Blockchain technológiák és alkalmazások

Házi feladat dokumentáció - L(a)unch codes with Solidity

Fekete Sámuel (GJ8J3A) - Száraz Dániel (GT5X34)

Tervezési döntések

A feladat megoldásában az őrváltás megvalósítása volt a legnehezebb, ebben is az, hogy biztosítsuk a megfelelő sorrendet. Sokat könnyített a helyzeten, hogy rájöttünk, ha nem engedünk be egyszerre háromnál több embert, akkor az az őrváltásnál fontos kritériumok egy részét is lefedi, hiszen az őrváltásnál mindenképpen hároman tartózkodnak az épületben, így természetesen más nem tud közben kilépéni. Ez alól kivétel az az időpont, amikor az első őr már lecserélődött és el is hagyta az épületet, de a második még nem lépett be. Ilyenkor bárki kérelmezhet belépést, de mindig a jelenlegi őrök kezében van, hogy kit engednek be. Amíg a teljes őrváltás végbe nem megy, nem engedhetnek be mást, csak a leendő őrt. Ezt viszont a szerződésben nem tudjuk biztosítani, hiszen a hívó címéről nem tudjuk eldönteni, hogy ő-e a leendő őr, azt csak a jelenlegi őrök tudják.

Adatmodell

A feladathoz két saját típust is létrehoztunk, ezek a LogEntry és a GuardExchangeState. A LogEntry egy olyan struct, melyet a logolásnál használunk, az éppen naplót író személy address-e kerül bele és az, hogy ki- vagy bement. A GuardExchangeState pedig az őrváltás lebonyolításáoz létrehozott enum, amely az őrváltás állapotait tartalmazza. Ez lehet NONE, amikor nincs őrváltás, G1_REQUESTED, amikor a 3. ember az őrrel váltani szeretne, G1_APPROVED ha ezt az első őr el is fogadta. G1_EXITED amikor az első őr akit leváltottak elhagyta az épületet, majd G2_REQUESTED és G2_APPROVED a második őrrel is az előzőhöz hasonlóan. Ha végbement a csere, akkor újra NONE állapotba kerülünk.
A kontrakton belül 4 címet tárolunk változókban: a 2 őrt, a 3. embert és aki bármilyen kérést folytat, amit az őröknek jóvá kell hagyni. Ez lehet kinti és benti ember is. Emellett van 4 állapotjelző változónk is: guard1Approve, guard2Approve: az adott őr elfogadta-e a kérést, ha van, és entryStarted, exitStarted, azaz éppen elkezdődött-e a belépése vagy a kimenési folyamat. Végezetül a saját típusainkat is használjuk: a log-ot egy LogEntry tömbbe írjuk és az őrváltás állapotát egy GuardExchangeState típusú state változóban tároljuk.

A napló

A naplózáshoz a korábban említett LogEntry-t használjuk. Naplózásra akkor kerül sor, ha valaki elhagyta, vagy belépett az épületbe. Az adott ember először engedélyt kér az őröktől a ki- vagy bemenetelre, majd ha erre engedélyt kapott, akkor ennek a 3. személynek kell meghívnia az entry() vagy pedig az exit() függvényt, amely egyéb funkciók mellett a logba írást is végzi.

Az őrváltás státusza

Az őrváltáashoz a korábban említett GuardExchangeState típust használtuk. Az őrváltáshoz 3-an kell bent lenni az épületben, a 3. személy ott tudja meghívni a requestGuardChange() fv-t, mely egyaránt működik első és második őr váltásának kérésére is. NONE esetén G1_REQUESTED, G1_EXITED esetén pedig G2_REQUESTED-be vált. Ez után hívhatja meg az őrök egyike a approveGuardChange() fv-t, mely szintén működik mindkét őr váltásának elfogadására, tehát átkerülhetünk G1_APPROVED és G2_APPROVED állapotokba.
Az őrváltás alatt kétszer kerül sor kimenésre, és egyszer bejövésre. A kimenést az exit() fv biztosítja és állapottól függően átkerülhetünk G1_EXITED-be, vagy ha a második ex-őr is elhagyta az épületet, akkor véget ért az őrváltás, így NONE-ba kerülünk. Fontos még megemlíteni, hogy az entry() fv abban az esetben, ha G1_EXITED státuszban vagyunk (és ha meg lehet hívni az azt jelenti, hogy az őrök elfogadták), akkor meghívjuk a requestGuardChange() fv-t, mivel az őrcserének egy folyamatnak kell lenni, és nem szabad egy őr váltása utan másoknak ki-be mászkálni, aki az első őr váltása után bemegy, annak kell lennie a másik őrnek, de a beengedést az őrök tudják intézni.

Az okos szerződés API-ja

requestEntry()

Ezzel a függvénnyel lehet belépést kérni az őröktől. Amennyiben a küldő valamelyik őr, van az épületben a két őrön kívül még valaki, vagy valaki már kért belépést, de az még nem lett elfogadva vagy elutasítva, a tranzakció revert-elve lesz.

approve()

Az őrök ezzel a függvénnyel fogadhatják el az éppen aktív belépési vagy kilépési kérelmet. Ha nem egy őr hívja, vagy nincsen aktív kérelem, a tranzakció revertelve lesz.

deny()

Az őrök ezzel a függvénnyel utasíthatják el az éppen aktív belépési vagy kilépési kérelmet. Mivel mindkét őrnek jóvá kell hagynia egy belépést, ezen függvény sikeres lefutása után a kérelem automatikusan visszavonódik, függetlenül attól, hogy a másik őr reagált-e rá. Ha nem egy őr hívja, vagy nincsen aktív kérelem, a tranzakció revertelve lesz.

entry()

Ezzel a függvénnyel lehet belépni az épületbe, de csak akkor, ha a küldő küldött korábban belépési kérelmet, és azt mindkét őr elfogadta. A belépéssel együtt a naplóba egy új bejegyzés kerül, a küldő címével. Ha őrváltás van folyamatban és az első őr már lecserélődött, akkor ez a függvény az új belépő nevében automatikusan kérelmezi, hogy ő lecserélje a második őrt, hiszen mást ilyenkor nem szabad beengedni az épületbe.

exit()

Ezzel a függvénnyel lehet kilépni az épületből, de csak akkor, ha a küldő küldött korábban kilépési kérelmet, és azt mindkét őr elfogadta. A kilépéssel együtt a naplóba és új bejegyzés kerül, a küldő címével. Ha őrváltás van folyamatban és az első őr már lecserélődött, akkor ez a függvény az új belépő nevében automatikusan kérelmezi, hogy ő lecserélje a második őrt, hiszen mást ilyenkor nem szabad beengedni az épületbe.

requestGuardChange()

Ezzel a függvénnyel kérelmezhet őrváltást az az épületben tartózkodó ember, aki jelenleg nem őr. Az, hogy melyik őrt váltja le, attól függ, hogy mi az őrváltás állapota. Ha eddig még nem kezdődött el, akkor az első őrt fogja, ha az első őr váltása megtörtént, akkor a másodikat. Ha a küldő nem a harmadik ember az épületben, van aktív kilépési kérelme vagy korábbi őrváltási kérelme, akkor revert-elni fog a tranzakció.

approveGuardChange()

Ezzel a függvénnyel fogadhatja el az őrváltást a leváltandó őr. Az őrváltás állapotától függ, hogy melyik őr hívhatja, mindig az, akit éppen le szeretnének váltani. A függvény meghívásával a váltás megtörténik, az extra ember lesz az őr, a korábbi őr pedig az extra ember.

denyGuardChange()

Ha az őrváltás során már kérelmezték az első őr leváltását, de azt a leváltandó őr nem akarja elfogadni, ezzel a függvénnyel utasíthatja el. Ha más hívja vagy más állapotban, akkor revert-el a tranzakció. A második őr leváltását azért nem lehet elutasítani, mert őrváltás közben más nem léphet be az épületbe. Ha az őrváltás megkezdődött, akkor az első őr távozása után a jelenlegi őröknek csak azt az embert szabad beengedniük az épületbe, aki a második őrt fogja leváltani.

getGuard1()

Visszaadja az egyes számú őr címét.

getGuard2()

Visszaadja az kettes számú őr címét.

getExtraPerson()

Visszaadja az extra ember címét, aki a két őrön kívül az épületben tartózkodik. Amennyiben nincs más bent, a nullás címet adja vissza.

getRequestor()

Visszaadja az éppen aktuális (belépést vagy kilépést) kérelmező címét. Őrváltást csak az extra ember kérelmezhet, ezért arra nincs külön függvény.

getState()

Visszaadja az őrváltás aktuális státuszát.

getLog()

Visszaadja a napló jelenlegi állapotát.

Tesztesetek

Takarító belép az épületbe, majd elhagyja azt

A tesztben egy külső személy kér belépési engedélyt, megkapja és bemegy. Ellenőrizzük a logot és a változókat, hogy megtörtént-e a belépés. Majd ez után engedélyt kér a kimenésre, megkapja és kimegy. Ez után ismét ellenőrizzük a logot és a változókat.

Negyedik ember belépési próbálkozása

Az előzőhöz hasonlóan bejön egy 3. ember és ellenőrizzük. Ezután egy 4. kint lévő ember is megpróbál belépési engedélyt kérni az őröktől, de ez hibát dob, mivel már 3-an bent vannak. Ellenőrizzük, hogy van-e hiba, majd a 3. ember is elhagyja az épületet és ezt is ellenőrizzük.

Műszakban lévő őr megpróbálja elhagyni az épületet

Egy őr megpróbál kilépési engedélyt kérni, de mivel őr, ezért nem mehet ki, így ez hibát dob és ezt ellenőrizzük is.

Támadó megpróbál belépni az épületbe a szükséges engedélyek nélkül

A támadó megpróbál először kérelem nélkül bemenni, ez természetesen hibát dob és ellenőrizzük. Ez után kér belépési engedélyt, de mielőtt kapna bármi választ megpróbál bemenni, ez se sikerül, hibát dob és ellenőrizzük. Ez után az egyik őr elfogadja, majd a másik őr elfogadása előtt ismét megpróbál bemenni, ismét sikertelenül. Hibát dob és ellenőrizzük. Ez után a 2. őr elutasítja a kérést, miután újból megpróbál bejönni, de ez is hibára fut, amit ellenőrzünk és mivel elutasítás volt, így a kérelem is törlődik, ezt is ellenőrizzük.

Őrváltás

Minden akció után ellenőrizzük a state-et, a 2 őrt és a 3. embert is (ahol aminek értelme van). Ezekre a továbbiakban csak ellenőrzés-sel fogok utalni. Kezdetben van még egy state ellenőrzés, ami NONE. Kezdetben őr1 és őr2 vannak, őr3 és őr4 vált.
Őr3 engedélyt kér, bejön, ellenőrzés. Őrváltást kér, ellenőrzés. Őr1 elfogadja, megtörténik a váltás, ellenőrzés. Most már őr3 az első őr, és őr1 lett az extra ember. Őr1 kimenési engedélyt kér, megkapja, kimegy, ellenőrzés. Ez után őr4 jön, belépési engedélyt kér, megkapja, bemegy, ellenőrzés. A váltási igényét már a belépéssel jelezte. Őr2 elfogadja, ellenőrzés. Most már az őrök leváltódtak, őr3 és őr4 van szolgálatban. Végezetül ő2 is kér kilépési engedélyt, megkapja, kimegy, ellenőrzés.

A projekt futtatása, tesztelése

A projektet a tárgy Teams csoportjában megosztott dokumentum alapján, a Truffle MetaCoin kiinduló projektből hoztuk létre, így a futtatásához az ott felsorolt szoftverek szükségesek (node 16, Truffle extension for VS Code, ganache-cli, truffle cli).
A projekt fordítani a következő paranccsal lehet: truffle compile
A teszteket pedig ezzel a paranccsal lehet futtatni: truffle test
A tesztek egymás után futnak, így az állapotok végigmennek az összes teszten.