МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
МЭДЭЭЛЛИЙН ТЕХНОЛОГИ, ЭЛЕКТРОНИКИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРЫН УХААНЫ ТЭНХИМ

Энхбаярын Жавхлан

Блокчэйн суурьт лиценз баталгаажуулалт

(Licence validation with blockchain)

Программ хангамж
Бакалаврын судалгааны ажил

Улаанбаатар хот

2024 он

МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
МЭДЭЭЛЛИЙН ТЕХНОЛОГИ, ЭЛЕКТРОНИКИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРЫН УХААНЫ ТЭНХИМ

Блокчэйн суурьт лиценз баталгаажуулалт
(Licence validation with blockchain)

Программ хангамж
Бакалаврын судалгааны ажил

Удирдагч: _____ Дэд профессор Ч.Алтангэрэл

Гүйцэтгэсэн: _____ Э.Жавхлан (20B1NUM0649)

Улаанбаатар хот

2024 он

Зохиогчийн баталгаа

Миний бие Энхбаярын Жавхлан ”Блокчэйн суурьт лиценз баталгаажуулалт” сэдэвтэй судалгааны ажлыг гүйцэтгэсэн болохыг зарлаж дараах зүйлсийг баталж байна:

- Ажил нь бүхэлдээ эсвэл ихэнхдээ Монгол Улсын Их Сургуулийн зэрэг горилохоор дэвшүүлсэн болно.
- Бусдын хийсэн ажлаас хуулбарлаагүй, ашигласан бол ишлэл, зүүлт хийсэн.
- Ажлыг би өөрөө (хамтарч) хийсэн ба миний хийсэн ажил, үзүүлсэн дэмжлэгийг тайлангийн ажилд тодорхой тусгасан.
- Ажилд тусалсан бүх эх сурвалжид талархаж байна.

Гарын үсэг: _____

Огноо: _____

ГАРЧИГ

ЗУРГИЙН ЖАГСААЛТ

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

УДИРТГАЛ

Өнөөгийн цахим орчинд зонхилон тохиолдож буй оюуны өмч болон цахим бүтээгдэхүүний хулгай, өмчлөх эрхийн ил тод байдал, зөвшөөрөлгүй түгээлт зэрэг сорилтуудтай тулгарч байна. Блокчэйн технологийн төвлөрсөн бус, ил тод, хувиршгүй шинж чанарыг ашигласнаар цахим бүтээгдэхүүн эзэмших, лиценз олгоход итгэлцэл, ил тод байдлыг бий болгож, улмаар оюуны өмчийн зөвшөөрөлгүй хулгайн гэмт хэргийг бууруулах зорилготой уг сэдвийг сонгосон. Энэхүү судалгааны ажлаар хэрэглэгчид блокчэйн технологиор дамжуулан цахим бүтээлийг хамгаалах, хуваалцах, түүнд хандах зөвшөөрөл олгох цахим бүтээлийн лицензийн төвлөрсөн бус систем хөгжүүлэх зорилготой билээ. Уг зорилгын хүрээнд дараах зорилтуудыг тавьсан болно.

1. Блокчэйн технологи болон дижитал эрхийн менежментийн талаар судлах
2. Системийг хэрэгжүүлэх технологийн талаар судлах
3. Системийн зохиомж, архитектурыг боловсруулах
4. Блокчэйн дээр суурилсан цахим бүтээлийн лицензийн систем хөгжүүлэх

1. ОНОЛЫН СУДАЛГАА

Онолын судалгаа бүлгийн хүрээнд блокчэйн технологи болон дижитал эрхийн менежментийн талаар судлах мөн цахим бүтээлийн лицензийн төвлөрсөн бус системтэй ижил төстэй системүүд болон, энэхүү сэдвээр судалгааны ажил хийгдсэн эсэхийг судалж хооронд нь харьцуулалт хийх бөгөөд үүний үр дүнд системийн хэрэгцээ шаардлага, давуу тал, өөр ямар нэмэлт боломжууд байж болох талаар мэдэх юм. Мөн системийг хөгжүүлэхэд шаардагдах технологи болон фреймворкуудын талаарх судалгаа хийлээ.

1.1 Блокчэйн технологи

Хамгийн анх блокчэйн технологийн талаар 2008 онд Сатоши Накамото гэдэг этгээдийн нийтэлсэн “Биткойн: Peer-to-Peer Электрон Мөнгөний Тогтолцоо” судалгааны ажлын нийтлэлд дурдагдсан байдаг.

Блокчэйн гэдэг нь өгөгдөл буюу дата мэдээллүүдийг хадгалдаг нэгэн төрлийн мэдээллийн бааз гэж хэлж болно. Бааз доторх дата мэдээллийг Блок гэж нэрлэгдэх хэсгүүдэд багцлан хадгалж уг сүлжээнд холбогдсон бүх компьютерт ижил хуулбар болгон тархмал хэлбэрээр хадгална. Тэдгээр блокуудыг өөр хоорондоо гинжин хэлхээ буюу математик тооцоолол, цахим нууцлалын аргаар хэлхэн холбосноор бидний ярьж буй Блокчэйн үүсэх юм.

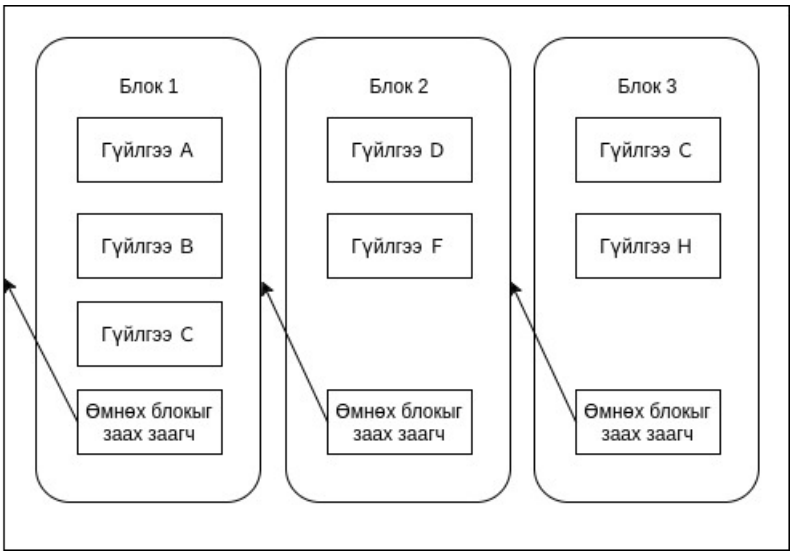
1.1.1 Блокчэйний онцлог

Тархсан Peer-to-Peer (P2P) сүлжээ гэдэг нь сүлжээнд оролцогч буюу зангилаанууд нь газарзүйн хувьд тархсан байдаг ба аль ч хоёр зангилаа хоорондоо байршлаас үл хамааран ямар нэг серверээр дамжилгүйгээр өөр хоорондоо шууд холбогддог сүлжээ юм.

Тархсан бүртгэлийн дэвтэр буюу Distributed ledger technology (DLT) технологи нь мэдээллийг тархсан P2P сүлжээний оролцогч нар дээр хадгалдаг технологи бөгөөд уламжлалт өгөгдлийн сангийн системээс ялгарах гол ялгаа нь төвлөрсөн өгөгдлийн сан болон төвлөрсөн удирдлагын функц байхгүйд оршино.

Блокчэйн нь DLT технологийн гол төлөөлөгч бөгөөд мэдээллийн гинжин хэлхээ юм. Блокчэйнд тогтсон хэмжээтэй блок үүсгэж, үүн дотроо мэдээллийг хадгалах ба эхний блок дүүрэхэд дараагийн шинэ блок үүсгэдэг. Эдгээр блок нь хэш функцээр кодлогдсон байх ба блокийг цаг хугацааны дагуу жагсааж, блок тус бүр яг өөрийн өмнөх блокийн мэдээллийг өөр дотроо хадгалах байдлаар гинжин бүтцийг үүсгэнэ.

Блокчэйн технологийн хамгийн чухал, онцлох давуу тал нь төвлөрсөн бус тархсан бүтэцтэй бөгөөд сүлжээнд байгаа бүх компьютер блокчэйний халдашгүй чанарыг үргэлж баталгаажуулж байдаг ба хэн нэгэн, эсвэл аль нэг компани үүн доторх өгөгдөл түүний бүрэн бүтэн байдлыг удирдах боломжгүй байдагт байгаа юм. Блокчэйний бүх зангилаа ижил мэдээллийг агуулж байдаг болохоор “A” зангилаан дахь өгөгдөл эвдэрч гэмтвэл блокчэйний хэсэг болж чадахгүй, учир нь өгөгдөл нь бусад “B” болон “C” зангилааны өгөгдөлтэй ижил байж чадахгүй болно.



Зураг 1.1: Блокчэйний өгөгдлийн бүтэц

1.1.2 Блокчэйний нууцлалын технологи

Криптограф хэш

Криптограф хэш функц нь оруулсан өгөгдлийн уртаас үл хамааран тогтсон урттай хэш утгуудыг буцаадаг. Оролтын зөвхөн нэг тэмдэгт өөрчлөгдөхөд гаралтын хэш утгууд нь эрс ялгаатай

байна. Энэ шинж чанарыг ашиглан, гүйлгээний өгөгдөл болон бусад бүх өгөгдлийн хувьд засвар ороогүй болохыг баталгаажуулах боломжийг олгодог. Жишээ нь, та Mac-ын командлайнаар дараах командыг оруулбал, SHA-256 hash функцийн утгыг хялбархан олох болно.

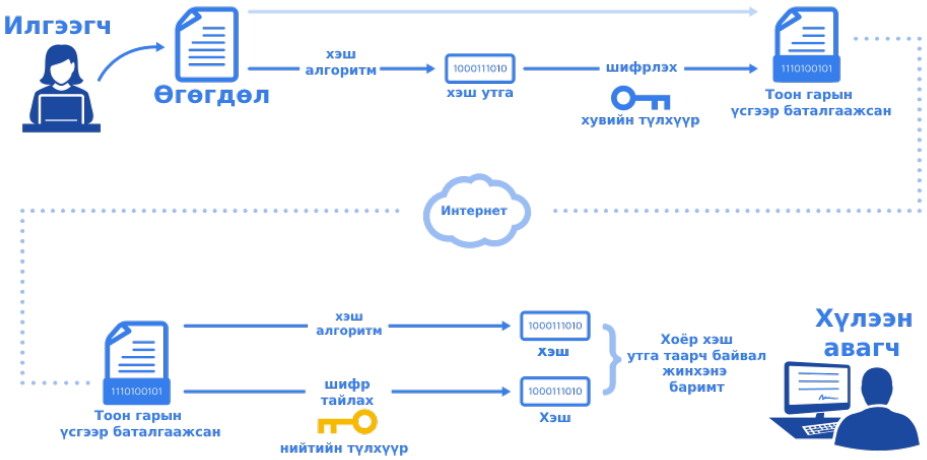
```
❯ echo "Hello World" | openssl sha256
SHA2-256(stdin)= d2a84f4b8b650937ec8f73cd8be2c74add5a911ba64df27458ed8229da804a26
❯ echo "Hallo World" | openssl sha256
SHA2-256(stdin)= e6c1396639c0b79bebc94e4448cfe2700b871d45d0d38d98df6ee9da3f09d35c
```

Зураг 1.2: ”Hello World”, ”Hallo World” гэсэн үгнүүдийн хэшийг бодсон байдал

”Hello World”, ”Hallo World” гэсэн үгнүүдийн sha256 хэшийг бодсон байдал жишээн дээр, ”Hello World” гэсэн үгний SHA-256 hash утга болон, ”e”-г ”a”-гээр сольсон үгийн SHA-256 hash утгыг харуулсан байна. Зөвхөн нэг үсгээр ялгаатай боловч, тэдгээрийн hash утгууд нь эрс ялгаатай байна. Энэ мэтчилэн hash функц нь оролт нь 1 байтаар л ялгаатай байхад, эрс өөр үр дүн гаргадаг шинж чанарыг агуулж байдаг. Энэ шинж чанарыг ашиглан, гүйлгээний өгөгдөл болон тэдгээрийг хадгалах блокийн бүх өгөгдлийн хувьд гарах hash утгыг тухайн өгөгдлийн бүтцэд оруулснаар, засвар ороогүй болохыг баталгаажуулах боломжтой болно.

Тоон гарын үсэг

Тоон гарын үсэг нь дижитал мессеж эсвэл баримт бичгийн жинхэнэ эсэхийг шалгах математик аргачлал юм. Энэ нь хос түлхүүр үүсгэх замаар ажилладаг: өргөн тархсан нийтийн түлхүүр, нууцлагдсан хувийн түлхүүр. Гарын үсэг зурахдаа баримт бичгийн өвөрмөц хэшийг үүсгэж, хувийн түлхүүрээр шифрлэж, тоон гарын үсгийг бүрдүүлдэг. Хүлээн авсны дараа хэшийг илгээгчийн нийтийн түлхүүрээр тайлж, хүлээн авсан баримтаас шинэ хэш үүсгэнэ. Хэрэв хоёулаа таарч байвал энэ нь тухайн баримт бичиг нь жинхэнэ бөгөөд ямар нэгэн өөрчлөлт ороогүй гэсэн үг юм.



Зураг 1.3: Цахим гарын үсгийн ажиллах зарчим

Тоон гарын үсэгт хаш функц болон хос түлхүүрийг нийлүүлж ашигласнаар, өгөгдөл илгээгчийг болон агуулгын засагдаагүй гэдгийг баталгаажуулах ажлыг зэрэг гүйцэтгэдэг юм. Блокчэйнд өмнөх хэсгийн хаш функц болон дээр өгүүлсэн цахим гарын үсгийг аль алийг нь ашигладаг бөгөөд гүйлгээ тус бүрийн үнэн зөв байдал, нийцтэй байдлын талаарх мэдээллийн илгээгч, агуулгын бүрэн бүтэн(засагдаагүй) байдлын баталгаа зэрэг төрөл бүрийн зорилгоор ашигладаг.

1.1.3 Ухаалаг гэрээ

Ухаалаг гэрээ гэдэг нь дундын зуучлагч буюу хуульч, нотариатгүйгээр хоёр этгээд гэрээ байгуулсныг баталсан компьютерын код бөгөөд тухайн гэрээний нөхцөл, үүрэг, хариуцлагыг багтаасан байна. Анх этереум нь ухаалаг гэрээг оруулсан блокчэйн гэдгийг гаргаж, түүний дараагаар олон тооны блокчэйнд ухаалаг гэрээг оруулж ирсэн. Ухаалаг гэрээ нь зөвхөн нөхцөл, үүргийг заахаас гадна автоматаар биелэх боломжтой байдаг.

Анх 1996 онд Nick Szabo ухаалаг гэрээний санааг нь гаргаж ирсэн. Гол санаа нь хүнээс хамааралгүйгээр урьдчилан тодорхойлсон ямар нэг нөхцөлийн биелэх үед автоматаар үйлдэл хийгдэнэ.

1. Хийгдсэн үйлдэл/гүйлгээ нь олон нийтэд ил байх ч, хэн хийсэн бэ гэдэг нь нууц байж болдог.

2. Блокчэйн сүлжээний бүх зангилаанууд ухаалаг гэрээг ажиллуулдаг.

3. Цаг хугацаа хэмнэхээс гадна гарч болох олон асуудлыг шийдэх боломжтой. (3-дагч этгээдийг оролцоо хэрэггүй)

1.1.4 Блокчэйн зарим хэрэглээ

Олон улсын хэмжээнд стартап компаниуд блокчэйн технологийг ашигласан шинэ системийг эрүүл мэнд, даатгал, татвар зэрэг олон салбарт санал болгож байна.

Жишээлбэл, эрүүл мэндийн салбарт блокчэйн рүү иргэний эрүүл мэндийн болон эмчилгээний түүхийг оруулдаг болгох систем юм. Энэ тохиолдолд эмчлэгч эмч тухайн иргэний мэдээллийг харах судалгаа, шинжилгээний зорилгоор авч ашиглахаар бол системд хүсэлт гаргахад зөвхөн тухайн иргэний зөвшөөрлөөр системээс мэдээлэл нь харагдана. Хүний эрүүл мэндийн мэдээлэл блокчэйнд хадгалагдсанаар тухайн хүн дэлхийн аль ч улс оронд эмчилгээнд хамрагдахад асуудалгүй болж байгаа юм. Мөн блокчэйнд хүн өөрийн итгэмжлэгдсэн төлөөллийг нэмж өгөх боломжтой бөгөөд тухайн хүн өөрөө блокчэйнээс мэдээллээ гаргаж өгөх боломжгүй нөхцөлд ашиглагдах юм. Хэрэв блокчэйн ашиглагдаж эхэлбэл зайнаас эмчлэх, эмчилгээний зөвлөгөө өгөх зэрэг шинэ төрлийн үйлчилгээнүүд хүчээ авах юм.

Нэгдсэн Үндэстний Байгууллага 2017 онд блокчэйн технологи ашигласан олон төрлийн санал, санаачилагыг хэрэгжүүлснээс үүний нэг болох тусламж түгээлтийн бүртгэлийн систем амжилттай хэрэгжсэн байна. НҮБ-аас гаргасан судалгаагаар, нийт тусламжийн 30 орчим хувь нь очих ёстой хүлээн авагчдаа хүрдэггүй гэж гарсан байна. 2017 оны тавдугаар сараас НҮБ-ын Дэлхийн хүнсний хөтөлбөрт хэрэгжсэн хүрээнд Сирийн дүрвэгчдэд үзүүлж байгаа тусламжийг этереум блокчэйн ашиглаж түгээжээ. Тодруулбал, Иордан улсын дүрвэгчдийн хуаранд байрлаж байгаа Сири улсын 10500 дүрвэгчид хүнсний бүтээгдэхүүн (1.4 сая ам.доллар) түгээхэд криптовалютад суурилсан ваучер тарааж, уг ваучераа ашиглан хуаранд байрлах дэлгүүрээс хүнсний бүтээгдэхүүн авах боломжийг хангажээ. НҮБ-аас уг төслийг өмнөх тусламжтай харьцуулахад маш амжилттай хэрэгжсэн гэж үзэж байгаа бөгөөд 2018 оны хоёрдугаар улиралд

тусламжинд хамрагдах хүний тоог 500,000-д хүргэхээр төлөвлөж байна гэж мэдээлж байна.

НҮБ-аас хамгийн сүүлд эхлүүлсэн нэг ажил нь хүүхдийг блокчэйнд бүртгэлжүүлэх систем юм. Хуурамч бичиг баримт үйлдэн хүүхэд хил дамнуулахыг зогсооход хамгийн ээдрээтэй зүйл нь жинхэнэ юм шиг бүрдүүлсэн хуурамч бичиг баримтыг таних ажил байдаг. Хүний наймаа ихээр явагддаг бүс нутагт хүүхдүүдийг шат дараатайгаар албан ёсны бүртгэлтэй болгож, түүнийг нь НҮБ-ын блокчэйн системд хадгална. Энэ төрлийн гэмт хэрэг хамгийн их явагддаг Молдав улсад хэрэгжүүлж эхэлсэн ажээ. НҮБ-ийн судалгаагаар 5-аас доош насны хүүхэд бүртгэлжээгүй байх тохиолдол зарим бүс нутагт их байдаг байна.

Швейцарийн Зуг (ZUG) хот нь крипто хот болохоор ажиллаж байгаа бөгөөд ийм уриа гаргасан бусад хот болох Сан-Франциско, Лондон, Токио, Сингапур, Нью-Иорк, Амстердамаас ялгагдах зүйл нь санхүү болон технологийн гарааны бизнесээ эхэлж буй компаниудад хууль эрх зүйн орчин нь маш тааламжтай юм. Зуг хотын удирдлага крипто хөндий байгуулж, иргэдээ блокчэйнд бүртгэж эхэлсэн ба 2017 оны арваннэгдүгээр сараас иргэддээ зориулж цахим ID авах вебийн үйлчилгээг нээсэн нь этереум блокчэйнд суурилсан ба хэрэглэгч хаанаас ч өөрийн мэдээллийг оруулан цахим ID-гаа авах боломжтой бөгөөд хотын зүгээс уг мэдээллийг зөвхөн шалгаж баталгаажуулах эрхтэй. Энэхүү цахим ID-гаа ашиглаад иргэд зөвхөн хотын үйлчилгээг (хэрэглээний төлбөр, түрээсийн төлбөр) авахаар хязгаарлагдахгүй ба 2018 оны хавар сонгуулийн санал өгөхөд (e-vote) ашиглахаар бэлдэж байна.

1.2 Дижитал эрхийн менежмент (DRM)

Digital Rights Management (DRM) системийг дижитал контентыг хэрэглэх, тараах, хуваалцахад хяналт тавих зорилготой технологиуд гэж ойлгож болно. Энэ нь ном, хөгжим, кино, программ хангамж зэрэг дижитал контентыг хамгаалах, агуулгын эзэмшигч, хэвлэн нийтлэгчдийн оюуны өмчийн эрхийг хамгаалахад чиглэдэг.

1.2.1 DRM-ийн үндсэн ойлголт ба бүрэлдэхүүн хэсгүүд:

- **Шифрлэлт:** Шифрлэлт нь криптограф алгоритмыг ашиглан цахим контентыг унших боломжгүй формат руу хөрвүүлэх явдал юм. Шифрлэгдсэн контентод зөвхөн шаардлагатай код тайлах түлхүүрийг эзэмшсэн эрх бүхий хэрэглэгчид хандах буюу тайлж болно.
- **Хандалтын хяналт:** DRM систем нь дижитал контент руу хэн хандах, үзэх, өөрчлөх, түгээх боломжтойг зохицуулах хандалтын хяналтын механизмыг хэрэгжүүлдэг. Хандалтын эрхийг ихэвчлэн хэрэглэгчийн үүрэг, лиценз эсвэл контент эзэмшигчээс олгосон зөвшөөрөл дээр үндэслэн тодорхойлдог.
- **Лицензийн менежмент:** DRM шийдлүүд нь хэрэглэгчдэд дижитал контент руу нэвтрэх, ашиглах зөвшөөрөл олгохын тулд лицензэд суурилсан загваруудыг ашигладаг. Лицензүүд нь ашиглалтын хугацаа, зөвшөөрөгдсөн төхөөрөмж, нэгэн зэрэг хэрэглэгчдийн тоо зэрэг ашиглалтын нөхцөл, нөхцөлийг тодорхойлдог.
- **Тоон усан тэмдэг:** Тоон усан тэмдэг нь үл үзэгдэх танигч эсвэл гарын үсгийг цахим контентод оруулахад ашигладаг техник юм. Усан тэмдэглэгээг контентын зөвшөөрөлгүй хуулбарыг эх сурвалж руу нь буцаах эсвэл контентын жинхэнэ эсэхийг шалгахад ашиглаж болно.
- **Хуулбарлах хамгаалалт:** DRM системүүд нь цахим контентыг зөвшөөрөлгүй хуулбарлах, хуулбарлахаас сэргийлэхийн тулд хуулбарлах хамгаалалтын механизмыг хэрэгжүүлдэг. Хулгайлах, зөвшөөрөлгүй түгээхээс урьдчилан сэргийлэхийн тулд хуулбарлахаас урьдчилан сэргийлэх, хуулбарлах хяналт, хуулбар илрүүлэх зэрэг арга техникийг ашигладаг.

1.3 DRM систем дэх блокчэйн технологийн боломжууд

Блокчейн технологи нь төвлөрсөн бус, ил тод, өөрчлөх боломжгүй шинж чанартай бөгөөд уламжлалт дижитал эрхийн менежментийн (DRM) системийн тулгардаг олон асуудлыг шийдвэрлэх боломжтой. Үүнд:

1. **Төвлөрсөн бус өмчлөлийн болон гарал үүслийн бүртгэл:** Блокчейн технологи нь дижитал хөрөнгийн өмчлөл болон гарал үүслийг өөрчлөх боломжгүй, төвлөрсөн бус бүртгэлийг хангаж чадна. Ингэснээр зохиогч болон эзэмшигч нар зохих хүлээн зөвшөөрөл, нөхөн төлбөрийг авах боломжтой бөгөөд хөрөнгийн өмчлөл болон шилжүүлгийн түүхийг ил тод байдлаар хянах боломжтой болно. Энэ нь уламжлалт DRM системд тулгардаг өмчлөлийн эргэлзээ болон ил тод бус байдлын асуудлыг шийдвэрлэнэ.

2. **Ухаалаг гэрээгээр дамжуулан эрхийн автоматжуулсан менежмент:** Блокчейн дээр суурилсан ухаалаг гэрээнүүд нь дижитал хөрөнгийн хэрэглээ, роялти төлбөр тооцоо болон бусад холбогдох нөхцөлүүдийг автоматаар хэрэгжүүлэх боломжтой. Эдгээр нь өөрөө гүйцэтгэгддэг гэрээнүүд нь зөвхөн ил тод, үнэн зөв байдлыг хангаад зогсохгүй тохиролцсон нөхцөлүүдийг автоматаар хэрэгжүүлж, оролцогч талуудын итгэлцлийг нэмэгдүүлнэ.

3. **Хэрэглэгчийн хяналт ба эрх мэдэл:** Блокчейн технологид суурилсан дижитал хөрөнгийн менежментийн систем нь хэрэглэгчдэд хөрөнгийн хандах эрхийн зөвшөөрөл, өмчлөх эрхийг шилжүүлэх, дахин зарах зэрэг хяналтыг олгож, шинэ бизнес загваруудыг хэрэгжүүлэх боломжийг олгоно. Энэ нь хэрэглэгчийн эрх мэдэл, хяналтын дутагдлыг шийдвэрлэж, уламжлалт DRM системд тулгардаг асуудлыг арилгана.

4. **Төлбөр тооцоо ба мөнгө олох шинэ загварууд:** Блокчейн нь төлбөр тооцоог хөнгөвчлөх чадвар нь ашиглалтын төлбөр эсвэл захиалгад суурилсан хандалт зэрэг дижитал контентоор мөнгө олох шинэ загваруудыг бий болгож чадна. Энэ нь уламжлалт DRM загвартай харьцуулахад илүү уян хатан, хэрэглэгчдэд ээлтэй сонголтуудыг өгөх боломжтой.

1.4 Технологийн судалгаа

1.4.1 React & Next.js

Declarative

React нь хэрэглэгчийн интерактив интерфейс бүтээхийг хялбарчилдаг. Аппликейшны state бүрд зориулсан энгийн бүтэц зохион байгуулахаас гадна, React нь өгөгдөл өөрчлөгдөхөд яг зөв компонентоо өөрчлөн рендер хийдэг. Declarative бүтэц нь кодыг тань debug хийхэд хялбар болгохоос гадна, ажиллагаа нь илүү тодорхой болдог.

Next.js

Netflix, TikTok, Hulu, Twitch, Nike гэсэн орчин үеийн аваргууд ашигладаг энэхүү орчин үеийн фрэймворк нь React технологи дээр үндэслэгдсэн бөгөөд Frontend, Backend хоёр талд хоёуланд нь ажилладаг веб аппуудыг хийх чадвартайгаараа бусдаасаа давуу юм. Next.js-ийн үндсэн дизайн нь клиент болон сервер талын аль алиных давуу талыг ашиглаж чаддаг, ямар нэг дутагдалгүй веб сайтыг яаж хамгийн хурдан хялбар бүтээх вэ гэдгийг бодож тусгасан байдаг. Next.js нь сервер талд react компонентуудыг рендерлэн энгийн html, css, json файл болгон хувиргах замаар ажилладаг бөгөөд 2020 оноос олон нийтэд танигдсан JAMStack технологи болон статик сайт, автоматаар статик хуудас үүсгэх, CDN deployment, сервергүй функц, тэг тохиргоо, файлын системийн рүүтинг (PHP-ээс санаа авсан), SWR (stale while revalidate), сервер талд рендерлэх зэрэг асар олон орчин үеийн шинэхэн технологиудыг бүгдийг хийж чаддаг анхны бүрэн веб фрэймворк гэж хэлж болно.

1.4.2 Ethereum блокчэйн

Төвлөрсөн бус, блокчэйн дээр суурилсан программуудыг хангамжийн платформ анх Ethereum-ийг 2013 онд программист Vitalik Buterin бичсэн бөгөөд 2015 онд олон нийтэд анх танилцуулагдсан юм. Ethereum нь бусад койныг бодвол зөвхөн арилжааны бус тус платформыг ашиглан smart

contract буюу ухаалаг гэрээ үүсгэх боломжтой. Энэ нь энгийнээр хөгжүүлэгчдэд төвлөрсөн бус хэрэглээний программуудыг бүтээх, ажиллуулах боломжийг олгодог.

1.4.3 Hardhat

Hardhat нь ухаалаг гэрээг хөгжүүлэх орчин юм. Энэ нь Ethereum ухаалаг гэрээг бичих, туршихаас эхлээд байршуулах, дибаг хийх хүртэлх бүх амьдралын мөчлөгийг хөнгөвчлөх зорилготой юм. Hardhat нь Ethereum Virtual Machine (EVM) дээр бүтээгдсэн бөгөөд Ethereum, Polygon, Avalanche болон бусад EVM-тэй нийцтэй блокчэйнүүдийг дэмждэг.

1.4.4 Wagmi

Wagmi нь блокчэйнтэй ажиллахад шаардлагатай бүх зүйлийг агуулсан React Hook-ийн цуглуулга юм. Wagmi нь крифто түрийвч холбох, мэдээллийг авах, ухаалаг гэрээтэй харилцах гэх мэт үйлдлүүдийг хөнгөвчлөх боломжийг олгодог.

1.4.5 IPFS & Pinata

IPFS буюу Interplanetary File System нь peer-to-peer сүлжээн дэх файлуудыг хадгалах, хуваалцахад зориулагдсан төвлөрсөн бус протокол юм. Үндсэндээ IPFS нь файлуудыг жижиг хэсгүүдэд хувааж, сүлжээний олон зангилаанд хадгалдаг. Энэ нь файлуудыг нэг байршилд хадгалдаггүй, харин сүлжээгээр тарааж байршуулдаг. Pinata нь төвлөрсөн бус бичиг баримт хадгалалтын сүлжээ болох Interplanetary File System (IPFS) дээр бүтээгдсэн үйлчилгээ юм. Pinata нь хөгжүүлэгчид болон хэрэглэгчдэд IPFS сүлжээнд өгөгдөл хадгалах, уншихад хялбар болгодог. Энэ нь IPFS дээр хадгалагдсан файлуудыг байршуулах, удирдах, хандахад зориулсан API болон бусад хэрэгслээр хангаснаар IPFS-тэй харилцах үйл явцыг хялбаршуулдаг.

1.4.6 Lit Protocol

Lit Protocol нь хэрэглэгчдэд өгөгдөл шифрлэх, түүнд хандах хандалтыг хянах боломжийг олгодог төвлөрсөн бус түлхүүр удирдлагын систем юм. Lit Protocol-ийн хандалтын хяналтын систем нь хэрэглэгчид өөрсдийн өгөгдөлд хэн хандаж, ашиглах боломжтойг нарийн хянах боломжийг олгодог. Lit Protocol-г дараах зорилгоор ашиглаж болно.

- Lit сүлжээнд өгөгдлийг шифрлэх түлхүүр үүсгэх, удирдах
- Lit сүлжээнээс өгөгдлийн шифрлэлтийг тайлах түлхүүрийг авах эрхийг нарийвчилсан нөхцөлтэйгөөр тодорхойлох
- Lit сүлжээнээс түлхүүрийг авч өгөгдлийн шифрлэлтийг тайлах

Lit Protocol бол төвлөрсөн бус сүлжээ бөгөөд үйл ажиллагааны хувьд програмчлагдах түлхүүр хэлбэрээр ажилладаг. ”threshold криптографи” гэж нэрлэгддэг процессоор Lit нь хувийн түлхүүрийг төвлөрсөн бус болгож, уг түлхүүр нь Lit зангилаанууд дээр тархсан байдлаар хадгалдаг. Lit сүлжээ нь өгөгдөл шифрлэхэд BLS тоон гарын үсэг ашиглан тодорхой нөхцөл хангасан хүмүүст л шифрлэлтийг тайлахыг зөвшөөрдөг таних тэмдэгт (ID) суурилсан шифрлэлтийн схемийг ашигладаг. BLS (Boneh-Lynn-Shacham) тоон гарын үсэг нь нийтийн түлхүүрт суурилсан криптографийн нэг төрөл бөгөөд өгөгдлийг нууцлах, баталгаажуулахад өргөн ашиглагддаг. Энэ нь тодорхой нөхцөл хангасан хүмүүст л нууцлалыг тайлах эрх олгодог тул зөвхөн зөвшөөрөгдсөн хүмүүс л өгөгдлийг үзэх боломжтой болдог.

Өгөгдөл шифрлэхийн тулд дараах алхмуудыг хийх хэрэгтэй.

1. Крипто хэтэвчний гарын үсгийг авах, өөрөөр хэлбэл хэтэвчийг эзэмшдэг эсэхийг нотлох
2. Хэн таны өгөгдлийн шифрлэлтийг тайлах түлхүүрийг авч чадах талаар хандалтын хяналтын нөхцөлийг тодорхойлох
3. Lit сүлжээнд холбогдож, өгөгдлийг шифрлүүлэх түлхүүр үүсгэн өгөгдлийг шифрлэх

1.5 Ижил төстэй системийн судалгаа

2. СИСТЕМИЙН ШИНЖИЛГЭЭ, ЗОХИОМЖ

Системийн шинжилгээ, зохиомж бүлгийн хүрээнд өмнөх бүлгээс олж авсан мэдээллийн дагуу системийн функциональ ба функциональ бус шаардлагыг боловсруулж түүнээс системийн статик болон динамик загварыг боловсруулан системийн ерөнхий архитектурыг гаргасан болно.

2.1 Системийн шаардлага

2.1.1 Функциональ шаардлагуудыг дараах хүснэгтэд тодорхойлов

ФШ 10	Систем нь цахим бүтээл болон лицензийн талаарх мэдээллийн найдвартай байдлыг хадгалахын тулд блокчэйнтэй харилцах ёстой.
ФШ 20	Системд хэрэглэгчид крипто хэтэвчээ ашиглан өөрийгөө баталгаажуулах боломжтой байх ёстой (жишээ нь, MetaMask).
ФШ 30	Системд зөвхөн холбогдсон түрийвчтэй, баталгаажуулсан хэрэглэгчид системийн функцэд хандах эрхтэй байх ёстой.
ФШ 40	Систем нь хэрэглэгчдэд янз бүрийн төрлийн цахим бүтээлийг (жишээ нь, видео, аудио, PDF, зураг) аюулгүйгээр байршуулах боломжтой байх ёстой.
ФШ 50	Систем нь цахим бүтээлийг байршуулахдаа бүтээлийн мэдээллийг бүртгэх мөн үнийг тогтоох боломжтой байх ёстой.
ФШ 60	Систем нь өгөгдлийн нууцлал, аюулгүй байдлыг хангах үүднээс байршуулахаас өмнө файлуудыг шифрлэн IPFS (InterPlanetary File System) дээр найдвартай хадгалах ёстой.
ФШ 70	Систем нь шифрлэгдсэн файлуудыг зохих зөвшөөрөлтэй эрх бүхий хэрэглэгчдэд зориулан тайлах ёстой.

ФШ 80	Системд хэрэглэгчдэд цахим бүтээлийн үнэд тохирсон шаардлагатай төлбөрийг төлж эзэмшигчдээс тухайн цахим бүтээлд хандах лиценз буюу хандах зөвшөөрөл авах хүсэлт илгээх боломжтой байх ёстой.
ФШ 90	Системд цахим бүтээл эзэмшигчид лицензийн хүсэлтийг зөвшөөрөх эсвэл татгалзах боломжтой байх ёстой. Зөвшөөрөгдсөн үед худалдан авагчид бүтээлийн лицензийг өгөх мөн зохих төлбөрийг эзэмшигчид шилжүүлэх ёстой.
ФШ 100	Системд цахим бүтээл эзэмшигчид мөн хэрэглэгчдийн лицензийн хүсэлтээс татгалзах сонголттой байх ёстой. Иймтохиолдолд төлбөрийг хэрэглэгчдэд буцааж өгөх ёстой.
ФШ 110	Систем нь цахим бүтээл эзэмших, түүнд лиценз олгох асуудлыг зохицуулахын тулд ухаалаг гэрээг блокчэйн дээр байршуулж, удирдах ёстой.
ФШ 120	Системд хэрэглэгчид хүссэн үедээ системээс цахим бүтээлийн орлогоо өөрийн крипто хэтэвч рүү татах боломжтой байх ёстой.
ФШ 130	Систем нь бүтээл байршуулах, удирдах, өмчлөлийг баталгаажуулахын тулд ухаалаг гэрээтэй харилцах ёстой.
ФШ 140	Хэрэглэгчид систем дээр байршуулсан цахим бүтээлийн дэлгэрэнгүй мэдээллийг үзэх боломжтой байх ёстой.
ФШ 150	Цахим бүтээлийн лиценз авахад лицензэд өвөрмөц дугаар олгож, блокчэйн дээр хадгалах ёстой.
ФШ 160	Систем нь хэрэглэгчид лиценз авсны дараа лицензийн дугаар, файлын мэдээлэл зэрэг лицензийнхээ дэлгэрэнгүй мэдээллийг агуулсан цахим гэрчилгээ олгох ёстой.

Table 2.1: Функциональ шаардлагууд

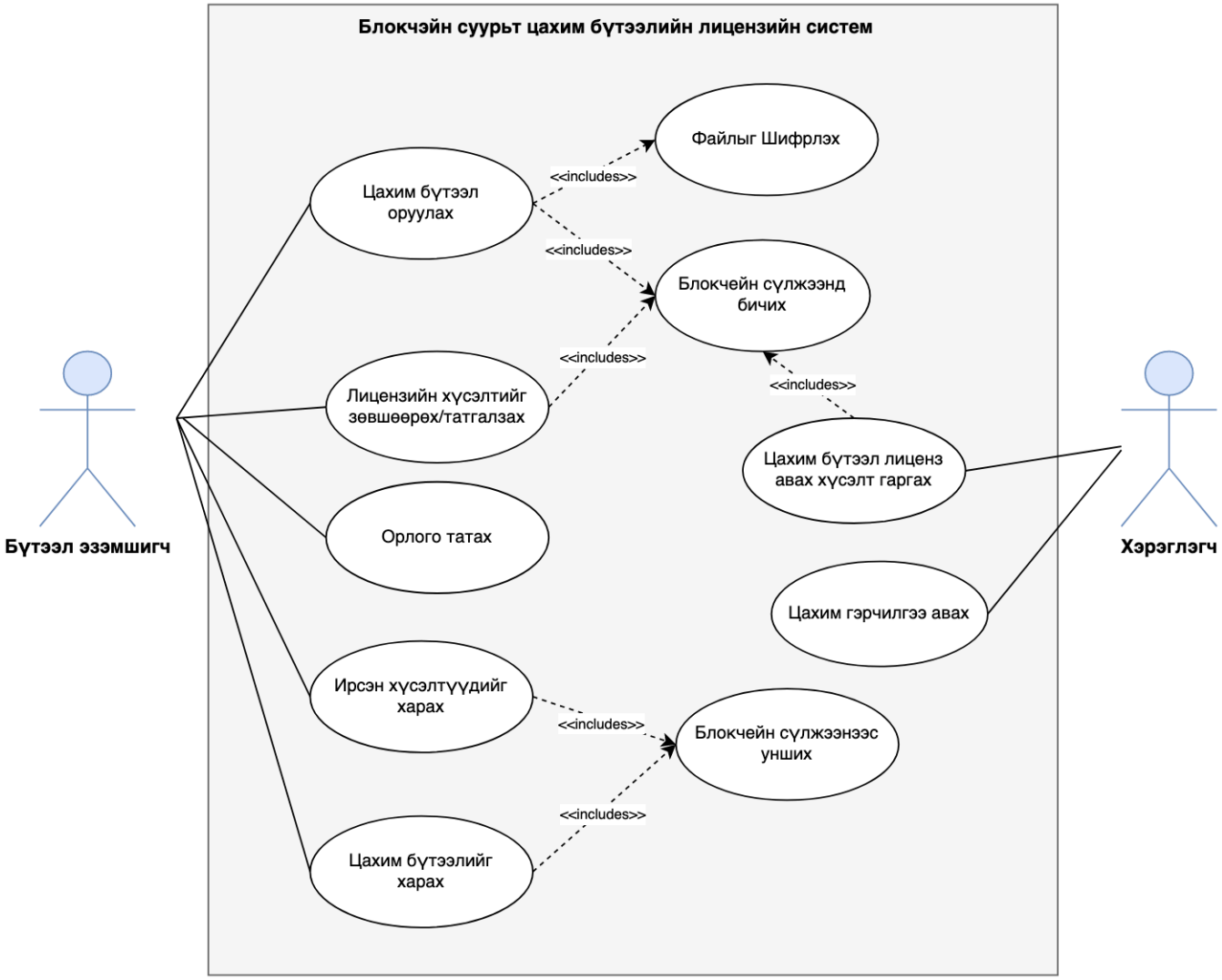
2.1.2 Функциональ бус шаардлагуудыг дараах хүснэгтэд тодорхойлов

ФБШ 10	Блокчэйн технологи нь өгөгдлийн бүрэн бүтэн байдлыг хангаж, лицензийн мэдээллийг зөвшөөрөлгүй өөрчлөхөөс сэргийлнэ.
ФБШ 20	Систем нь гүйцэтгэлийн бууралтгүйгээр олон тооны хэрэглэгчид болон лицензүүдийг зохицуулах чадвартай байх ёстой.
ФБШ 30	Ухаалаг гэрээ нь модульчлагдсан байх ёстой бөгөөд шинэчлэгдэхэд хялбар байх ёстой.
ФБШ 40	Систем нь хүлээн зөвшөөрөгдсөн тодорхой хугацааны дотор баталгаажуулах хүсэлтийг хурдан боловсруулах чадвартай байх ёстой.
ФБШ 50	Систем нь янз бүрийн техникийн чадвартай хэрэглэгчдэд үүнийг үр дүнтэй ашиглах боломжийг олгодог хэрэглэгчдэд ээлтэй интерфейстэй байх ёстой.
ФБШ 50	Систем нь янз бүрийн үйлдлийн систем, хөтөч, төхөөрөмжтэй нийцтэй байх ёстой.
ФБШ 60	Систем нь лиценз олгох, дижитал гүйлгээ, блокчэйн технологитой холбоотой аливаа зохицуулалтын шаардлагад нийцэж байх ёстой.
ФБШ 70	Энэ систем нь гамшгийн үед өгөгдөл алдагдахгүй байхын тулд найдвартай нөөцлөх, сэргээх механизмтай байх ёстой.

Table 2.2: Функциональ бус шаардлагууд

2.2 Системийн ажлын явцын диаграмм

Цахим бүтээлийн ажлын явцыг дараах байдлаар тодорхойлов. Системийн оролцогч талуудыг цахим бүтээлийн лиценз буюу хандах зөвшөөрөл авах гэж буй хэрэглэгч. Нөгөө талаас цахим бүтээл оруулах, түүнд лиценз буюу хандах зөвшөөрөл олгож буй бүтээл эзэмшигч гэж тодорхойлсон.

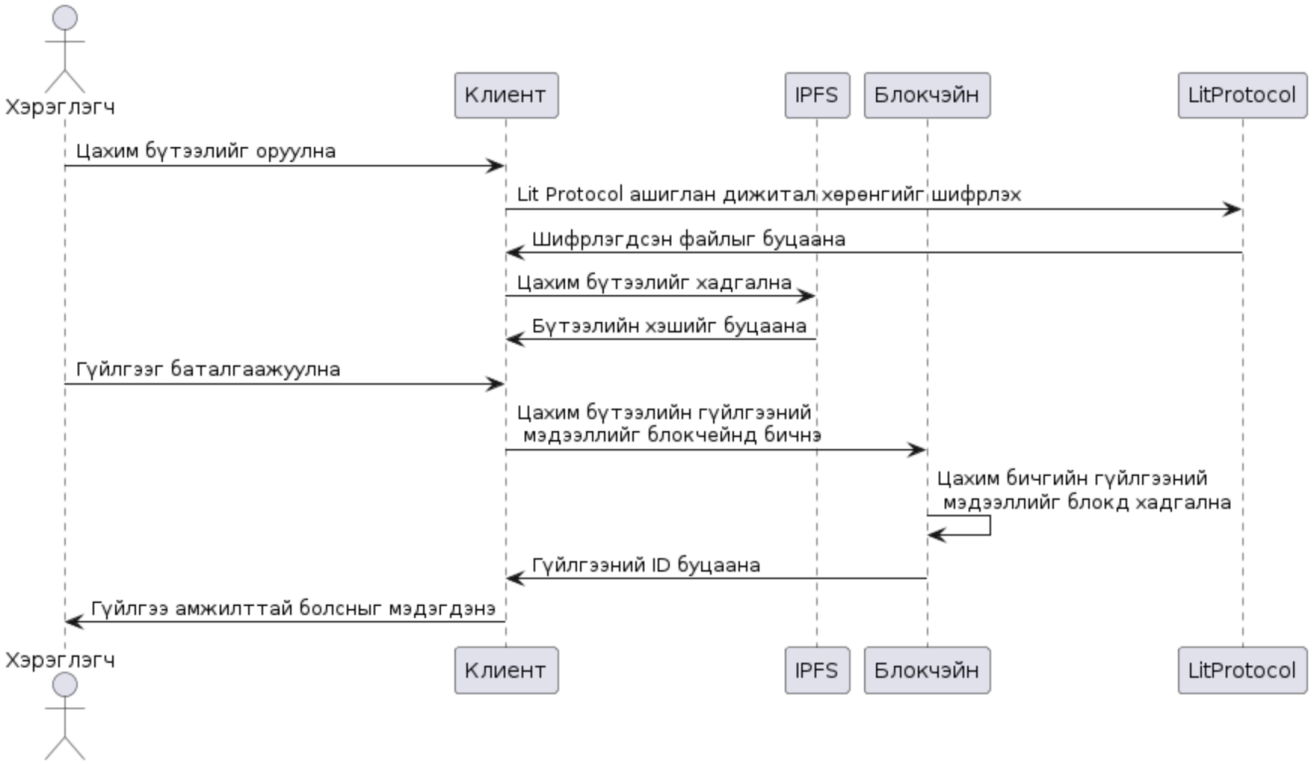


Зураг 2.1: Use-case диаграмм

2.3 Дарааллын диаграмм

2.3.1 Хэрэглэгч цахим бүтээл оруулах дарааллын диаграмм

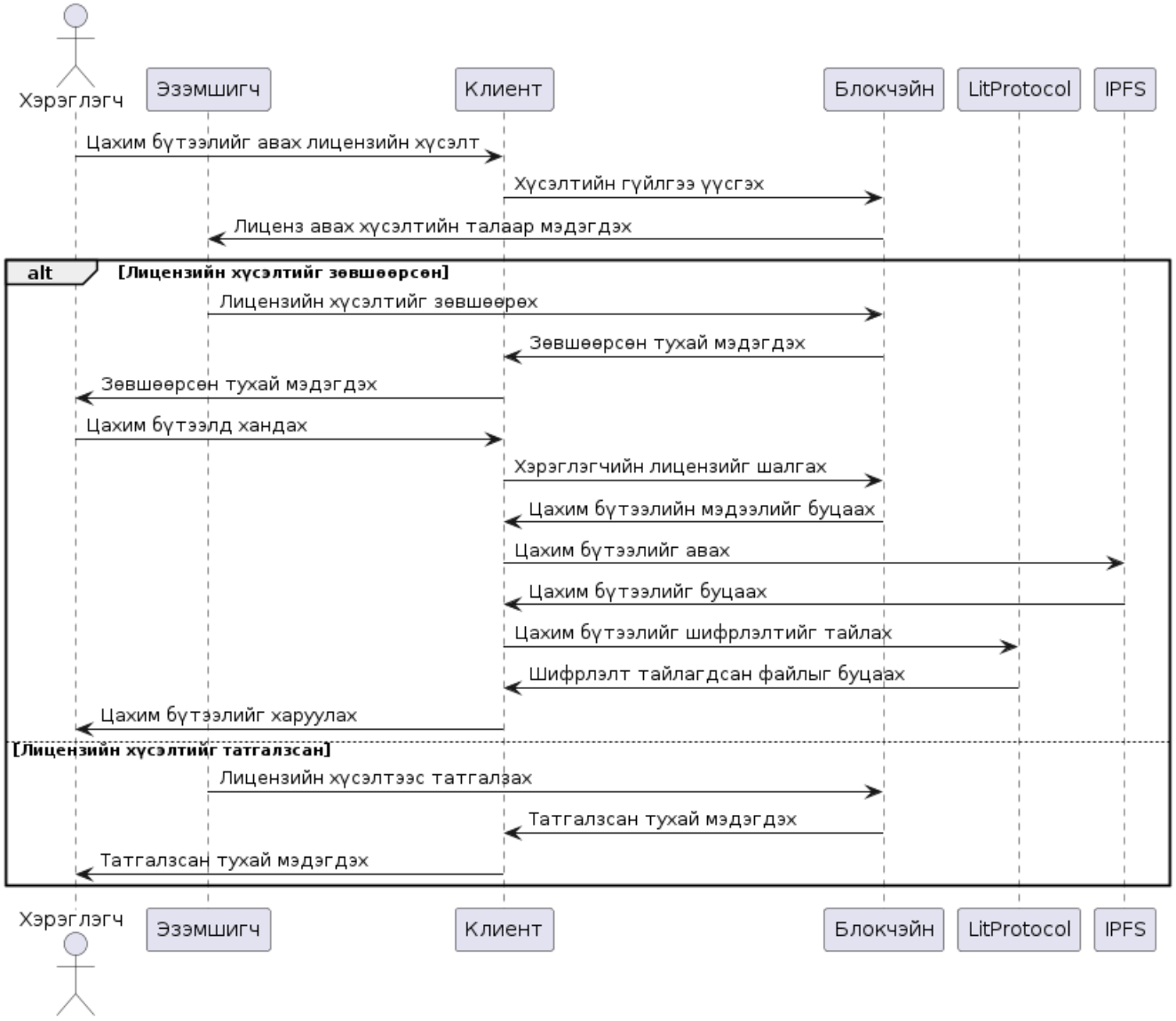
Энэхүү дарааллын диаграмм нь цахим бүтээлийг төвлөрсөн бус системд байршуулах, Lit протокол ашиглан шифрлэх, IPFS сүлжээнд хадгалах, блокчэйн дээр бүртгэх үйл явцыг дүрсэлсэн болно.



Зураг 2.2: Хэрэглэгч цахим бүтээл оруулах дарааллын диаграмм

2.3.2 Хэрэглэгч цахим бүтээлд хандах эрх хүсэх дарааллын диаграмм

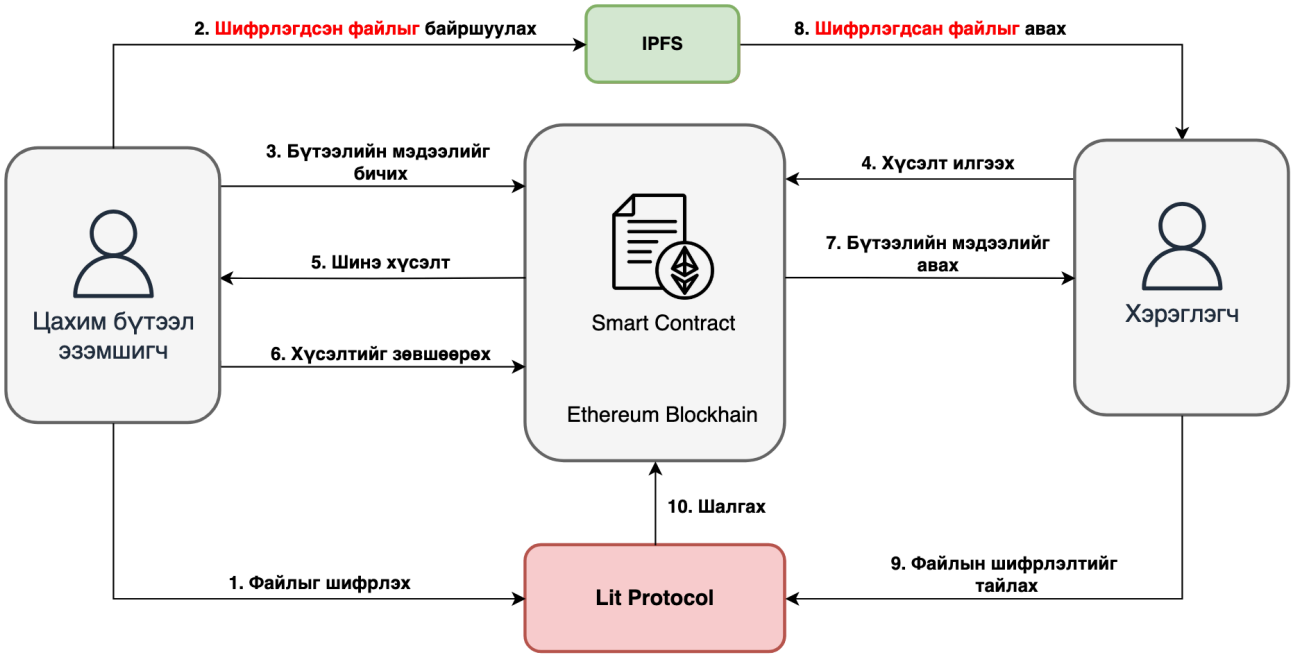
Энэхүү дарааллын диаграмм нь эзэмшигчээс цахим бүтээлийн лиценз авах хүсэлт гаргасан хэрэглэгч, дараа нь хүсэлтийг зөвшөөрөх эсвэл татгалзах үйл явцыг дүрсэлсэн болно.



Зураг 2.3: Хэрэглэгч цахим бүтээлийг хүсэх дарааллын диаграмм

2.4 Архитектур

Энэхүү төслийн фронтэнд хэсэг нь NextJS-н ашигласан тул сервер талын рендер хийж байгаа ба хэрэглэгчийн оруулсан цахим бүтээл болон лицензийн мэдээллийг этереум блокчэйн сүлжээнд байршсан ухаалаг гэрээнд бичих болон унших үйлдлийг хийх юм. Мөн хэрэглэгчийн оруулсан цахим бүтээлийн файлыг IPFS сүлжээнд шифрлэн байршуулж, шифрлэлтийн түлхүүрийг Lit сүлжээнд хадгална. Шифрлэлтийн түлхүүрийг хандалтын хяналтын нөхцөлд тодорхойлсны дагуу ухаалаг гэрээгээр зөвшөөрөлтэй эсэхийг шалган авч файлын шифрлэлтийг тайлна.



Зураг 2.4: Системийн ерөнхий архитектур

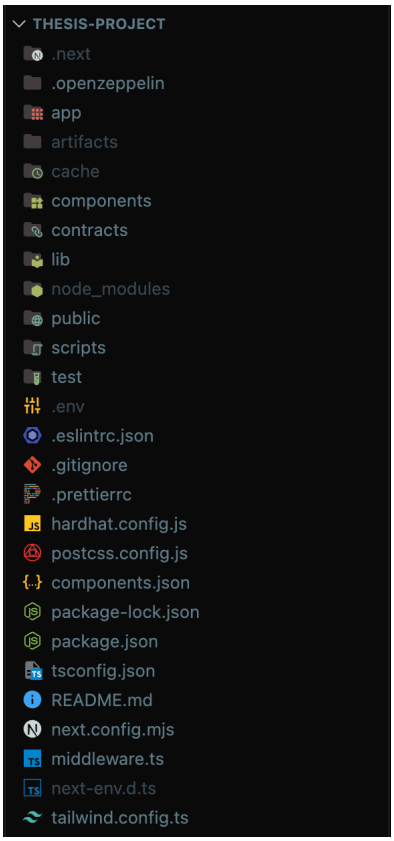
3. СИСТЕМИЙН ХЭРЭГЖҮҮЛЭЛТ

Системийн хэрэгжүүлэлт бүлгийн хүрээнд системийн хөгжүүлэлтийг дээрх бүлэг сэдвүүд дээр тодорхойлсон шинжилгээ ба зохиомжийн дагуу хийлээ.

3.1 Хөгжүүлэлт

3.1.1 Хөгжүүлэлтийн орчныг бэлдэх

Энэхүү судалгааны ажлын практик хэсэгт би NextJS, Hardhat, Pinata IPFS, Lit Protocol, Wagmi, Tailwind CSS зэргийг ашиглан хөгжүүлэлт хийх билээ. NextJS нь монолитик төсөл хийхэд тохиромжтой ба төслийн ухаалаг гэрээн хөгжүүлэлт, клайнт талуудыг нэг repository-д хадгалж байгаа. Version Control System-ээр Github-г сонгосон юм. Кодын фолдер бүтэц нь дараах байдлаар байна.



Зураг 3.1: Хөгжүүлэлтийн орчин

- **components** - React компонентууд
- **lib** - Хэрэглэгчийн талын шаардлагатай код туслах функцүүд
- **app** - NextJS дээрх хуудаснууд
- **public** - Статик зураг, файлууд
- **scripts** - Ухаалаг гэрээний хөгжүүлэлттэй холбоотой файлууд
- **contracts** - Ухаалаг гэрээний файлууд

3.1.2 Ухаалаг гэрээн хөгжүүлэлт

Миний төсөл нэг ухаалаг гэрээнээс бүтнэ. Ухаалаг гэрээг Solidity хэл дээр бичсэн бөгөөд Ethereum блокчэйн дээр байршуулсан. Уг ухаалаг гэрээ нь цахим файлууд болон тэдгээртэй холбоотой лицензүүдийг төлөөлдөг Файл, Лиценз ба Хүсэлт гэсэн гурван бүтцийг тодорхойлсон. Файл бүтэц нь id, эзэмшигчийн хаяг, файлын нэр, дэлгэрэнгүй, ангилал, шифрлэсэн файлын хэш, файлын хэмжээ, бүтээлийн үнэ, үүсгэсэн хугацаа зэрэг атрибутуудыг агуулна. Лиценз бүтэц нь лицензийн дугаар, бүтээл эзэмшигчийн хаяг, лиценз эзэмшигчийн хаяг, файлын нэр, дэлгэрэнгүй, ангилал, шифрлэсэн файлын хэш, файлын хэмжээ, лиценз үүсгэсэн хугацаа зэрэг атрибутуудыг агуулна. Мөн дараах функцүүдтэй:

createFile	Цахим бүтээлийн мэдээллийг бичих
getAllPublicFiles	Оруулсан бүх цахим бүтээлийн мэдээллийг авах
getAllUserFiles	Хэрэглэгчийн оруулсан цахим бүтээлийн мэдээллийг авах
getAllUserLicenses	Хэрэглэгчийн эзэмшиж буй цахим бүтээлийн лицензүүдийг авах
getPublicFileById	Цахим бүтээлийн мэдээллийг дугаараар нь авах
requestLicense	Цахим бүтээлийг авах бүтээлийн эзэмшигч рүү лицензийн хүсэлт илгээх
approveLicenseRequest	Өөрийн цахим бүтээлд ирсэн хүсэлтийг зөвшөөрөх
rejectLicenseRequest	Өөрийн цахим бүтээлд ирсэн хүсэлтээс татгалзах
getFileOwnerLicenseRequests	Бүтээлийн эзэмшигчид ирсэн хүсэлтүүдийг авах
isFileOwnedOrLicensed	Цахим бүтээлийн эзэмшигч эсэх эсвэл түүний лицензийг авсан эсэхийг шалгах
generateUniqueLicense	Лицензэд өвөрмөц дугаар бий болгох

Table 3.1: Ухаалга гэрээний функцүүд

3.1.3 Ухаалаг гэрээг этереум блокчэйн сүлжээнд байршуулах

```
const { ethers } = require('hardhat');

async function deployContract() {
  let contract;

  try {
    contract = await ethers.deployContract('LicenseMarketplace');
    await contract.waitForDeployment();

    console.log('Contracts deployed successfully.');
    return contract;
  } catch (error) {
    console.error('Error deploying contracts:', error);
    throw error;
  }
}

async function main() {
  let contract;

  try {
    contract = await deployContract();
    await saveContractAddress(contract);

    console.log('Contract deployment completed successfully.');
  } catch (error) {
    console.error('Unhandled error:', error);
  }
}

main().catch((error) => {
  console.error('Unhandled error:', error);
  process.exitCode = 1;
});
```

Код 3.1: Ухаалаг гэрээг блокчэйнд байршуулах

3.1.4 Lit protocol-н файл шифрлэлт болон хандалтын хяналтын нөхцөл

Lit Protocol-оор дамжуулан блокчэйнд байршуулсан ухаалаг гэрээнд бичсэн isFileOwnedOrLicensed функц хэрэглэгчийн крипто хэтэвчийн хаяг, файлын дугаараар файлыг зөвхөн бүтээл эзэмшигч эсвэл тухайн бүтээлд лиценз буюу хандах зөвшөөрөлтэй хэрэглэгч эсэхийг баталгаажуулан файлын шифрлэлтийг тайлах түлхүүрийг авах боломжтойгоор хандалтын хяналтын нөхцөлийг тодорхойлон клиент талд файлыг шифрлэнэ. Файлыг шифрлэсний дараа Lit сүлжээ нь хандалтын хяналтын нөхцөлийн тодорхойлсноор хэн шифрлэлтийг тайлах түлхүүр авахыг зөвшөөрнө. Файлын шифрлэлтийн түлхүүр үүсгэх болон авахын тулд хэрэглэгчийн

крипто хэтэвчээр баталгаажуулна.

```
1
2  const client = new LitJsSdk.LitNodeClient({});
3  const chain = "sepolia";
4
5  const createAccessControlCondition = (id: string) => {
6     return [
7        {
8           contractAddress: licenseValidationContract.
               contractAddress,
9           chain: "sepolia",
10          functionName: "isFileOwnedOrLicensed",
11          functionParams: [":userAddress", id],
12          functionAbi: {
13             inputs: [
14                {
15                   internalType: "address",
16                   name: "_user",
17                   type: "address",
18                },
19                {
20                   internalType: "uint256",
21                   name: "_fileId",
22                   type: "uint256",
23                },
24             ],
25             name: "isFileOwnedOrLicensed",
26             outputs: [
27                {
28                   internalType: "bool",
29                   name: "isOwned",
30                   type: "bool",
31                },
32             ],
33             stateMutability: "view",
34             type: "function",
35          },
36          returnValueTest: {
37             key: "isOwned",
38             comparator: "=",
39             value: "true",
40          },
41       },
42    ];
43 };
44 class Lit {
45    litNodeClient: any;
46
47    async connect() {
48       await client.connect();
49       this.litNodeClient = client;
50    }
51
52    async encryptFile(id: string, file: any) {
53       if (!this.litNodeClient) await this.connect();
54       const authSig = await LitJsSdk.checkAndSignAuthMessage({
55          chain: "sepolia",
56          nonce: this.litNodeClient.getLatestBlockhash() as string
               ,
57       });
```

```
58 |
59 |       const encryptedZip = await LitJsSdk.
   |          encryptFileAndZipWithMetadata({
60 |          evmContractConditions: createAccessControlCondition(
   |             String(id)),
61 |          authSig,
62 |          chain: "sepolia",
63 |          file: file,
64 |          litNodeClient: this.litNodeClient,
65 |          readme: "Encrypted␣file",
66 |       });
67 |
68 |       const encryptedBlob = new Blob([encryptedZip], {
69 |          type: "text/plain",
70 |       });
71 |       const encryptedFile = new File([encryptedBlob], file.name);
72 |
73 |       return encryptedFile;
74 |    }
75 |
76 |    async decryptFile(file: any) {
77 |       if (!this.litNodeClient) await this.connect();
78 |       const authSig = await LitJsSdk.checkAndSignAuthMessage({
79 |          chain: "sepolia",
80 |          nonce: this.litNodeClient.getLatestBlockhash() as string
   |             ,
81 |       });
82 |
83 |       const { decryptedFile } = await LitJsSdk.
   |          decryptZipFileWithMetadata(
84 |          {
85 |             file: file,
86 |             litNodeClient: this.litNodeClient,
87 |             authSig: authSig,
88 |          },
89 |       );
90 |
91 |       return decryptedFile;
92 |    }
93 |}
94 |const lit = new Lit();
95 |
96 |export default lit;
```

Код 3.2: Lit protocol-н файл шифрлэлт болон хандалтын хяналтын нөхцөл

3.1.5 Pinata API-ээр дамжуулан файлыг IPFS-д хадгалах

Pinata нь IPFS дээрх үйлдлүүдийг хийх боломжийг олгодог бөгөөд Pinata API-г ашиглан хэрэглэгч контент хадгалах боломжтой болно. Pinata API-г ашиглахын тулд хэрэглэгч Pinata дээр бүртгүүлж, API түлхүүр үүсгэх шаардлагатай.

Pinata API ашиглан файлыг IPFS-д байршуулах функцийг тодорхойлсон. Энэ нь POST

хүсэлтийг файлын өгөгдөл болон шаардлагатай аюулгүй байдлын header мэдээллийн хамт илгээж, серверээс хариу хүлээн авсны дараа өгөгдлийг JSON форматаар буцаана.

```
async function pinFileToIPFS(file: File): Promise<any> {
    const formData = new FormData();
    formData.append('file', file);

    const res = await fetch('https://api.pinata.cloud/pinning/
       pinFileToIPFS', {
       method: 'POST',
       headers: {
          pinata_api_key: process.env.NEXT_PUBLIC_PINATA_API_KEY!,
          pinata_secret_api_key: process.env.
             NEXT_PUBLIC_PINATA_API_SECRET!,
       },
       body: formData,
    });
    return res.json();
}
```

Код 3.3: Файлыг IPFS-д хадгалах

3.1.6 Ухаалаг гэрээнээс өгөгдлийг унших

Хэрэглэгчийн ethereum хэтэвчний хаяг, ухаалаг гэрээний хаяг болон ухаалаг гэрээний ABI (ухаалаг гэрээ ба ухаалаг гэрээнүүдтэй харилцах программуудын хоорондын харилцааг тодорхойлдог интерфейс) авч, Wagmi сангийн ухаалаг гэрээнээс өгөгдөл авахад зориулагдсан useReadContract hook-г ашиглана. Ethereum блокчэйн дээр байрласан ухаалаг гэрээнээс өгөгдөл авах бөгөөд гэрээ нь 'getAllUserFiles' гэсэн нэртэй функцтэй байх ёстой. Энэ функц нь тухайн хэрэглэгчийн эзэмшдэг цахим бүтээлүүдийг буцаана.

```
const { address } = useAccount();
const {
  data: userFiles,
  isLoading,
  error,
} = useReadContract({
  address: licenseValidationContract.contractAddress as `0x${
     string}`,
  abi: licenseValidationAbi.abi,
  functionName: 'getAllUserFiles',
  account: address,
}) as { data: UploadedFile[]; isLoading: boolean; error: any };
```

Код 3.4: Ухаалаг гэрээний өгөгдлийг унших

3.1.7 Ухаалаг гэрээнд цахим бүтээлийн өгөгдлийг бичих

Wagmi-н тодорхойлсон React Hook-г ашиглан цахим бүтээл эзэмшигчийн оруулсан бүтээлийн өгөгдлийг ухаалаг гэрээнд бичнэ. Товчхондоо, файл байршуулж, нууцлал, Ethereum ухаалаг гэрээтэй харилцах үйл ажиллагааг React компонент дотор хийнэ. Энэ нь хэрэглэгч Ethereum түрийвчтэй холбогдсон эсэхийг баталгаажуулж, Lit Protocol ашиглан файлыг шифрлэж, шифрлэгдсэн файлыг IPFS-д байршуулж, ухаалаг гэрээнд өгөгдлийг бичнэ. Уг үйл явцын туршид төлөвүүдийн мэдээллийг хэрэглэгчдэд шинэчлэн мэдэгдэнэ.

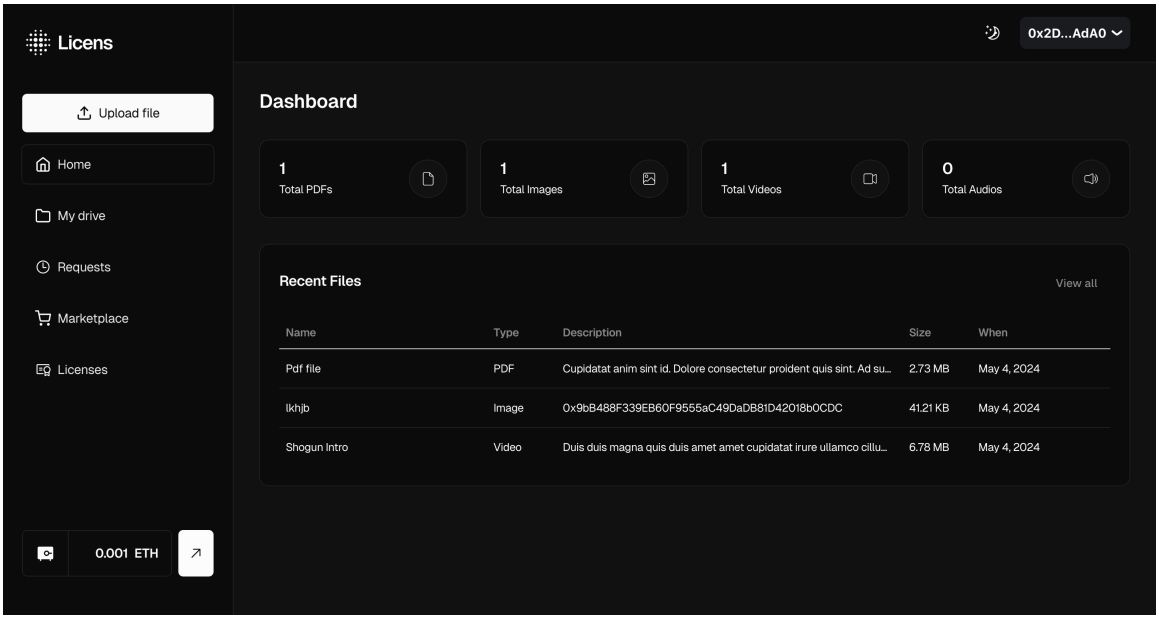
```
const { writeContract, isPending, data: hash } = useWriteContract
    ();
const { isLoading, isSuccess, isError } =
    useWaitForTransactionReceipt({
    hash,
});
const { isConnected, address } = useAccount();

async function onSubmit(data: z.infer<typeof formSchema>) {
    if (!isConnected) {
        connect({ connector: injected() });
    }

    if (!file) {
        toast({
            variant: "destructive",
            description: "Please␣select␣a␣file␣to␣upload.",
        });
        return;
    }

    setUploading(true);

    try {
        if (!thumbnail) {
            if (data.category === "Video") {
                const fileUrl = URL.createObjectURL(file);
                const thumbnailVideo = await getThumbnailForVideo(
                    fileUrl,
                    file.name,
                );
                setThumbnail(thumbnailVideo as File);
            }
        }
        let thumbnailRes = null;

        if (thumbnail) {
            thumbnailRes = await pinFileToIPFS(thumbnail);
        }

        const encryptedFile = await lit.encryptFile(String(fileId),
            file);
        const res = await pinFileToIPFS(encryptedFile);

```

```
        if (res.isDuplicate) {
            if (fileInputRef.current) {
                fileInputRef.current.value = "";
            }
            toast({
                variant: "destructive",
                description: "This file has already been uploaded.",
            });
        } else {
            writeContract({
                abi: licenseValidationAbi.abi,
                address:
                    licenseValidationContract.contractAddress as `0x${
                        string}`,
                functionName: "createFile",
                args: [
                    data.fileName,
                    data.description,
                    data.category,
                    res.IpfsHash,
                    file.size,
                    thumbnailRes?.IpfsHash || "",
                    parseEther(String(data.price)),
                ],
            });
        }
    } catch (error) {
        console.error("Error uploading file:", error);
        toast({
            variant: "destructive",
            description:
                "An error occurred while uploading the file. Please
                    try again.",
        });
    } finally {
        setUploading(false);
        refetch();
    }
}
```

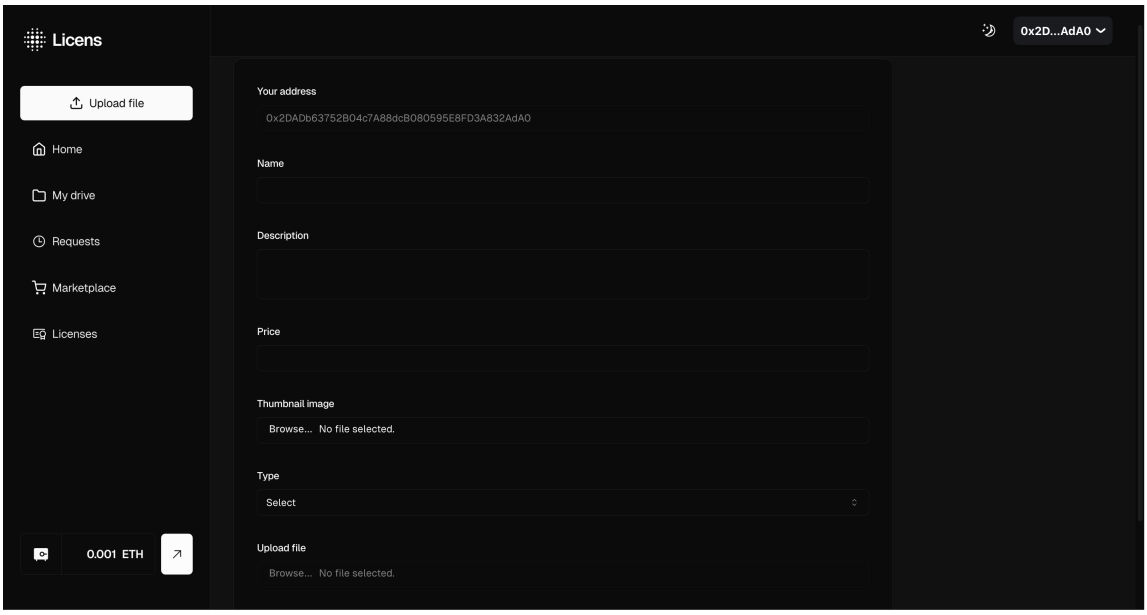
Код 3.5: Ухаалаг гэрээний өгөгдлийг унших

3.2 Үр дүн

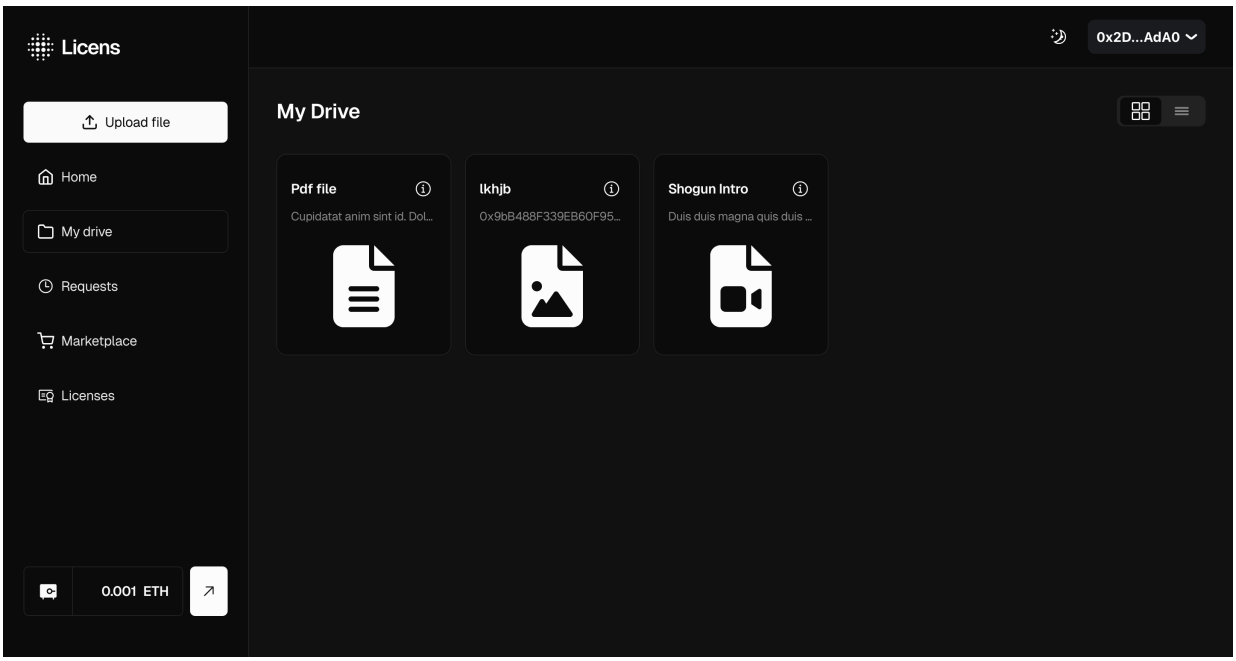
Хөгжүүлэлтийг дээрх бүлэг сэдвүүд дээр тодорхойлсон шинжилгээ, зохиомжийн дагуу хийсэн болно. Системийн интерфейс дараах байдлаар харагдана.



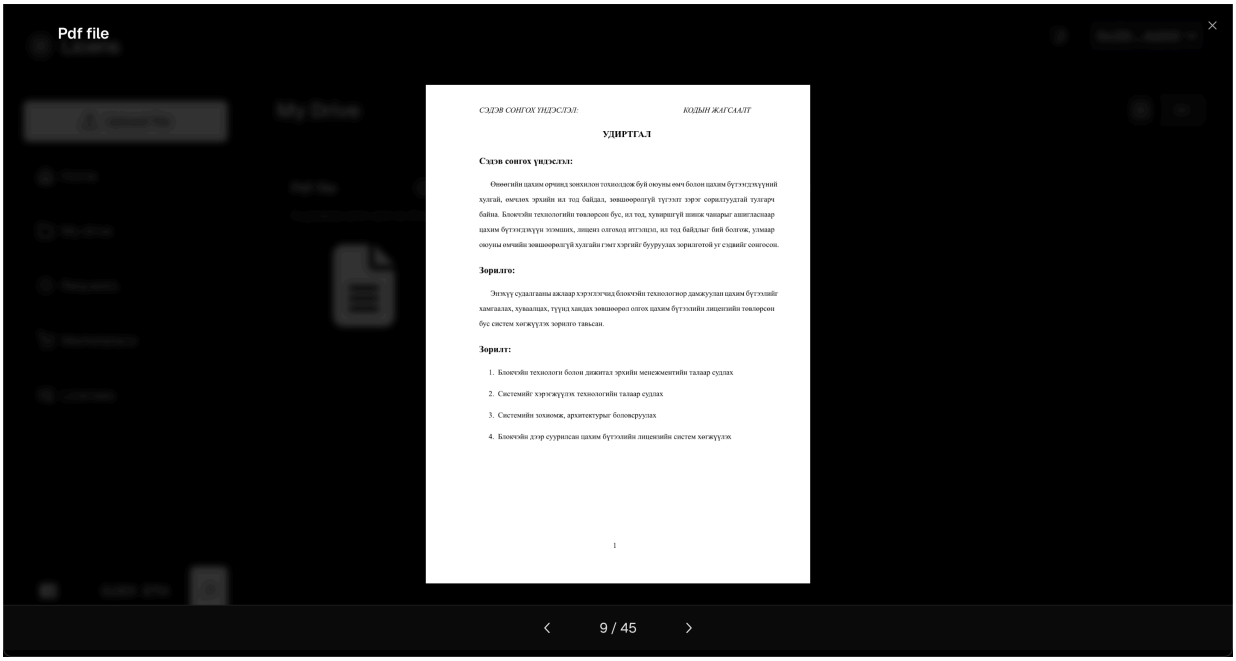
Зураг 3.2: Хэрэглэгчийн үндсэн хуудас



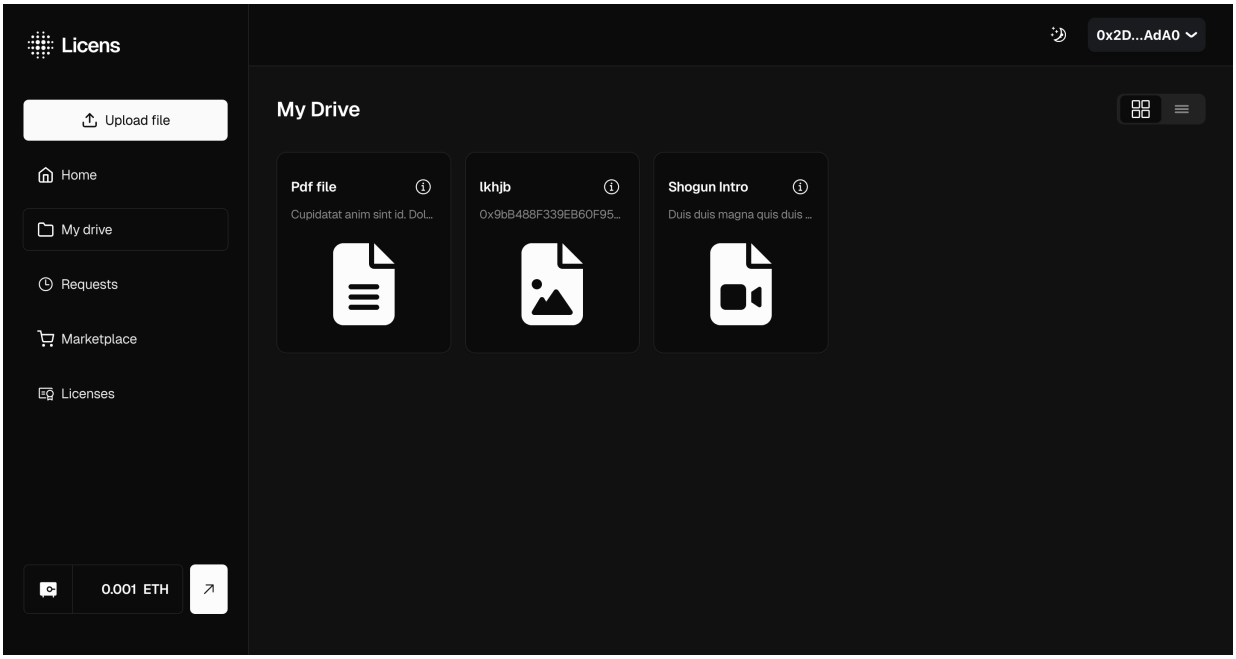
Зураг 3.3: Цахим бүтээл оруулах хуудас



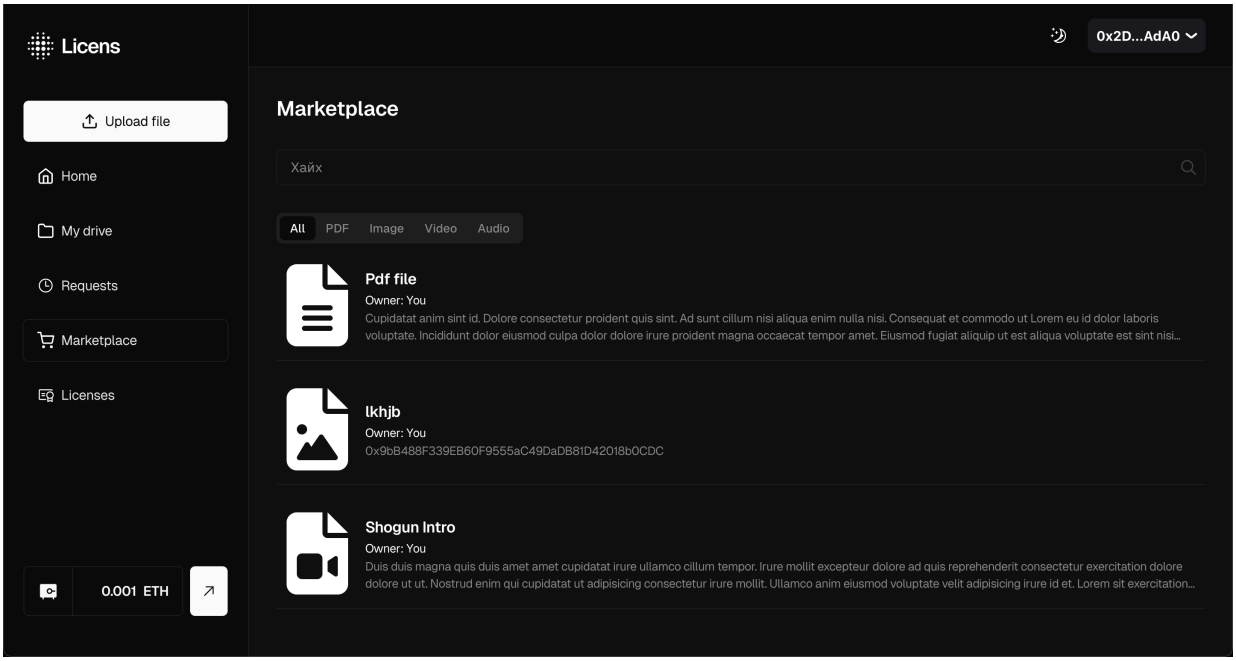
Зураг 3.4: Миний сан хуудас



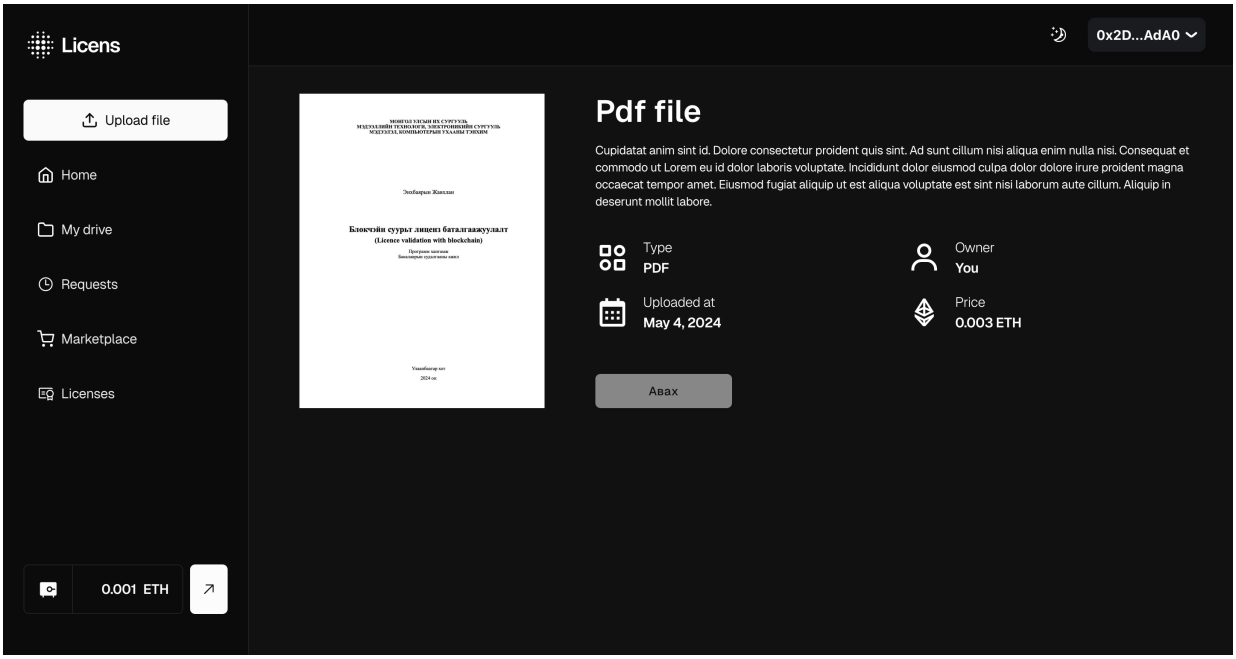
Зураг 3.5: PDF файл харах



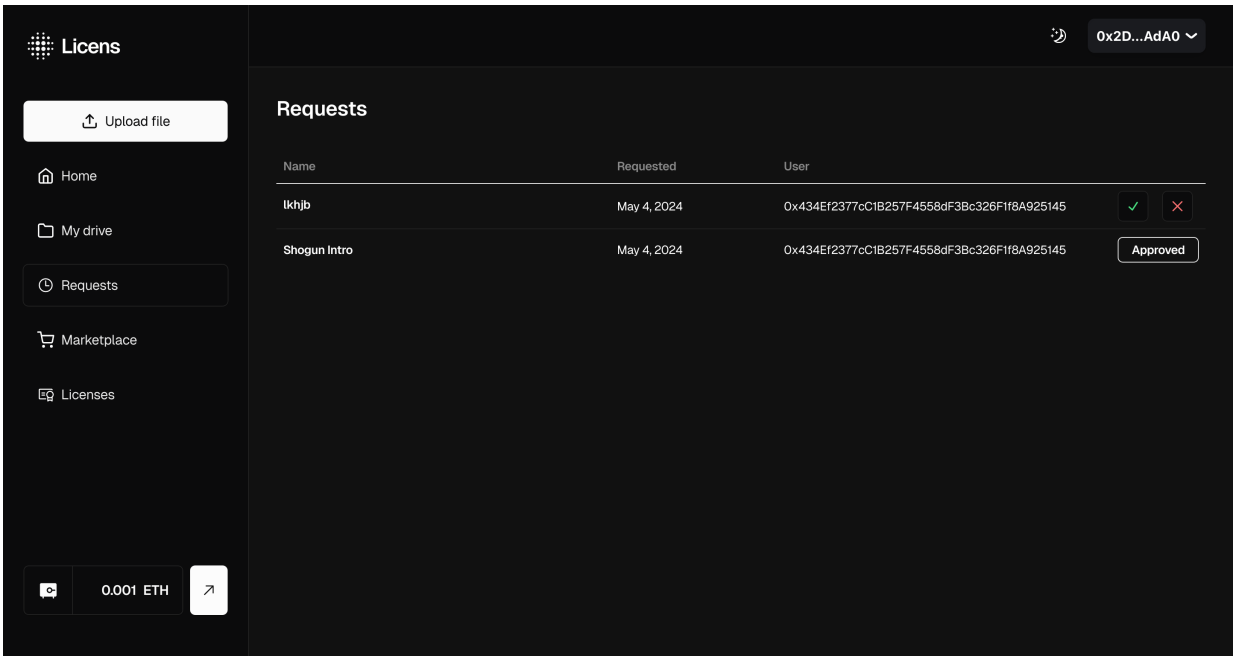
Зураг 3.6: Миний сан хуудас



Зураг 3.7: Marketplace хуудас



Зураг 3.8: Marketplace дэх бүтээлийн хуудас



Зураг 3.9: Лиценз хүсэлтийн хуудас

Дүгнэлт

Бакалаврын судалгааны ажлын хүрээнд блокчэйн технологи болон дижитал эрхийн менежментийн талаар судалсан. Энэхүү судалж суралцсан мэдлэгээ ашиглан практикт цахим бүтээлийн лицензийн системийг бүтээхээр зорьсон. Уг системийн гол зорилго нь хэрэглэгчид блокчэйн технологиор дамжуулан цахим бүтээлийг хамгаалах, хуваалцах, түүнд хандах зөвшөөрөл олгох төвлөрсөн бус, ил тод систем юм.

Энэхүү системийг эхнээс нь алхам алхмаар шаардлагаас нь хэрэгжүүлэлт хүртэл хийж гүйцэтгэсэн ба системийн функциональ ба функциональ бус шаардлагыг боловсруулж түүнээс системийн статик болон динамик загварыг гаргасан билээ. Үр дүнд нь блокчэйн, ухаалаг гэрээ болон IPFS технологиудыг ашиглан найдвартай, ил тод, төвлөрсөн бус системийг бүтээлээ. Түүнчлэн Lit протоколыг ашиглан файлын нууцлалыг сайжруулсан нь платформын аюулгүй байдал, нууцлалыг улам бэхжүүлсэн.

Гэсэн хэдий ч, судалгааны явцад тулгарсан сорилтууд байсан. Нэг том бэрхшээл нь ухаалаг гэрээ боловсруулах болон системд Lit протоколыг нэгтгэхэд шаардлагатай байсан бөгөөд үүний тулд блокчэйн хөгжүүлэлт, криптографын талаар гүнзгий ойлголт шаардлагатай байв. Түүнчлэн, системийн хэмжээнд аюулгүй байдлыг хангахын зэрэгцээ үр ашиг, хурдыг хадгалахад тогтмол сорилт тулгарч байв.

Цаашид ухаалаг гэрээний функцүүдийг оновчтой болгох, гүйлгээний шимтгэлийг буруулах талаар нарийвчилсан судалгаа хийх нь зүйтэй гэж үзэж байна. Түүнчлэн блокчэйний өөр өөр сүлжээ болон шифрлэлтийн аргуудыг судлах нь системийн цар хүрээ, үр ашгийг дээшлүүлэхэд тустай байж болох юм.

Bibliography

[1] Adam Hayes, Blockchain Facts: What Is It, How It Works, and How It Can Be Used. (December 15, 2023) https://www.investopedia.com/terms/b/blockchain.asp

[2] Scott Nevil, Distributed Ledger Technology (DLT): Definition and How It Works. (May 31, 2023) https://www.investopedia.com/terms/d/distributed-ledger-technology-dlt.asp

[3] Sundararajan S. UN Agencies Turn to Blockchain In Fight Against Child Trafficking. (Nov 13, 2017) https://www.coindesk.com/markets/2017/11/13/un-agencies-turn-to-blockchain-in-fight-against-child-trafficking/

[4] Zug Digital ID: Blockchain Case Study for Government Issued Identity. https://www.investopedia.com/terms/b/blockchain.asp

[5] What is digital rights management (DRM)?. https://business.adobe.com/blog/basics/digital-rights-management

А. КОДЫН ХЭРЭГЖҮҮЛЭЛТ

А.1 Ухаалаг гэрээ

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract LicenseMarketplace {
    address public owner;

    struct File {
        uint256 id;
        address fileOwner;
        string fileName;
        string description;
        string category;
        string fileCid;
        string imgUrl;
        uint256 fileSize;
        uint256 createdAt;
        uint256 price;
    }

    struct License {
        uint256 licenseNumber;
        uint256 fileId;
        address fileOwner;
        address buyer;
        string fileName;
        string description;
        string category;
        string fileCid;
        string imgUrl;
        uint256 fileSize;
        uint256 createdAt;
        uint256 price;
    }

    struct LicenseRequest {
        uint256 requestId;
        uint256 fileId;
        string fileName;
        address requester;
        address fileOwner;
        uint256 requestedAt;
        bool isApproved;
        uint256  assetPrice;
    }

       mapping(address => File[]) private userFiles;
       mapping (address => License[]) private fileLicenses;
       mapping (uint256 => LicenseRequest) private licenseRequests
          ;

       mapping(uint256 => bool) public usedLicenses;

       mapping(address => uint256) private userFund;
       mapping(address => uint256) private userFundWithdrawable;

```

```
55        File[] public publicFiles;
56
57        uint256 public fileId;
58        uint256 public requestId;
59
60        event FileShared(address indexed  owner, string fileName);
61        event FileLicense(address indexed  owner, address indexed
              buyer, string fileName);
62        event LicenseRequestCreated(uint256 indexed requestId,
              uint256 fileId, address requester, address fileOwner);
63        event LicenseRequestApproved(uint256 indexed requestId,
              uint256 fileId, address requester, address fileOwner);
64        event LicenseRequestRejected(uint256 indexed requestId,
              uint256 fileId, address requester, address fileOwner);
65
66        modifier onlyOwner(){
67            require(msg.sender == owner,  "Only␣the␣owner␣can␣call␣
                  this␣function");
68            _;
69        }
70
71        constructor() {
72            owner = msg.sender;
73        }
74
75        function createFile(string memory _fileName, string memory
              _description, string memory _category,  string memory
              _fileCid, uint256  _fileSize, string memory _imgUrl,
              uint256 _price) public {
76            fileId++;
77
78            File memory newFile = File({
79                id: fileId,
80                fileOwner: msg.sender,
81                fileName: _fileName,
82                description: _description,
83                category: _category,
84                fileCid: _fileCid,
85                imgUrl: _imgUrl,
86                fileSize: _fileSize,
87                createdAt: block.timestamp,
88                price: _price
89            });
90
91            userFiles[msg.sender].push(newFile);
92            publicFiles.push(newFile);
93
94            emit FileShared(msg.sender, _fileName);
95        }
96
97        function requestLicense(uint256 _fileId, address _fileOwner
              , string memory _fileName, uint256 _assetPrice) public
              payable {
98            require(!isFileOwnedOrLicensed(msg.sender, _fileId), "
                  User␣already␣owns␣or␣has␣a␣license␣for␣this␣file");
99            require(msg.value == _assetPrice, "Price␣not␣met");
100
101           requestId++;
102           LicenseRequest memory newRequest = LicenseRequest({
103               requestId: requestId,
104               fileId: _fileId,
```

```
            fileName: _fileName,
            requester: msg.sender,
            fileOwner: _fileOwner,
            requestedAt: block.timestamp,
            isApproved: false,
            assetPrice: _assetPrice
        });

        licenseRequests[requestId] = newRequest;
        userFund[_fileOwner] += msg.value;

        emit LicenseRequestCreated(requestId, _fileId, msg.
            sender, _fileOwner);
    }


    function approveLicenseRequest(uint256 _requestId) public {
        LicenseRequest storage request = licenseRequests[
            _requestId];
        require(request.fileOwner == msg.sender, "Only␣the␣file␣
            owner␣can␣approve␣the␣request");
        require(!request.isApproved, "Request␣already␣approved")
            ;

        request.isApproved = true;

        File memory file = getPublicFileById(request.fileId);
        uint256 licNum = generateUniqueLicense();

        License memory newLicense = License({
            licenseNumber: licNum,
            fileId: file.id,
            fileOwner: file.fileOwner,
            buyer: request.requester,
            fileName: file.fileName,
            description: file.description,
            category: file.category,
            fileCid: file.fileCid,
            imgUrl: file.imgUrl,
            fileSize: file.fileSize,
            createdAt: block.timestamp,
            price: file.price
        });

        fileLicenses[request.requester].push(newLicense);

        userFund[msg.sender] -= file.price;
        userFundWithdrawable[msg.sender] += file.price;

        emit FileLicense(file.fileOwner, request.requester, file
            .fileName);
        emit LicenseRequestApproved(_requestId, request.fileId,
            request.requester, request.fileOwner);
    }

    function rejectLicenseRequest(uint256 _requestId) public  {
        LicenseRequest storage request = licenseRequests[
            _requestId];
        require(request.fileOwner == msg.sender, "Only␣the␣file␣
            owner␣can␣reject␣the␣request");
```

```
157            require(!request.isApproved, "Request already approved");
158
159            userFundWithdrawable[request.requester] +=  request.
                   assetPrice;
160            userFund[msg.sender] -= request.assetPrice;
161
162
163            delete licenseRequests[_requestId];
164            emit LicenseRequestRejected(_requestId, request.fileId,
                   request.requester, request.fileOwner);
165        }
166
167        function getAllUserFiles() public view returns(File[]
               memory) {
168            return userFiles[msg.sender];
169        }
170
171        function getAllUserLicenses() public view returns(License[]
                memory) {
172            return fileLicenses[msg.sender];
173        }
174
175     function getPublicFiles() public view returns (File[] memory)
            {
176          return publicFiles;
177     }
178
179     function generateUniqueLicense() internal returns (uint256) {
180          uint256 randomNumber = uint256(keccak256(abi.
                encodePacked(block.timestamp, blockhash(block.number)
                , msg.sender)));
181          uint256 license = randomNumber % 10000000000;
182
183          while (usedLicenses[license]) {
184              randomNumber = uint256(keccak256(abi.encodePacked(
                    randomNumber, block.timestamp)));
185              license = randomNumber % 10000000000;
186          }
187
188          usedLicenses[license] = true;
189
190          return license;
191        }
192
193    function validateLicense(uint256 licenseNumber) external view
           returns (bool) {
194          return usedLicenses[licenseNumber];
195    }
196
197    function getPublicFileById(uint256 _id) public view returns (
           File memory) {
198          for (uint256 i = 0; i < publicFiles.length; i++) {
199              if (publicFiles[i].id == _id) {
200                    return publicFiles[i];
201              }
202          }
203          revert("Public file not found");
204        }
205
```

```
206         function getUserLicenseRequests() public view returns (
                LicenseRequest[] memory) {
207             uint256 count = 0;
208             for (uint256 i = 1; i <= requestId; i++) {
209                 if (licenseRequests[i].requester == msg.sender) {
210                     count++;
211                 }
212             }
213 
214             LicenseRequest[] memory userRequests = new
                    LicenseRequest[](count);
215             uint256 index = 0;
216             for (uint256 i = 1; i <= requestId; i++) {
217                 if (licenseRequests[i].requester == msg.sender) {
218                     userRequests[index] = licenseRequests[i];
219                     index++;
220                 }
221             }
222 
223             return userRequests;
224         }
225 
226     function getFileOwnerLicenseRequests() public view returns (
            LicenseRequest[] memory) {
227             uint256 count = 0;
228             for (uint256 i = 1; i <= requestId; i++) {
229                 if (licenseRequests[i].fileOwner == msg.sender) {
230                     count++;
231                 }
232             }
233 
234             LicenseRequest[] memory ownerRequests = new
                    LicenseRequest[](count);
235             uint256 index = 0;
236             for (uint256 i = 1; i <= requestId; i++) {
237                 if (licenseRequests[i].fileOwner == msg.sender) {
238                     ownerRequests[index] = licenseRequests[i];
239                     index++;
240                 }
241             }
242 
243             return ownerRequests;
244     }
245 
246     function isFileOwnedOrLicensed(address _user, uint256 _fileId)
             public view returns (bool) {
247             for (uint256 i = 0; i < userFiles[_user].length; i++) {
248                 if (userFiles[_user][i].id == _fileId) {
249                     return true;
250                 }
251             }
252 
253             for (uint256 j = 0; j < fileLicenses[_user].length; j++)
                     {
254                 if (fileLicenses[_user][j].fileId == _fileId) {
255                     return true;
256                 }
257             }
258 
259             return false;
```

```
260     }
261
262
263     function getFileId() public view returns (uint256) {
264           return fileId + 1;
265     }
266
267     function payTo(address to, uint256 amount) internal {
268         (bool success, ) = payable(to).call{ value: amount }('');
269         require(success, "Transfer␣fund␣failed");
270      }
271
272      function withdrawFund() public {
273         uint256 amountToWithdraw = userFundWithdrawable[msg.
               sender];
274         require(amountToWithdraw > 0, "No␣funds␣to␣withdraw");
275
276         userFundWithdrawable[msg.sender] = 0;
277         payTo(msg.sender, amountToWithdraw);
278      }
279
280      function getWithdrawableFund() public view returns (uint256)
            {
281         return userFundWithdrawable[msg.sender];
282      }
283
284      function getFund() public view returns (uint256) {
285         return userFund[msg.sender];
286      }
287 }
```

Код А.1: Ухаалаг гэрээ