilt fokus. Däremot är det kanske en relativt viktig aspekt för att i slutändan döma produkten i sin helhet.

På Ethereums blockkedja förekommer en kostnad vid transaktioner som skriver ny data på blockkedjan. Om ett liknande system ska lanseras bör hänsyn tas till hur mycket gas de olika funktionerna det smarta kontraktet kostar, samt undersöka olika optimeringsmöjligheter för att reducera gaskostnaden.

7.2 Digitala valutor

Smarta kontrakt för Ethereums blockkedja kan utnyttja funktioner för andra smarta kontrakt. Valutan som finns definierad i det smarta kontraktet, SCONES, fungerar likt många andra redan existerande valutor på Ethereums blockkedja, bortsett från kopplingen till forumhanteringen i samma smarta kontrakt. En annan systemlösning hade kunnat vara att utnyttja en sådan redan existerande valuta, istället för att skapa en helt ny.

7.3 Smart kontrakt

Dagens internet består av många olika forum. Reddit och StackOverflow är bara två exempel. Ofta kan användare i dessa forum få uppvoteringar, förtjäna medaljer, och erhålla andra typer av uppskattningsmekanismer. Detta kan dock skilja sig från forum till forum. En ägarnod kan vara i egenskap av vem som helst. Det skulle kunna vara den faktiska ägaren av Reddit eller Stackoverflow. Det skulle också kunna vara någon som bara skapar en tråd i ett forum, eller någon som bara övervakar en tråd. Flera av de större forumen har API'er som gör det möjligt att få ut lämplig data, så som forumtrådar, användare med deras svar och uppvoteringar med mera. Teoretiskt sett, behöver inte ägarnoden vara anslutet till något forum över huvud taget. Detta eftersom vem som helst kan utnyttja det smarta kontraktets funktioner. Detta kan innebära att en del av syftet går förlorat. Istället för incitament för problemlösning, kan det bli incitament för att få uppskattning på internet lite hur som helst. Exempelvis kan det handla om uppvoteringar på instagram.

Lösningsdesignen av det smarta kontraktet fyller vissa syften. Däremot innebär

det inte att denna lösningsdesignen är den enda i relation till syftet. Därför är det av intresse att undersöka och jämföra olika lösningar. Bland annat kan funktionen *createForum* vara värd att diskutera. Den tillåter nämligen bara att lyckas en gång per ethereum-adress. Det innebär att en adress inte kan deklarera fler än en forumnod. Att ge adresser möjlighet till att deklararera sig som ägarnoder flera gånger, för flera forumnoder, hade inneburit en annan design med andra möjligheter, vilket kan vara intressant att jämföra.

Hur mycket och på vilket sätt ägarnoder väljer att dela ut karman, är upp till de själva. De bestämmer också utbetalningsration. En ägarnod kan då själv anpassa sig till vilka uppskattningsmekanismer den vill använda sig av för att bestämma användarnoders karma. Det är alltså alltid upp till ägarnoden att anpassa sig efter vilka uppskattningsmekanismer som är relevanta och då också själv ansvara för att premiera lösningar av god kvalitet. Det kan exempelvis avgöras genom att enbart räkna uppvoteringar för varje användare i en tråd på ett faktiskt forum (exempelvis Reddit). Det kan också avgöras med hjälp av flera andra uppskattningsvariabler på det faktiska forumet. Fördelen är att det är öppet att använda redan existerande uppskattningsmekanismer på olika, anpassade sätt. En nackdel med detta dock, kan möjligtvis vara att ägarnoden ges för mycket ansvar över både utbetalningsration och användarnodens karma.

En annan aspekt av det smarta kontraktet är hur användarnoder läggs till i forumnoder. Det är en ägarnod som lägger till en användarnod. En annan lösning hade varit om användarnoden lägger till sig själv i forumnoden istället. Ytterligare en lösningen kan vara att låta en handskakning ske. Dvs, att användarnod och ägarnod godkänner varandra, och först då placeras användarnoden i forumnoden.

En aspekt som är viktig vad gäller data på blockkedjor, är dess synlighet. Teoretiskt sett, är inte läsfunktioner i smarta kontrakt nödvändigtvis avgörande. All data är ju de facto synlig i blockkedjan. Det smarta kontraktet blir dock en genväg för att enklare läsa datan. De flesta läsfunktionerna i *Forum.sol* kan kasta fel - men inte alla (på grund av tidsbrist). I en färdig produkt borde antingen detta finnas på plats, eller inga felmeddelanden vid läsfunktioner överhuvud taget. Att försöka läsa data som inte existerar genom ett smart kontrakt kommer aldrig returnera null. Istället returneras 0-värden. En fråga som kan ställas är om ansvaret för felhantering vid läsning enbart ska ligga på applikationssidan istället för i det smarta kontraktet.

7.3.1 Applikation

Det bestämdes redan i början av arbetet att en applikation skulle byggas, men det var oklart exakt vad den skulle innehålla. Tidigt var idén att bygga ett forum, men det hade till stor del begränsat den digitala valutan och det smarta kontraktet till det specifika forumet. Att begränsa en digital valuta till en applikation går till viss del emot decentralisering. Om applikationen tas ner förlorar valutan sitt värde, om inte en ny applikation skapas som använder samma valuta. Det finns därför en fördel att använda valutan, med smart kontrakt, i flera applikationer. Om en applikation går ner har kryptovalutan ändå kvar sitt värde i de andra applikationerna.

Att bygga ett tillägg till en webbläsare var ett bra alternativ. På så sätt hade användare endast behövt ladda ner tillägget och logga in för att ta del av belöningssystemet. Användaren hade på så sätt inte behövt lära sig något nytt, utan tillägget hade smidigt gjort att användaren fått betalt av att använda sitt favoritforum som vanligt. Problemet med att bygga ett sådant tillägg är att forumen de ska användas på måste dela med sig av sin data. Det är alltså data som hade varit svår att få tag i. Dock hade det här alternativet varit intressant att genomföra i ett vidarearbete.

Valet blev att bygga en applikation som samlar information och funktioner hos alla forum användaren är med i. Det kräver inte att forumen faktiskt existerar, utan endast att de kan skapas som forumobjekt på blockkedjan. På så sätt kunde funktionaliteten av det smarta kontraktet demonstreras tillsammans med ett centraliserat användarsystem, som lagras på en databas. Centraliserade delar ska normalt sett inte existera i en decentraliserad applikation. Anledningen att det används i applikationen är på grund av att de flesta forum använder en databas. Applikationen demonstrerar därmed hur vårt decentraliserade system kan användas tillsammans med en centraliserad databas.

7.3.2 Decentralisering

Applikationen kan tolkas som decentraliserad eftersom den använder ett smart kontrakt för backend-logik. Resonemanget är då att applikationen endast tillför ett användargränssnitt till en redan existerande backend. Det som går emot resonemanget är att gränssnittet krävs för att interagera med det smarta kontraktet. Om HTTP-servern stängs av går det inte att interagera med det smarta kontraktet. En ny applikation kan dock skapas för att interagera med det smarta kontraktet.

Att en databas används tyder på centralisering. Anledning att det inte går emot decentralisering är att databasen inte lagrar något av värde till det smarta kontraktet. I applikationen används databasen endast för ett användarsystem, som är kopplat till applikationen men inte det smarta kontraktet. Om systemet implementeras på ett forum blir det viktigt att allt av värde som lagras på databasen även lagras på det smarta kontraktet. Om karma är baserat på uppvoteringar borde omvandlingen alltså ske direkt.

7.4 Hållbarhet och etik

Teknisk och vetenskaplig innovation kan tänkas spela en roll för att lösa globala problem, och desto bättre kan det bli om fler kan tänka sig delge lösningar. Genom att skapa incitament för att lösa problemen, ökar sannolikheten att fler hjälper till att lösa alla möjliga problem, även de globala. Systemet som beskrivs i detta projekt kan användas för att skapa mer incitament där lösningar av god kvalitet premieras. Därför går det att argumentera för att systemet kan användas i hållbart ändamål.

Även om det smarta kontraktet i detta projekt teoretiskt sett kan användas i ett hållbart ändamål, finns det en brist som borde diskuteras. Eftersom det smarta kontraktet kan användas brett på sådant vis att det inte används till kvalitativ problemlösning, kan syftet med det gå förlorat. Exempelvis kan det användas för att koppla valutan SCONES till uppvoteringar på instagram-bilder. Vilka etiska aspekter det innebär borde diskuteras. Kontraktet kan också tillämpas på forum där illvilja är i fokus. Exempelvis kan då också farliga idéer premieras.

8 Slutsats

Det teoretiska bidraget är en modell som demonstrerar hur smarta kontrakt kan utnyttjas för att utforma en valuta med ett nischat syfte. Det praktiska bidraget är ett systemet som utnyttjar blockkedjeteknik. Detta genom att utveckla en applikation som kan intereagera med ett smart kontrakt på en blockkedja. Incitament för att delge lösningar på problem skapas genom att låta användare på forum använda det smarta kontraktet. Där finns funktionalitet för ersättning i form av en digital valuta implementerad. Denna funktionalitet kan baseras på ett forums uppskattningsmekanismer. Där kan en ägarnod anpassa och ansvara för vilka uppskattningsvariabler som är relevanta i en kvalitativ bemärkelse. Det smarta kontraktet kan dock användas brett och kan således också användas till annat än just problemlösning. Dessutom kan eventuella förbättringar av systemet utforskas.

Referenser

[1] M. Swezey, “What is a Smart Contract?” 2017. [Online]. Tillgänglig: https://medium.com/pactum/what-is-a-smart-contract-10312f4aa7de, hämtad: 2020-06-05.

[2] M. G. Solomon, *Ethereum For Dummies*. Hoboken, NJ, USA: Wiley, 2019.

[3] M. Condon, “Getting Up to Speed on Ethereum,” 2017. [Online]. Tillgänglig: https://medium.com/@mattcondon/getting-up-to-speed-on-ethereum-63ed28821bbe, hämtad: 2020-06-05.

[4] A. Tar, “Proof-of-Work, Explained,” 2018. [Online]. Tillgänglig: https://cointelegraph.com/explained/proof-of-work-explained, hämtad: 2020-06-05.

[5] J. Frankenfield, “Gas (Ethereum),” 2019. [Online]. Tillgänglig: https://www.investopedia.com/terms/g/gas-ethereum.asp, hämtad: 2020-06-05.

[6] M. William, “ERC-20 Tokens, Explained,” 2018. [Online]. Tillgänglig: https://cointelegraph.com/explained/erc-20-tokens-explained, hämtad: 2020-06-05.

[7] Truffle Blockchain Group, “SWEET TOOLS FOR SMART CONTRACTS,” 2020. [Online]. Tillgänglig: https://www.trufflesuite.com/, hämtad: 2020-06-05.

[8] ——, “TRUFFLE - SMART CONTRACTS MADE SWEETER,” 2020. [Online]. Tillgänglig: https://www.trufflesuite.com/truffle, hämtad: 2020-06-05.

[9] ——, “Ganache - ONE CLICK BLOCKCHAIN,” 2020. [Online]. Tillgänglig: https://www.trufflesuite.com/ganache, hämtad: 2020-06-05.

[10] Ethereum, “Solidity,” 2020. [Online]. Tillgänglig: https://solidity.readthedocs.io/en/v0.6.8, hämtad: 2020-06-05.

[11] Refsnes Data, “HTML Introduction,” 2020. [Online]. Tillgänglig: https://www.w3schools.com/html/html_intro.asp, hämtad: 2020-06-16.

[12] I. Kantor, “An Introduction to JavaScript,” 2020. [Online]. Tillgänglig: https://javascript.info/intro, hämtad: 2020-06-16.

[13] Khan Academy, “What’s a JS library?” 2020. [Online]. Tillgänglig: https://www.khanacademy.org/computing/computer-programming/html-css-js/using-js-libraries-in-your-webpage/a/whats-a-js-library, hämtad: 2020-06-16.

[14] OpenJS Foundation, “About Node.js,” 2020. [Online]. Tillgänglig: https://nodejs.org/en/about, hämtad: 2020-06-05.

[15] Refsnes Data, “Node.js NPM,” 2020. [Online]. Tillgänglig: https://www.w3schools.com/nodejs/nodejs_npm.asp, hämtad: 2020-06-05.

[16] Ethereum, “web3.js - Ethereum JavaScript API,” 2016. [Online]. Tillgänglig: https://web3js.readthedocs.io/en/v1.2.6, hämtad: 2020-06-05.

[17] Facebook Inc., “React - A JavaScript library for building user interfaces,” 2020. [Online]. Tillgänglig: https://reactjs.org, hämtad: 2020-06-05.

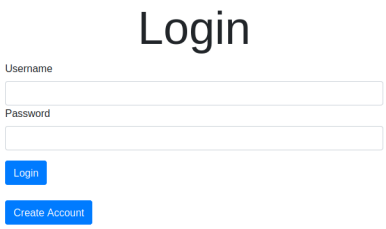
[18] Manjunath M. och M. Wanyoike, “React Router v5: The Complete Guide,” 2020. [Online]. Tillgänglig: https://www.sitepoint.com/react-router-complete-guide, hämtad: 2020-06-05.

[19] Refsnes Data, “Node.js MySQL,” 2020. [Online]. Tillgänglig: https://www.w3schools.com/nodejs/nodejs_mysql.asp, hämtad: 2020-06-05.

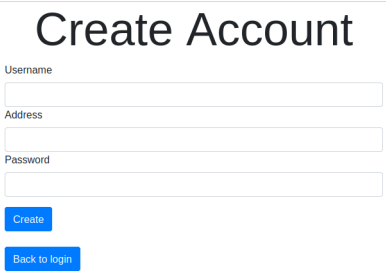
[20] F. Vogelsteller och V. Buterin, “EIPs,” 2019. [Online]. Tillgänglig: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md, hämtad: 2020-03-02.

[21] Facebook Inc., “React.Component,” 2020. [Online]. Tillgänglig: https://reactjs.org/docs/react-component.html, hämtad: 2020-06-05.

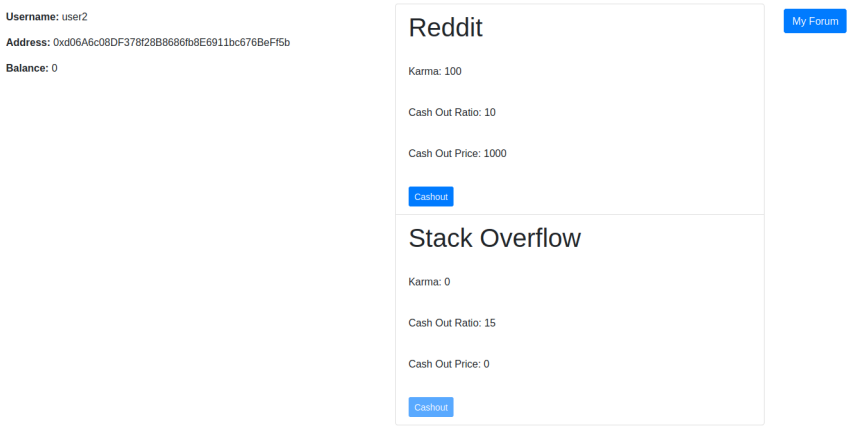
Bilaga A: Applikationens användargränssnitt



Figur A.1: Login

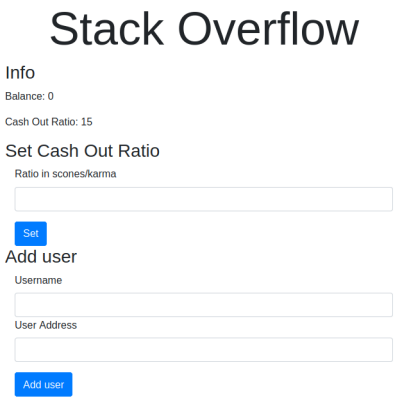


Figur A.2: Create Account



Figur A.3: Home

Figur A.4: My Forum - Create Forum



Figur A.5: My Forum - Owner

Bilaga B: Forum.sol

```
/*
Implements EIP20 token standard: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
.*/
pragma solidity ^0.5.0;

import "./EIP20Interface.sol";

contract Forum is EIP20Interface {
  // The coin
  uint256 constant private MAX_UINT256 = 2**256 - 1;
  mapping (address => uint256) public balances;
  mapping (address => mapping (address => uint256)) public allowed;

  string public name;
  uint8 public decimals;
  string public symbol;

  // Forum
  mapping (address => forumData) private forums;    // An address can hold a forum
  mapping (uint => address) private fidOwner;       // A forumID (fid) is mapped to an owner of a forum with this fid.
  uint256 private forumId;                          // Increments when a new forum is created
  mapping (address => uint256) addressForumId;

  struct forumData{
    address owner;
    string forumName;
    uint256 fid;
    uint256 userCount;
    uint256 cor;
    mapping(uint256 => userData) users;
    mapping(address => uint256) userKey;
    mapping(address => bool) userExists;
  }

  struct userData{
    address userAddress;
    string userName;
    uint256 karma;
  }

  constructor (
    uint256 _initialAmount,
    string memory _tokenName,
    uint8 _decimalUnits,
    string memory _tokenSymbol
  ) public {
    balances[msg.sender] = _initialAmount;
    totalSupply = _initialAmount;
    name = _tokenName;
    decimals = _decimalUnits;
    symbol = _tokenSymbol;
    forumId = 1;
  }

  // EIP20Interface functions...
```

```
56    function transfer(address _to, uint256 _value) public returns (bool
          success) {
57      require(balances[msg.sender] >= _value);
58      balances[msg.sender] -= _value;
59      balances[_to] += _value;
60      emit Transfer(msg.sender, _to, _value); //solhint-disable-line
            indent, no-unused-vars
61      return true;
62    }
63
64    function transferFrom(address _from, address _to, uint256 _value)
          public returns (bool success) {
65      uint256 allowance = allowed[_from][msg.sender];
66      require(balances[_from] >= _value && allowance >= _value);
67      balances[_to] += _value;
68      balances[_from] -= _value;
69      if (allowance < MAX_UINT256) {
70        allowed[_from][msg.sender] -= _value;
71      }
72      emit Transfer(_from, _to, _value); //solhint-disable-line indent,
            no-unused-vars
73      return true;
74    }
75
76    function balanceOf(address _owner) public view returns (uint256
          balance) {
77      return balances[_owner];
78    }
79
80    function approve(address _spender, uint256 _value) public returns (
          bool success) {
81      allowed[msg.sender][_spender] = _value;
82      emit Approval(msg.sender, _spender, _value); //solhint-disable-line
            indent, no-unused-vars
83      return true;
84    }
85
86    function allowance(address _owner, address _spender) public view
          returns (uint256 remaining) {
87      return allowed[_owner][_spender];
88    }
89    // .. end of EIP20Interface functions
90
91
92
93    // Forum functions...
94
95    function createForum(string memory _fname) public returns (bool
          success){
96      uint256 a = forums[msg.sender].fid;
97      uint256 b = addressForumId[msg.sender];
98      if((a != 0) && (a == b)){
99        revert("You already own a forum.");
100     }
101     forums[msg.sender] = forumData(msg.sender, _fname, forumId, 0, 0);
102     forums[msg.sender].users[0] = userData(msg.sender, _fname, 0);
103     forums[msg.sender].userExists[msg.sender] = true;
104     forums[msg.sender].userKey[msg.sender] = 0;
105     forums[msg.sender].userCount ++;
106     addressForumId[msg.sender] = forumId;
107     fidOwner[forumId] = msg.sender;
108     forumId = forumId + 1;
```

```
109       emit CreateForum(true, msg.sender, _fname, forums[msg.sender].fid);
110       return (true);
111     }
112     function getForumCount() public view returns (uint256 id){
113       return (forumId-1);
114     }
115
116     function getForumDataByFid(uint256 _fid)
117     public view returns (address forumOwner,
118                          string memory forumName,
119                          uint256 fid,
120                          uint256 cor){
121                          address owner = fidOwner[_fid];
122       if(owner == address(0x0))
123         revert("Forum does not exist.");
124       string memory fName = forums[owner].forumName;
125       uint256 id = forums[owner].fid;
126       uint256 fcor = forums[owner].cor;
127       return(owner, fName, id, fcor);
128       }
129
130     function getForumDataByAddress(address _owner)
131     public view returns (address forumOwner,
132                           string memory forumName,
133                          uint256 fid,
134                          uint256 cor){
135       if(addressForumId[msg.sender] == 0){
136         revert("Forum does not exist.");
137       }
138       string memory fName = forums[_owner].forumName;
139       uint256 id = forums[_owner].fid;
140       uint256 fcor = forums[_owner].cor;
141       return(_owner, fName, id, fcor);
142     }
143
144         //Max cash out, that is if all users were to cash out.
145     function getCurrentMCO() public view returns (uint256 max){
146       uint256 res = accForumKarma(msg.sender) * forums[msg.sender].cor;
147       return res;
148     }
149
150     function addUserToForum(address _userAddress,
151                             string memory _userName,
152                             uint256 _karma)
153     public returns (bool success){
154       if(forums[msg.sender].owner != msg.sender){
155         revert("You need to create a forum first.");
156       }
157       if(forums[msg.sender].userExists[_userAddress] == true){
158         revert("User already exists.");
159       }
160       if(( getCurrentMCO() + ( forums[msg.sender].cor * _karma)) >
              balanceOf(msg.sender))
161         revert("User is given too much karma. Your balace might not be
                 able to handle it."); //t
162       uint256 newUserCount = forums[msg.sender].userCount;
163       forums[msg.sender].users[newUserCount] = userData(_userAddress,
              _userName, _karma);
164       forums[msg.sender].userExists[_userAddress] = true;
165       forums[msg.sender].userKey[_userAddress] = newUserCount;
166       forums[msg.sender].userCount++;
167       emit AddUserToForum(_userAddress, _userName, _karma);
```

```
168       return (true);
169     }
170
171     function setKarma(uint256 _karma, address _userAddress)
172     public returns (bool success){
173       if(forums[msg.sender].owner == address(0x0)){
174         revert("You need to create a forum first."); //t
175       }
176       if(!forums[msg.sender].userExists[_userAddress]){
177         revert("This user does not exist in your forum."); //t
178       }
179       if(( getCurrentMCO() + ( forums[msg.sender].cor * _karma)) >
              balanceOf(msg.sender))
180         revert("User is given too much karma. Your balace might not be
              able to handle it."); //t
181       uint256 ukey = forums[msg.sender].userKey[_userAddress];
182       forums[msg.sender].users[ukey].karma = _karma;
183       emit SetKarma(msg.sender, _userAddress, _karma);
184       return true;
185     }
186
187     function getUserByAddress(address _userAddress)
188     public view returns (address usrAddress,
189                          string memory usrName,
190                          uint256 usrKarma){
191       if(forums[msg.sender].owner == address(0x0)){
192         revert("You need to create a forum first.");
193       }
194       if(!forums[msg.sender].userExists[_userAddress]){
195         revert("This user does not exist.");
196       }
197       uint256 uKey = forums[msg.sender].userKey[_userAddress];
198       address uaddress = forums[msg.sender].users[uKey].userAddress;
199       string memory uname = forums[msg.sender].users[uKey].userName;
200       uint256 ukarma = forums[msg.sender].users[uKey].karma;
201       return (uaddress, uname, ukarma);
202       }
203
204     function getUserByIndex(uint256 _ucount)
205     public view returns (address usrAddress,
206                          string memory usrName,
207                          uint256 usrKarma){
208       if(forums[msg.sender].owner == address(0x0)){
209         revert("You need to create a forum first");
210       }
211       if(forums[msg.sender].users[_ucount].userAddress == address(0x0)){
212         revert("This user does not exist.");
213       }
214       address uaddress = forums[msg.sender].users[_ucount].userAddress;
215       string memory uname = forums[msg.sender].users[_ucount].userName;
216       uint256 ukarma = forums[msg.sender].users[_ucount].karma;
217       return (uaddress, uname, ukarma);
218     }
219
220     function getUserCount() public view returns (uint256 ucount){
221       return forums[msg.sender].userCount;
222     }
223
224     function getMyInfoByAddress(address _admin)
225     public view returns (address fAddress,
226                          address myAddress,
227                          string memory myUserName,
```

```
                        uint256 myKarma){

    bool exists = forums[_admin].userExists[msg.sender];
    if(!exists)
      revert("You are not a member in this forum.");
    uint256 key = forums[_admin].userKey[msg.sender];
    string memory uname = forums[_admin].users[key].userName;
    uint256 karma = forums[_admin].users[key].karma;
    return (_admin, msg.sender, uname, karma);
  }

  function getMyInfoByFid(uint256 _fid)
  public view returns (uint256 fID,
                        address myAddress,
                        string memory myUserName,
                        uint256 myKarma){

    address admin = fidOwner[_fid];
    bool exists = forums[admin].userExists[msg.sender];
    if(!exists)
      revert("You are not a member in this forum.");
    uint256 key = forums[admin].userKey[msg.sender];
    string memory uname = forums[admin].users[key].userName;
    uint256 karma = forums[admin].users[key].karma;
    return (_fid, msg.sender, uname, karma);
  }

  function getMemberStatus(uint256 _fid)
  public view returns (uint256 fID, bool success){
    address admin = fidOwner[_fid];
    return (_fid, forums[admin].userExists[msg.sender]);
  }

  function accForumKarma(address _forum) public view returns (uint256
      totalKarma) {
    uint256 numUsers = forums[_forum].userCount +1;
    uint256 accKarma = 0;
    for(uint256 i = 0; i<numUsers; i++){
      accKarma += forums[_forum].users[i].karma;
    }
    return accKarma;
  }

  function setCashOutRatio(uint256 _cor) public returns (uint256 fcor){

    uint256 accKarma = accForumKarma(msg.sender);
    if(accKarma * _cor > balanceOf(msg.sender))
      revert("Too high. Aquire more scones or try a lower COR.");
    forums[msg.sender].cor = _cor;
    emit SetCashOutPrice(msg.sender, forums[msg.sender].cor);
    return forums[msg.sender].cor;
  }

  function getCashOutRatio(address _forum)
  public view returns (uint256 fcor){

    return forums[_forum].cor;
  }

  function cashOut(address _forum) public returns (bool success){

    if (forums[_forum].fid == 0)
```

```
289       revert("Forum does not exist");
290
291     if (forums[_forum].cor == 0)
292       revert("Cash-out price is 0 scones.");
293     if(!forums[_forum].userExists[msg.sender])
294       revert("You are not a member in this forum.");
295
296     uint256 myKey = forums[_forum].userKey[msg.sender];
297     uint256 myKarma = forums[_forum].users[myKey].karma;
298     if(myKarma == 0)
299       revert("You need karma to cash out.");
300
301     uint256 price = forums[_forum].cor;
302     if(balances[_forum] < myKarma*price)
303       revert("You can not cash out since forum does not have sufficient
              amount of scones");
304     balances[_forum] -= myKarma*price;
305     balances[msg.sender] += myKarma*price;
306     forums[_forum].users[myKey].karma = 0;
307     emit CashOut(price, myKarma, _forum, msg.sender);
308     return true;
309   }
310
311   event SetKarma(address _forum, address _userAddress, uint256 _karma);
312   event SetCashOutPrice(address _forum, uint256 _cor);
313   event CashOut(uint256 _cor, uint256 _karma, address _from, address
          _to);
314   event AddUserToForum(address _userAddress, string _userName, uint256
          _karma);
315   event CreateForum(bool _success, address _forumAddress, string
          _forumName, uint256 _fid);
316   }
```

Bilaga C: ERC20Interface.sol

```
1  // Abstract contract for the full ERC 20 Token standard
2  // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
3  pragma solidity ^0.5.0;
4
5
6  contract EIP20Interface {
7      /* This is a slight change to the ERC20 base standard.
8      function totalSupply() constant returns (uint256 supply);
9      is replaced with:
10     uint256 public totalSupply;
11     This automatically creates a getter function for the totalSupply.
12     This is moved to the base contract since public getter functions
           are not
13     currently recognised as an implementation of the matching abstract
14     function by the compiler.
15     */
16     /// total amount of tokens
17     uint256 public totalSupply;
18
19     /// @param _owner The address from which the balance will be
           retrieved
20     /// @return The balance
21     function balanceOf(address _owner) public view returns (uint256
           balance);
22
23     /// @notice send `_value` token to `_to` from `msg.sender`
24     /// @param _to The address of the recipient
25     /// @param _value The amount of token to be transferred
26     /// @return Whether the transfer was successful or not
27     function transfer(address _to, uint256 _value) public returns (bool
            success);
28
29     /// @notice send `_value` token to `_to` from `_from` on the
           condition it is approved by `_from`
30     /// @param _from The address of the sender
31     /// @param _to The address of the recipient
32     /// @param _value The amount of token to be transferred
33     /// @return Whether the transfer was successful or not
34     function transferFrom(address _from, address _to, uint256 _value)
           public returns (bool success);
35
36     /// @notice `msg.sender` approves `_spender` to spend `_value`
           tokens
37     /// @param _spender The address of the account able to transfer the
            tokens
38     /// @param _value The amount of tokens to be approved for transfer
39     /// @return Whether the approval was successful or not
40     function approve(address _spender, uint256 _value) public returns (
           bool success);
41
42     /// @param _owner The address of the account owning tokens
43     /// @param _spender The address of the account able to transfer the
            tokens
44     /// @return Amount of remaining tokens allowed to spent
45     function allowance(address _owner, address _spender) public view
           returns (uint256 remaining);
46
47     // solhint-disable-next-line no-simple-event-func-name
```

```
48      event Transfer(address indexed _from, address indexed _to, uint256
            _value);
49      event Approval(address indexed _owner, address indexed _spender,
            uint256 _value);
50  }
```

Bilaga D: Migrations.sol

```
1
2  pragma solidity >=0.4.21 <0.7.0;
3
4  contract Migrations {
5    address public owner;
6    uint public last_completed_migration;
7
8    constructor() public {
9      owner = msg.sender;
10   }
11
12   modifier restricted() {
13     if (msg.sender == owner) _;
14   }
15
16   function setCompleted(uint completed) public restricted {
17     last_completed_migration = completed;
18   }
19 }
```

Bilaga E: 1_initial_migration.js

```
1
2  const Migrations = artifacts.require("Migrations");
3
4  module.exports = function(deployer) {
5    deployer.deploy(Migrations);
6  };
```

Bilaga F: 2_Forum_migration.js

```
1
2
3  const Forum = artifacts.require("Forum");
4
5  module.exports = function(deployer) {
6    deployer.deploy(Forum, 10000, "Scone", 0, "SCN");
7  };
```

Bilaga G: forum.js

```
1
2  const Forum = artifacts.require('Forum');
3
4  contract('Forum', accounts => {
5    let fc = null;
6    before(async () => {
7      fc = await Forum.deployed();
8    });
9
10   it('Should deploy contract', async () => {
11     assert(fc.added != ''); // Contract address is not empty
12   });
13
14   it('Initial balance should be 10000', async () => {
15     let b = null;
16     b = await fc.balanceOf(accounts[0], {from: accounts[2]});
17     assert(b.toNumber() === 10000);
18   });
19
20   it('Name of token should be "Scone"', async () => {
21     let name = null;
22     name = await fc.name();
23     assert(name === "Scone");
24   });
25
26   it('Symbol of token should be "SCN"', async () => {
27     let name = null;
28     name = await fc.symbol();
29     assert(name === "SCN");
30   });
31
32   it('Forum counter should be 0', async () => {
33     let counter = null;
34     counter = await fc.getForumCount();
35     assert(counter.toNumber() === 0);
36   });
37
38   it('User 0 should create forum, F1', async () => {
39     let f = null;
40     f = await fc.createForum("F1");
41     let e = f.receipt.logs[0].event;
42     let success = f.receipt.logs[0].args._success;
43     let self = f.receipt.logs[0].args._forumAddress;
44     let fname = f.receipt.logs[0].args._forumName;
45     let fid = f.receipt.logs[0].args._fid;
46     assert(e === "CreateForum");
47     assert(success === true);
48     assert(self === accounts[0]);
49     assert(fname === "F1");
50     assert(fid.toNumber() === 1);
51   });
52
53   it('User 0 should NOT be able to create a forum again', async () => {
54     let f = null;
55     let msg = '';
56     let numForums = null;
57     try {
58         f = await fc.createForum("DoesNotMatter");
59     }catch(e){
```

```
60        msg = e.message;
61      }
62      numForums = await fc.getForumCount();
63      numForums = numForums.toNumber();
64      assert(numForums === 1)
65      assert(msg.includes("You already own a forum."))
66    });
67
68    it('Forum counter should be 1', async () => {
69      let counter = null;
70      counter = await fc.getForumCount();
71      assert(counter.toNumber() === 1);
72    });
73
74    it('Should return correct forum data via fid', async () => {
75      let f = null;
76      f = await fc.getForumDataByFid(1);
77      let owner = f.forumOwner;
78      let fName = f.forumName;
79      let forumId = f.fid;
80      assert(owner === accounts[0]);
81      assert(fName === "F1")
82      assert(forumId.toNumber() === 1)
83    });
84
85    it('Should return correct forum data via forum address', async () =>
          {
86      let res = null;
87      res = await fc.getForumDataByAddress(accounts[0]);
88      let owner = res.forumOwner;
89      let fName = res.forumName;
90      let forumId = res.fid;
91      assert(owner === accounts[0]);
92      assert(fName === "F1")
93      assert(forumId.toNumber() === 1)
94    });
95
96    it('Only one user should be registered in F1', async () => {
97      let uCount = null;
98      uCount = await fc.getUserCount();
99      assert(uCount.toNumber() === 1);
100   });
101
102   it("Only user 0 should be registered in F1 (using getUserByIndex(
          uint256 x))", async () => {
103     let res = null;
104     res = await fc.getUserByIndex(0); //user index 0
105     let address = res.usrAddress;
106     let name = res.usrName;
107     let karma = res.usrKarma.toNumber();
108     assert(address == accounts[0]);
109     assert(name === "F1");
110     assert(karma === 0);
111   });
112
113   it("Only user 0 should be registered in F1 (using getUserData(address
          user))", async () => {
114     let res = null;
115     res = await fc.getUserByAddress(accounts[0]); //user index 0
116     let address = res.usrAddress;
117     let name = res.usrName;
118     let karma = res.usrKarma.toNumber();
```

```
119       assert(address === accounts[0]);
120       assert(name === "F1");
121       assert(karma === 0);
122     });
123
124     it('Should not be possoble to get a non-existing user (using
          getUserByIndex(uint256 x))', async () => {
125       let res = null;
126       let msg = '';
127       try {
128         res = await fc.getUserByIndex(1);
129       }catch(e){
130         msg = e.message;
131       }
132       assert(msg.includes('This user does not exist.'));
133     });
134
135     it('Should not be possoble to get a non-existing user (using
          getUserByAddress(address a))', async () => {
136       let res = null;
137       let msg = '';
138       try {
139         res = await fc.getUserByAddress(accounts[1]);
140       }catch(e){
141         msg = e.message;
142       }
143       assert(msg.includes('This user does not exist.'));
144     });
145
146
147     it('Using the contract methods, user 1 should not be able to get user
           data without having a forum (using getUserByIndex(uint256 x))',
          async () => {
148       let res = null;
149       let msg = '';
150       try {
151         res = await fc.getUserByIndex(0, {from: accounts[1]});
152       }catch(e){
153         msg = e.message;
154       }
155       assert(msg.includes('You need to create a forum first'));
156     });
157
158     it('Using the contract methods, user 1 should not be able to get user
           data without having a forum  (using getUserByAddress(address a))
          ', async () => {
159       let res = null;
160       let msg = '';
161       try {
162         res = await fc.getUserByAddress(accounts[0], {from: accounts[1]})
              ;
163       }catch(e){
164         msg = e.message;
165       }
166       assert(msg.includes('You need to create a forum first'));
167     });
168
169     it('Should not be possoble to get a non-existing forum', async () =>
          {
170       let res = null;
171       let msg = '';
172       try {
```

```
173       res = await fc.getForumDataByFid(2);
174     }catch(e){
175       msg = e.message;
176     }
177     assert(msg.includes('Forum does not exist.'));
178   });
179
180   it('User 0 should be able to add user 1 to F1', async () => {
181     let res = null;
182     res = await fc.addUserToForum(accounts[1], "1", 10);
183     let args = res.receipt.logs[0].args;
184     let userAddress = args._userAddress;
185     let userName = args._userName;
186     let karma = args._karma.toNumber();
187     assert(userAddress === accounts[1]);
188     assert(userName === "1");
189     assert(karma === 10)
190   });
191
192   it('Should not be possible to add user 1 again', async () => {
193     let res = null;
194     let msg = '';
195     try{
196       res = await fc.addUserToForum(accounts[1], "1", 10);
197     }catch(e){
198       msg = e.message;
199     }
200     assert(msg.includes("User already exists."));
201   });
202
203   it('A user with no forum ownership should not be able to add users',
        async () => {
204     let res = null;
205     let msg = '';
206     try{
207       res = await fc.addUserToForum(accounts[3], "AAA", 10, {from:
            accounts[1]});
208     }catch(e){
209       msg = e.message;
210     }
211     assert(msg.includes("You need to create a forum first"));
212   });
213
214   let karmaArray = [0,10,13,3,43,26,7,12];
215   it('User 0 should add users 2..7 to forum F1', async () => {
216     let res = null;
217     let kChecker = 0;
218     let uNameChecker = 0;
219     let addressChecker = 0;
220     for (let i = 2; i<8; i++){
221       let k = karmaArray[i];
222       res = await fc.addUserToForum(accounts[i], i.toString(), k);
223       let args = res.receipt.logs[0].args;
224       if(args._karma.toNumber() == karmaArray[i])
225         kChecker ++;
226       if(args._userName == i.toString())
227         uNameChecker ++;
228       if(args._userAddress == accounts[i])
229         addressChecker ++;
230     }
231     assert(kChecker === 6);
232     assert(uNameChecker === 6);
```

```
233     assert(addressChecker === 6);
234   });
235
236
237   it('Users 1..7 should be able to fetch its data from forum F1 via
          forum address', async () => {
238     let res = null;
239     let forumAddressChecker = 0;
240     let addressChecker = 0;
241     let uNameChecker = 0;
242     let karmaChecker = 0;
243     for (let i = 1; i<8; i++){
244       res = await fc.getMyInfoByAddress(accounts[0], {from: accounts[i
              ]});
245       if(res.fAddress === accounts[0])
246         forumAddressChecker++;
247       if(accounts[i] === res.myAddress)
248         addressChecker ++;
249       if(res.myUserName === i.toString())
250         uNameChecker ++;
251       if(res.myKarma.toNumber() === karmaArray[i])
252         karmaChecker ++;
253     }
254
255     assert(addressChecker === 7);
256     assert(forumAddressChecker === 7);
257     assert(uNameChecker === 7);
258     assert(karmaChecker === 7);
259   });
260
261   it('Users 1..7 should be able to fetch its data from forum F1 via fid
          ', async () => {
262     let res = null;
263     let fidChecker = 0;
264     let addressChecker = 0;
265     let uNameChecker = 0;
266     let karmaChecker = 0;
267     for (let i = 2; i<8; i++){
268       res = await fc.getMyInfoByFid(1, {from: accounts[i]});
269       if(res.fID.toNumber() === 1)
270         fidChecker ++;
271       if(accounts[i] === res.myAddress)
272         addressChecker ++;
273       if(res.myUserName === i.toString())
274         uNameChecker ++;
275       if(res.myKarma.toNumber() === karmaArray[i])
276         karmaChecker ++;
277     }
278     assert(fidChecker === 6);
279     assert(addressChecker === 6);
280     assert(uNameChecker === 6);
281     assert(karmaChecker === 6);
282   });
283
284   it('User 0 should be able to get user data from ALL its users (via
          user addresses)', async () => {
285     let res = null;
286     let addressChecker = 0;
287     let uNameChecker = 0;
288     let karmaChecker = 0;
289     for (let i = 1; i<8; i++){
290       res = await fc.getUserByAddress(accounts[i]);
```

```
291       if(accounts[i] === res.usrAddress)
292         addressChecker ++;
293       if(res.usrName === i.toString())
294         uNameChecker ++;
295       if(res.usrKarma.toNumber() === karmaArray[i])
296         karmaChecker ++;
297     }
298     assert(addressChecker === 7);
299     assert(uNameChecker === 7);
300     assert(karmaChecker === 7);
301   });
302
303   it('User 0 should be able to get user data from ALL its users (via
          user indexes)', async () => {
304     let res = null;
305     let addressChecker = 0;
306     let uNameChecker = 0;
307     let karmaChecker = 0;
308     for (let i = 1; i<8; i++){
309       res = await fc.getUserByIndex(i);
310       if(accounts[i] === res.usrAddress)
311         addressChecker ++;
312       if(res.usrName === i.toString())
313         uNameChecker ++;
314       if(res.usrKarma.toNumber() === karmaArray[i])
315         karmaChecker ++;
316     }
317     assert(addressChecker === 7);
318     assert(uNameChecker === 7);
319     assert(karmaChecker === 7);
320   });
321
322
323   // tot acc karma F1 = 10+13+3+43+26+7+12 = 114
324   it('User 1 should create forum, F2 ', async () => {
325     let f = null;
326     f = await fc.createForum("F2", {from: accounts[1]});
327     let e = f.receipt.logs[0].event;
328     let success = f.receipt.logs[0].args._success;
329     let self = f.receipt.logs[0].args._forumAddress;
330     let fname = f.receipt.logs[0].args._forumName;
331     let fid = f.receipt.logs[0].args._fid;
332     assert(true)
333     assert(e === "CreateForum");
334     assert(success === true);
335     assert(self === accounts[1]);
336     assert(fname === "F2");
337     assert(fid.toNumber() === 2);
338   });
339
340   it('User 1 should be able to add user 2 to F2', async () => {
341     let res = null;
342     res = await fc.addUserToForum(accounts[2], "2", 13, {from: accounts
          [1]});
343     let args = res.receipt.logs[0].args;
344     let userAddress = args._userAddress;
345     let userName = args._userName;
346     let karma = args._karma.toNumber();
347     assert(userAddress === accounts[2]);
348     assert(karma === 13);
349     assert(userName === "2");
350   });
```

```
351
352   it('Should be possible to add user 8 to F2', async () => {
353     let res = null;
354     res = await fc.addUserToForum(accounts[8], "8", 4, {from: accounts
          [1]});
355     let args = res.receipt.logs[0].args;
356     let userAddress = args._userAddress;
357     let userName = args._userName;
358     let karma = args._karma.toNumber();
359     assert(userAddress === accounts[8]);
360     assert(userName === "8");
361     assert(karma === 4);
362   });
363 // tot karma f2 = 4 + 13 = 17
364   it('User 8 should not exist in F1', async () => {
365     let res = null;
366     let msg = '';
367     try{
368       res = await fc.getUserByAddress(accounts[8]);
369     }catch(e){
370       msg = e.message;
371     }
372     assert(msg.includes("This user does not exist."))
373   });
374
375   it('Acting as user 2, one should get user data from F1 (using
          getMyInfoByFid(uint256 x))', async () => {
376     let res = null;
377     res = await fc.getMyInfoByFid(1, {from: accounts[2]});
378     let fid = res.fID;
379     let myAddress = res.myAddress;
380     let userName = res.myUserName;
381     let karma = res.myKarma;
382     assert(fid.toNumber() === 1);
383     assert(myAddress === accounts[2]);
384     assert(karma.toNumber() === 13);
385     assert(userName === "2");
386
387   });
388
389   it('Acting as user 2, one should get user data from F1 (using
          getMyInfoByAddress(address a))', async () => {
390     let res = null;
391     res = await fc.getMyInfoByAddress(accounts[0], {from: accounts[2]})
          ;
392     let forumAddress = res.fAddress;
393     let myAddress = res.myAddress;
394     let userName = res.myUserName;
395     let karma = res.myKarma;
396     assert(forumAddress === accounts[0] );
397     assert(myAddress === accounts[2]);
398     assert(karma.toNumber() === 13);
399     assert(userName === "2");
400
401   });
402
403   it('Acting as user 8, one should get user data from F2 (using
          getMyInfoByFid(uint256 x))', async () => {
404     let res = null;
405     res = await fc.getMyInfoByFid(2, {from: accounts[8]});
406     let fid = res.fID;
407     let myAddress = res.myAddress;
```

```
408     let userName = res.myUserName;
409     let karma = res.myKarma;
410     assert(fid.toNumber() === 2);
411     assert(myAddress === accounts[8]);
412     assert(karma.toNumber() === 4);
413   });
414
415   it('Acting as user 8, one should get user data from F2 (using
          getMyInfoByAddress(address a))', async () => {
416     let res = null;
417     res = await fc.getMyInfoByAddress(accounts[1], {from: accounts[8]})
            ;
418     let forumAddress = res.fAddress;
419     let myAddress = res.myAddress;
420     let userName = res.myUserName;
421     let karma = res.myKarma;
422     assert(forumAddress === accounts[1] );
423     assert(myAddress === accounts[8]);
424     assert(karma.toNumber() === 4);
425   });
426
427   it('Acting as user 8, one should NOT get user data from F1 (using
          getMyInfoByFid(uint256 x))', async () => {
428     let res = null;
429     let msg = '';
430     try{
431       res = await fc.getMyInfoByFid(0, {from: accounts[8]});
432     }catch(e){
433       msg = e.message;
434     }
435     assert(msg.includes('You are not a member in this forum.'));
436   });
437
438   it('Acting as user 8, one should NOT get user data from F1 (using
          getMyInfoByAddress(address a))', async () => {
439     let res = null;
440     let msg = '';
441     try{
442       res = await fc.getMyInfoByAddress(accounts[0], {from: accounts
              [8]});
443     }catch(e){
444       msg = e.message;
445     }
446     assert(msg.includes('You are not a member in this forum.'));
447   });
448
449   it('Acting as user 8, one should be able to see it is a member in F2'
          , async () => {
450     let res = null;
451     res = await fc.getMemberStatus(2, {from: accounts[8]});
452     assert(res.success === true);
453   });
454
455   it('Acting as user 8, one should be able to see it is NOT a member in
           F1', async () => {
456     let res = null;
457     res = await fc.getMemberStatus(1, {from: accounts[8]});
458     assert(res.success === false);
459   });
460
461   it('Should be able to calculate an accumulated karma sum', async ()
          => {
```

```
462       let res = null;
463       res = await fc.accForumKarma(accounts[0]);
464       assert(res.toNumber() === 114);
465     });
466
467     it('User 0 should not be able to cash out from F3 since F3 does not
              exist', async () => {
468       let res = null;
469       let msg = '';
470       try{
471         res = await fc.cashOut(accounts[2]);
472       }catch(e){
473         msg = e.message;
474       }
475       assert(msg.includes('Forum does not exist'));
476
477     });
478
479     it('User 8 should not be able to cash out from F2 since cash-out
              price is 0', async () => {
480       let res = null;
481       let msg = '';
482       try{
483         res = await fc.cashOut(accounts[1], {from: accounts[8]});
484       }catch(e){
485         msg = e.message;
486       }
487       assert(msg.includes('Cash-out price is 0 scones.'));
488     });
489   //This test
490   // max is 87
491     it("User 0 should not be able to set COR to a value that the user's
              balance can't handle.", async () => {
492       let res = null;
493       let msg = '';
494       try{
495         res = await fc.setCashOutRatio(88);
496       }catch(e){
497         msg = e.message;
498       }
499       assert(msg.includes('Too high. Aquire more scones or try a lower
              COR.'));
500     });
501
502     it("User 0 should be able to set COR to a value that the user's
              balance can handle.", async () => {
503       let fd = null;
504       fd = await fc.getForumDataByFid(1);
505       let fcorBefore = fd.cor.toNumber();
506       await fc.setCashOutRatio(10);
507       fd = await fc.getForumDataByFid(1);
508       let fcorAfter = fd.cor.toNumber();
509       assert(fcorBefore === 0);
510       assert(fcorAfter === 10);
511     });
512
513     it("User 9 should not be able to cash out from F1 since 9 is not a
              member in F1.", async () => {
514       let res = null;
515       let msg = '';
516       try{
517         await fc.cashOut(accounts[0], {from: accounts[9]});
```

```
518       }catch(e){
519         msg = e.message;
520       }
521       assert('You are not a member in this forum.')
522     });
523
524     it("User 0 should not be able to set user 9's karma since 9 is not a
          member in F1.", async () => {
525       let res = null;
526       let msg = '';
527       try{
528         await fc.setKarma(10, accounts[9]);
529       }catch(e){
530         msg = e.message;
531       }
532       assert(msg.includes('This user does not exist in your forum.'))
533     });
534
535     it("User 9 can not set a user's karma since 9 does not own a forum.",
           async () => {
536       let res = null;
537       let msg = '';
538       try{
539         await fc.setKarma(10, accounts[4], {from: accounts[9]});
540       }catch(e){
541         msg = e.message;
542       }
543       assert(msg.includes('You need to create a forum first.'))
544     });
545
546     it("User 0 can not add user 9 to F1 if user 9's karma is set too high
          ", async () => {
547       let res = null;
548       let msg = '';
549       try{
550         // too high if set to 887, (works if 886)
551         res = await fc.addUserToForum(accounts[9], "9", 887)
552       }catch(e){
553         msg = e.message;
554       }
555       assert(msg.includes('User is given too much karma. Your balace
            might not be able to handle it.'))
556     });
557
558     it("User 0 can add user 9 to F1 if user 9's karma is not set too high
          . (It is set to 0).", async () => {
559       let res = null;
560       await fc.addUserToForum(accounts[9], "9", 0);
561       res = await fc.getUserByAddress(accounts[9]);
562       let n = res.usrName;
563       let k = res.usrKarma.toNumber();
564       let a =res.usrAddress;
565       assert(n === "9");
566       assert(a === accounts[9]);
567       assert(k === 0);
568     });
569
570     it("User 0 can not change user 9's karma to a value that is too high"
          , async () => {
571       let res = null;
572       let msg = '';
573       try{
```

```
574       await fc.setKarma(887, accounts[9]);
575     }catch(e){
576       msg = e.message;
577     }
578     assert(msg.includes('User is given too much karma. Your balace
            might not be able to handle it.'))
579   });
580
581   it("User 9 should not be able to cash out from F1 since 9's karma is
          0.", async () => {
582     let res = null;
583     let msg = '';
584     try{
585       res = await fc.cashOut(accounts[0], {from: accounts[9]});
586     }catch(e){
587       msg = e.message;
588     }
589     assert(msg.includes('You need karma to cash out.'));
590   });
591
592 //this would be 886
593   it("User 0 should be able to set 9's karma in F1 to a value that is
          as high as possible.", async () => {
594     let res = null;
595     await fc.setKarma(886, accounts[9]);
596     res = await fc.getUserByIndex(8); // Has index 8. 9 is 8th member
            in F1, and second in F2. Sry for confusion.
597     let k = res.usrKarma.toNumber();
598     let a = res.usrAddress;
599     let n = res.usrName;
600     assert(k === 886);
601     assert(a === accounts[9]);
602     assert(n === "9");
603   });
604
605   it('User 1 should be able to cash out from F1', async () => {
606     let res = null;
607     res = await fc.cashOut(accounts[0], {from: accounts[1]});
608     let cor = res.logs[0].args._cor.toNumber();
609     let karma = res.logs[0].args._karma.toNumber();
610     let forum = res.logs[0].args._from;
611     let to = res.logs[0].args._to;
612     assert(cor === 10);
613     assert(karma === 10);
614     assert(forum === accounts[0]);
615     assert(to === accounts[1]);
616   });
617
618   var bal1 = 0;
619   it('Balance of F2 (or user 1) should be equal to what user 1 got from
          cash out.', async () => {
620     bal1 = await fc.balanceOf(accounts[1]);
621     bal1 = bal1.toNumber();
622     assert(bal1 === 10 * karmaArray[1]);
623
624   });
625
626   it('Balance of F1 (or user 0) should be 10 000 - balanceOf(user 1)',
          async () => {
627     let res = null;
628     res = await fc.balanceOf(accounts[0]);
629     assert(res.toNumber() === 10000-bal1);
```

```
630   });
631
632   it('Acting as user 1, one shold no longer be able to cash out scones
          from F1 (because karma becomes set to 0 when cash out)', async ()
           => {
633     let res = null;
634     let msg = '';
635     try{
636       res = await fc.cashOut(accounts[0], {from: accounts[1]});
637     }catch(e){
638       msg = e.message;
639     }
640     assert(msg.includes('You need karma to cash out.'));
641   });
642
643 //4+13 = 17
644   it('Given that F2 has a set COR, user 8 shold be able to cash out
          from F2', async () => {
645     await fc.setCashOutRatio(1, {from: accounts[1]});
646     let ud = null;
647     let res = null;
648     res = await fc.cashOut(accounts[1], {from: accounts[8]});
649     let cor = res.logs[0].args._cor.toNumber();
650     let karma = res.logs[0].args._karma.toNumber();
651     let forum = res.logs[0].args._from;
652     let to = res.logs[0].args._to;
653     assert(karma === 4);
654     assert(cor === 1)
655     assert(forum === accounts[1]);
656     assert(to === accounts[8]);
657   });
658
659
660   it('Acting as user 8, one shold NOT be able to cash out from F1 since
           user 8 is not a member in F1', async () => {
661     let res = null;
662     let msg = '';
663     try{
664       res = await fc.cashOut(accounts[0], {from: accounts[8]});
665     }catch(e){
666       msg = e.message;
667     }
668     assert(msg.includes('You are not a member in this forum.'));
669   });
670
671   it('The rest of all users in F1 (except 0) should be able to cash out
          ', async () => {
672     let ud = null;
673     let bal = null;
674     let cor = 0;
675     let errors = 0;
676     let check = 0;
677     for (var i = 2; i < 8 ; i++) {
678       try{
679         ud = await fc.getUserByIndex(i);
680         cor = await fc.getCashOutRatio(accounts[0]);
681         res = await fc.cashOut(accounts[0], {from: accounts[i]});
682         bal = await fc.balanceOf(accounts[i]);
683         if(bal.toNumber() === cor.toNumber() * ud.usrKarma.toNumber())
684           check ++;
685       }catch(e){
686         errors ++;
```

```
      }
    }
    // and user 9...
    try{
      ud = await fc.getUserByIndex(8);
      cor = await fc.getCashOutRatio(accounts[0]);
      res = await fc.cashOut(accounts[0], {from: accounts[9]});
      bal = await fc.balanceOf(accounts[9]);
      if(bal.toNumber() === cor.toNumber() * ud.usrKarma.toNumber())
        check ++;
    }catch(e){
      errors ++;
    }
    assert(check === 7)
    assert(errors === 0)
  });

  it('Balance of F1 (or user 0) should be 0', async () => {
    let res = null;
    res = await fc.balanceOf(accounts[0]);
    assert(res.toNumber() === 0)
  });
// tot acc karma F1 = 0+10+13+3+43+26+7+12 = 114
});
```

Bilaga H: Testresultat av forum.js

- Should deploy contract
- Initial balance should be 10000
- Name of token should be ”Scone”
- Symbol of token should be ”SCN”
- Forum counter should be 0
- User 0 should create forum, F1 (98ms)
- User 0 should NOT be able to create a forum again (65ms)
- Forum counter should be 1
- Should return correct forum data via fid
- Should return correct forum data via forum address
- Only one user should be registered in F1
- Only user 0 should be registered in F1 (using getUserByIndex(uint256 x))
- Only user 0 should be registered in F1 (using getUserData(address user))
- Should not be possible to get a non-existing user (using getUserByIndex(uint256 x))
- Should not be possible to get a non-existing user (using getUserByAddress(address a))
- Using the contract methods, user 1 should not be able to get user data without having a forum (using getUserByIndex(uint256 x))
- Using the contract methods, user 1 should not be able to get user data without having a forum (using getUserByAddress(address a))
- Should not be possible to get a non-existing forum
- User 0 should be able to add user 1 to F1 (76ms)
- Should not be possible to add user 1 again (51ms)
- A user with no forum ownership should not be able to add users (44ms)
- User 0 should add users 2..7 to forum F1 (418ms)
- Users 1..7 should be able to fetch its data from forum F1 via forum address (205ms)
- Users 1..7 should be able to fetch its data from forum F1 via fid (129ms)
- User 0 should be able to get user data from ALL its users (via user addresses) (144ms)

- User 0 should be able to get user data from ALL its users (via user indexes) (154ms)
- User 1 should create forum, F2 (65ms)
- User 1 should be able to add user 2 to F2 (62ms)
- Should be possible to add user 8 to F2 (62ms)
- User 8 should not exist in F1
- Acting as user 2, one should get user data from F1 (using getMyInfoByFid(uint256 x))
- Acting as user 2, one should get user data from F1 (using getMyInfoByAddress(address a))
- Acting as user 8, one should get user data from F2 (using getMyInfoByFid(uint256 x))
- Acting as user 8, one should get user data from F2 (using getMyInfoByAddress(address a))
- Acting as user 8, one should NOT get user data from F1 (using getMyInfoByFid(uint256 x))
- Acting as user 8, one should NOT get user data from F1 (using getMyInfoByAddress(address a))
- Acting as user 8, one should be able to see it is a member in F2
- Acting as user 8, one should be able to see it is NOT a member in F1
- Should be able to calculate an accumulated karma sum (83ms)
- User 0 should not be able to cash out from F3 since F3 does not exist (49ms)
- User 8 should not be able to cash out from F2 since cash-out price is 0 (44ms)
- User 0 should not be able to set COR to a value that the user's balance can't handle. (61ms)
- User 0 should be able to set COR to a value that the user's balance can handle. (79ms)
- User 9 should not be able to cash out from F1 since 9 is not a member in F1. (43ms)
- User 0 should not be able to set user 9's karma since 9 is not a member in F1. (42ms)
- User 9 can not set a user's karma since 9 does not own a forum. (45ms)
- User 0 can not add user 9 to F1 if user 9's karma is set too high (70ms)

- User 0 can add user 9 to F1 if user 9's karma is not set too high. (It is set to 0). (84ms)
- User 0 can not change user 9's karma to a value that is too high (64ms)
- User 9 should not be able to cash out from F1 since 9's karma is 0. (47ms)
- User 0 should be able to set 9's karma in F1 to a value that is as high as possible. (72ms)
- User 1 should be able to cash out from F1 (47ms)
- Balance of F2 (or user 1) should be equal to what user 1 got from cash out.
- Balance of F1 (or user 0) should be 10 000 - balanceOf(user 1)
- Acting as user 1, one should no longer be able to cash out scones from F1 (because karma becomes set to 0 when cash out) (47ms)
- Given that F2 has a set COR, user 8 should be able to cash out from F2 (87ms)
- Acting as user 8, one should NOT be able to cash out from F1 since user 8 is not a member in F1 (47ms)
- The rest of all users in F1 (except 0) should be able to cash out (693ms)
- Balance of F1 (or user 0) should be 0

Bilaga I: Applikationens kod

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1"
        >
    <title>Scone</title>
  </head>

  <body>
    <div id="root" class="container-fluid">
    </div>
    <script src="build.js"></script>
  </body>
</html>
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>,
    document.getElementById('root')
);
```

App.js

```
import React, { Component } from 'react';

import {
  Route,
  Link,
  Switch,
  Redirect
} from 'react-router-dom';

import Home from './Home';
import Login from './Login';
import CreateAccount from './CreateAccount';
import MyForum from './MyForum';

class App extends Component {
  render() {
    return (
      <main>
        <Switch>
          <Route exact path="/"  component={Login}/>
          <Route path="/CreateAccount" component={CreateAccount} />
          <Route path="/Home" component={Home} />
          <Route path="/MyForum" component={MyForum} />
        </Switch>
      </main>
```

```
    );
  }
}

export default App;
```

Login.js

```
import React, {Component} from 'react';
import {Link} from "react-router-dom";
import {useHistory} from "react-router-dom";

class Login extends Component {
    constructor(props) {
    super(props);
    this.state = {
      username: '',
      password: '',
      msg: '',
      data: ''
    }

    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleSubmit(event) {
    event.preventDefault();
    if(this.state.username && this.state.password) {
        var data = {
            username: this.state.username,
            password: this.state.password
        }

        let currentComponent = this;

        fetch("http://localhost:3000/users/auth", {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify(data)
        }).then(response => {
            if (response.status >= 400) {
              throw new Error("Bad response from server");
            }
            return response.json();
        }).then(obj => {
            if(obj.error) {
                currentComponent.setState({ msg: obj.error });
                console.log(obj.error);
            } else {
                currentComponent.setState({ data: obj});
            }
        }).catch(function(error) {
            console.log(error);
        });
    } else {
        this.setState({msg: "Please enter username and password."});
    }
  }

  logChange(e, key) {
    this.setState({[key]: e.target.value});
  }

```

```
  render() {
    return (
        <div className="container">
            <h1>Login</h1>
          <div className="container register-form">
            <form onSubmit={this.handleSubmit} method="POST">
                <label>Username</label>
                <input type="text" className="form-control" onChange={e
                     => this.logChange(e, "username")}/>
                <label>Password</label>
                <input type="password" className="form-control"
                    onChange={e => this.logChange(e, "password")}/>
                <div className="submit-section">
                    <button className="btn btn-uth-submit">Submit</
                        button>
                </div>
            </form>
          </div>

          <p>{this.state.msg}</p>

          <Link to="CreateAccount">Create Account</Link>

          <HistoryHook data={this.state.data} />

        </div>
    );
  }
}

/*
Allows us to use hooks on data from class.
Will generate a warning which can be ignored.
*/
const HistoryHook = (props) => {
    if(props.data) {
        const history = useHistory();
        history.push({
            pathname: '/Home',
            state: {data: props.data}
        });
    }

    return(
    <div>
    </div>
    );
};

export default Login;
```

Home.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Web3 from 'web3';
import TruffleContract from 'truffle-contract';
import Forum from '../../build/contracts/Forum.json';
import 'bootstrap/dist/css/bootstrap.css';
import equal from 'fast-deep-equal';
import {Link} from "react-router-dom";

class Home extends React.Component {
```

```
125   constructor(props) {
126     super(props);
127     this.state = {
128       forums: [],
129       balance: 0,
130       data: '',
131       msg: "You are currently not a member of any forum."
132     }
133
134   this.setup = this.setup.bind(this);
135   this.cashOut = this.cashOut.bind(this);
136
137   this.Web3 = require('web3');
138   this.web3 = new Web3('http://localhost:7545');
139
140   this.contractJson = require('../../build/contracts/Forum.json');
141   this.contractAddress = this.contractJson.networks[5777].address;
142   this.contract = new this.web3.eth.Contract(this.contractJson.abi,
          this.contractAddress);
143   }
144
145   componentDidMount() {
146     this.setup();
147   }
148
149   setup() {
150     this.contract.methods.getForumCount().call({gas: 6721975}, (err, id
          ) => {
151       var i;
152       for(i = 1; i <= id; i++) {
153         this.contract.methods.getMemberStatus(i).call({from: this.props
              .location.state.data[0].address}, (err, obj) => {
154           if(obj.success) {
155             this.setState({msg: ''});
156             this.contract.methods.getMyInfoByFid(obj._fID).call({from:
                  this.props.location.state.data[0].address}, (error,
                  result)=> {
157               this.setState(prevState => ({
158                 forums: [...prevState.forums, result]
159               }))
160             })
161           }
162         })
163       }
164     });
165
166     this.contract.methods.balanceOf(this.props.location.state.data[0].
          address).call((err, result) => {
167       this.setState({ balance: result })
168     })
169   }
170
171   cashOut(forumAddress) {
172     this.contract.methods.cashOut(forumAddress)
173       .send({from: this.props.location.state.data[0].address, gas:
              6721975}, (error, result) => {
174         if (!error) {
175           console.log('success');
176         } else {
177           console.log('fail');
178         }
179       })
```

```
180   }
181
182   logChange(e, key) {
183     this.setState({[key]: e.target.value});
184   }
185
186   render() {
187     return(
188       <div className="container-fluid">
189         <div className="row">
190
191           <div className="col-4">
192             <div>
193               <p>Username: {this.props.location.state.data[0].username
                      }</p>
194               <p>Address: {this.props.location.state.data[0].address}</
                      p>
195               <p>Balance: {this.state.balance}</p>
196             </div>
197           </div>
198
199           <div className="col-4">
200             <p>{this.state.msg}</p>
201             <ul className="list-group">
202               {this.state.forums.map(forum => {
203                 return <li className="list-group-item" key={forum._fID
                        }>
204                   <h1>{forum._forumName}</h1> <br></br>
205                   <p>Karma: {forum._karma}</p> <br></br>
206                   <p>Cash Out Price: {forum._cop}</p><br></br>
207                   <button onClick={() => this.cashOut(forum.
                          _forumAddress)}>Cashout</button>
208                 </li>
209               })}
210             </ul>
211           </div>
212
213           <div className="col-4">
214             <Link to={{
215               pathname:'MyForum',
216               data: {
217                 user: this.props.location.state.data[0],
218                 methods: this.contract.methods
219               }
220             }}> MyForum </Link>
221           </div>
222
223         </div>
224       </div>
225     )
226   }
227 }
228
229 export default Home;
```

MyForum.js

```
230 import React from 'react';
231 import ReactDOM from 'react-dom';
232 import 'bootstrap/dist/css/bootstrap.css';
233 import equal from 'fast-deep-equal';
234
235 class MyForum extends React.Component {
```

```
236   constructor(props) {
237     super(props)
238     this.state = {
239       forumName: '',
240       address: '',
241       username: '',
242       balance: '',
243       ownsForum: false,
244       addUserMsg: '',
245       cashOutPrice: '',
246       setCashOutMsg: '',
247       createForumMsg: ''
248     }
249
250     this.setup = this.setup.bind(this);
251     this.setCashOutPrice = this.setCashOutPrice.bind(this);
252     this.addUser = this.addUser.bind(this);
253     this.createForum = this.createForum.bind(this);
254   }
255
256   componentDidMount() {
257     this.setup();
258   }
259
260   setup() {
261     this.props.location.data.methods.getForumData(this.props.location.
          data.user.address).call((error, result) => {
262       if(error || !result.forumName) {
263         this.setState({ownsForum: false});
264       } else {
265         this.setState({
266           ownsForum: true,
267           forumName: result.forumName,
268           cashOutPrice: result.cop
269         });
270       }
271     })
272
273     this.props.location.data.methods.balanceOf(this.props.location.data
          .user.address).call((err, result) => {
274       this.setState({ balance: result });
275     })
276   }
277
278   setCashOutPrice(event) {
279     event.preventDefault();
280     if (this.state.price) {
281       this.props.location.data.methods.setCashOutPrice(this.state.price
            )
282         .send({from: this.props.location.data.user.address, gas:
              6721975}, (error, result) => {
283         if (error) {
284           this.setState({setCashOutMsg: "To high. Aquire more scones or
                try a lower COP."});
285         } else {
286           this.setState({
287             cashOutPrice: this.state.price,
288             setCashOutMsg: ""
289           });
290         }
291       })
292     } else {
```

```
293       this.setState({setCashOutMsg: "You must enter a price"});
294     }
295   }
296
297   addUser(event) {
298     event.preventDefault();
299     if (this.state.address && this.state.username) {
300       try{
301         this.props.location.data.methods.addUserToForum(this.state.
              address, this.state.username, 100)
302           .send({from: this.props.location.data.user.address, gas:
                6721975}, (error, result) => {
303           if(error) {
304             this.setState({ addUserMsg: "User could not be added" });
305           } else {
306             this.setState({
307               addUserMsg: "User was added",
308               address: '',
309               username: ''
310             });
311           }
312         })
313       } catch (e) {
314         console.log(e);
315         this.setState({ addUserMsg: "The user does not exist" });
316       }
317     } else {
318       this.setState({ addUserMsg: "You must enter an address and a
              username" });
319     }
320   }
321
322   createForum(event) {
323     event.preventDefault();
324     if (this.state.forumName) {
325       this.props.location.data.methods.createForum(this.state.forumName
              )
326         .send({from: this.props.location.data.user.address, gas:
                6721975}, (error, result) => {
327         if (error) {
328           console.log('fail');
329         } else {
330           console.log('success');
331           this.setState({ownsForum: true});
332         }
333       })
334     } else {
335       this.setState({createForumMsg: "You must enter a Forum name"});
336     }
337   }
338
339   logChange(e, key) {
340     this.setState({[key]: e.target.value});
341   }
342
343   render() {
344     if(this.state.ownsForum) {
345       return(
346         <div className="container">
347           <h1>{this.state.forumName}</h1>
348           <p>Balance: {this.state.balance}</p>
349           <p>Cash Out Price: {this.state.cashOutPrice}</p>
```

```
350
351
352             <h3>Set Cash Out Price</h3>
353             <div className="container register-form">
354               <form onSubmit={this.setCashOutPrice}>
355                   <label>Price in scones/karma</label>
356                   <input type="text" className="form-control" value={this
                          .state.price} onChange={e => this.logChange(e, "
                          price")}/>
357                   <p>{this.state.setCashOutMsg}</p>
358                   <div className="submit-section">
359                       <button className="btn btn-uth-submit">Set</button>
360                   </div>
361               </form>
362             </div>
363
364             <h3>Add user</h3>
365             <div className="container register-form">
366               <form onSubmit={this.addUser}>
367                   <label>Username</label>
368                   <input type="text" className="form-control" value={this
                          .state.username} onChange={e => this.logChange(e, "
                          username")}/>
369                   <label>User Address</label>
370                   <input type="text" className="form-control" value={this
                          .state.address} onChange={e => this.logChange(e, "
                          address")}/>
371                   <p>{this.state.addUserMsg}</p>
372                   <div className="submit-section">
373                       <button className="btn btn-uth-submit">Add user</
                              button>
374                   </div>
375               </form>
376             </div>
377           </div>
378         );
379       } else {
380       return(
381           <div className="container">
382             <h1>Create Forum</h1>
383             <div className="container register-form">
384               <form onSubmit={this.createForum}>
385                   <label>Forum name:</label>
386                   <input type="text" className="form-control" onChange={e
                          => this.logChange(e, "forumName")}/>
387                   <p>{this.state.createForumMsg}</p>
388                   <div className="submit-section">
389                       <button className="btn btn-uth-submit">Create</
                              button>
390                   </div>
391               </form>
392             </div>
393           </div>
394         );
395       }
396     }
397 }
398
399 export default MyForum;
```

Bilaga J: Databas-serverns kod

server.js

```
const express = require("express");
const bodyParser = require("body-parser");
const mysqlConnection = require("./connection");
const UsersRoutes = require("./routes/users");
const cors = require('cors');
const session = require('express-session');

var app = express();

app.use(cors());

app.use(session({
  secret: 'secret',
  resave: true,
  saveUninitialized: true
}));

app.use(bodyParser.json());

app.use("/users", UsersRoutes);

app.listen(3000);
```

connection.js

```
const mysql = require("mysql");
var mysqlConnection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'ForumDB',
  multipleStatements: true
});

mysqlConnection.connect((err)=>{
  if(!err) {
    console.log("Connected to ForumDB");
  } else {
    console.log("Connection to ForumDB Failed");
  }
});

module.exports = mysqlConnection;
```

users.js

```
const express = require("express");
const Router = express.Router();
const mysqlConnection = require("../connection");

Router.post('/new', function(request, response, next) {
  if (request.body.username && request.body.address && request.body.
      password) {
    mysqlConnection.query('INSERT INTO users (username, address,
        password) VALUES (?, ?, MD5(?))',
      [request.body.username, request.body.address, request.body.
          password], (error, results, fields) => {
      if(!error) {
        response.send(results);
```

```
      } else {
        response.send(error);
        console.log(error);
      }
    });
  } else {
    response.send(JSON.parse('{"error":"You have not entered a username and a password."}'));
  }
});

Router.post('/auth', function(request, response) {
  if (request.body.username && request.body.password) {
    mysqlConnection.query('SELECT * FROM users WHERE username = ? AND password = MD5(?);',
      [request.body.username, request.body.password], function(error, results, fields) {
      if(!error) {
        response.send(results);
      } else {
        console.log(error);
      }
    });
  } else {
    response.send(JSON.parse('{"error":"You have not entered username, address and password."}'));
  }
});

module.exports = Router;
```