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

Цэдэн-Ишийн Биндэрцэцэг

Програм хангамжийн зохиомжийн үлгэр загварууд ба түүний хэрэглээ

(Software design patterns and its application)

Програм Хангамж (D061302)
Үйлдвэрлэлийн дадлагын тайлан

Улаанбаатар хот

2025 оны 9 сар

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

Програм хангамжийн зохиомжийн үлгэр загварууд ба түүний хэрэглээ

(Software design patterns and its application)

Програм Хангамж (D061302)
Үйлдвэрлэлийн дадлагын тайлан

Удирдагч: _____ Х. Ганзориг

Хамтран удирдагч: _____ Б. Батням

Гүйцэтгэсэн: _____ Ц. Биндэрцэцэг (22B1NUM0027)

Улаанбаатар хот

2025 оны 9 сар

Зохиогчийн баталгаа

Миний бие Цэдэн-Ишийн Биндэрцэцэг нь ”Програм хангамжийн зохиомжийн үлгэр загварууд ба түүний хэрэглээ” сэдэвтэй дадлагын ажлыг гүйцэтгэсэн болохыг зарлаж дараах зүйлсийг баталж байна:

- Энэхүү ажлын аль нэг хэсгийг эсвэл бүхлээр нь ямар нэг их, дээд сургуулийн зэрэг горилохоор оруулаагүй болно.
- Бусдын хийсэн ажлаас хуулбарлаагүй, ашигласан бол ишлэл, зүүлт хийсэн.
- Ажлыг би өөрөө (хамтарч) хийсэн ба миний хийсэн ажил, үзүүлсэн дэмжлэгийг дадлагын ажилд тодорхой тусгасан.
- Ажилд тусалсан бүх эх сурвалжид талархаж байна.

Гарын үсэг: _____

Огноо: _____

Хавсралт 3

МУИС, ХШУИС–ИЙН МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ

Програм хангамжийн АНГИЙН
ОЮУТАН Б. Бат-Өлзий –ЫН
ДАДЛАГЫН АЖЛЫН УДИРДАГЧИЙН ТОДОРХОЙЛОЛТ

2021 оны .. сарын ..

Програм хангамжийн ангийн 18b1num3474 кодтой оюутан Б. Бат-Өлзий нь манай байгууллагад 2021 оны 06 сарын 07-ны өдрөөс 06 сарын 25-ны өдөр хүртэл мэргэшүүлэх дадлагыг батлагдсан удирдамж, ажлын төлөвлөгөөний дагуу гүйцэтгэлээ. Оюутан Б. Бат-Өлзий-ын удирдамжийн дагуу дадлагын ажлыг гүйцэтгэсэн байдал:

Б.Бат-Өлзий нь үйлдвэрлэлийн дадлагын хүрээнд React.js болон Next.js ашиглан хөгжүүлж байгаа вэб програм хангамжийн хөгжүүлэлтэд оролцсон. Өмнө нь React болон холбоотой технологиудыг ашиглаж байсан туршлагагүй тул судлах, суралцах зүйлс цөөнгүй бөгөөд өгсөн чиглэл, даалгаврын дагуу хичээл зүтгэлтэй, цаг тухай бүрт судалж бас суралцсан болно. Тэрээр React.js компонент болон hooks бичих зарчим, арга ажиллагаа мөн Material UI ашиглан хэрэглэгчийн интерфэйсийг бүтээхэд суралцаж бас хөгжүүлэлтэд оролцсон. Үүнээс гадна Continuous Integration хөгжүүлэлтийн аргачлал болон хэрэглэгчийн интерфэйсийн автоматжуулсан тест бичих зарчимтай танилцсан.

Бат-Өлзий нь өгсөн даалгаврыг цаг тухай бүрт нь биелүүлдэг, нээлттэй харилцаатай тул ойлгохгүй эсвэл чадахгүй байгаа зүйлээ чөлөөтэй асуудаг, тусламж авдаг давуу талтай, мөн UX сонирхдог тул бүтээгдэхүүний хэрэглэгчийн интерфэйс бүтээх ажилд бүтээлчээр оролцож олон үнэтэй санал өгч байсан. Дадлагын хугацаанд вэб хөгжүүлэлтийн тал дээр багагүй өсч хөгжсөн, туршлагатай болсон гэж дүгнэж байна.

Үнэлгээний санал: 10 оноо

Дадлагын удирдагч /

Дадлагын удирдагчийн талаарх мэдээлэл:

Table 1: Дадлагын төлөвлөгөө

№	Гүйцэтгэх ажил	Хугацаа	Биелэлт	Удирдагчийн үнэлгээ
1	Програм хангамжийн үлгэр загваруудын тухай судлах	08/11 - 08/15	дууссан	
2	Жава, Spring Boot, Spring MVC технологиудын тухай судлах	08/11 - 08/15	дууссан	
3	Auftragsverwaltung системийн классын диаграмыг гаргах	08/12 - 08/14	дууссан	
4	Auftragsverwaltung систем дээр ашигласан зохиомжийн үлгэр загваруудыг олж тогтоох	08/14 - 08/18	дууссан	
5	МУИС-ын дипломын ажлыг удирдах системийн шаардлагатай танилцах	08/18	дууссан	
6	Уг шаардлагын дагуу системийн зохиомжийг загварчлах	08/18 - 08/20	дууссан	
7	Системийн хэрэгжүүлэлтийг Жава технологи ашиглан гүйцэтгэх	08/20 - 08/22	дуусаагүй	
8	Банкны Excel хуулгыг боловсруулах модулийн шинжилгээг гүйцэтгэх	08/22	дууссан	
9	Уг шаардлага дээрээ үндэслэн зохиомж болон архитектурыг гаргах	08/23 - 08/24	дууссан	
10	Уг зохиомжийн дагуу модулийн хэрэгжүүлэлтийг Spring Boot фреймворк ашиглан гүйцэтгэх	08/25 - 08/29	дууссан	
11	Модулийг JUnit санг ашиглан тестлэх	08/27 - 08/30	дууссан	

ГАРЧИГ

ЗУРГИЙН ЖАГСААЛТ

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

УДИРТГАЛ

Миний бие Цэдэн-Ишийн Биндэрцэцэг нь үйлдвэрлэлийн дадлагын 3 долоо хоногийн хугацаанд програм хангамжийн зохиомжийн үлгэр загваруудын хүрээнд урвуу инженерчлэл-ийн аргачлал, Жава, Spring Boot, Spring MVC гэсэн технологиуд дээр голчлон ажилласан ба уг технологиуд ямар шалтгаанаар үүссэн, хөгжүүлэлтийн ямар арга барил ашигладаг, компаниуд хэрхэн үүн дээр хөгжүүлэлт хийж эцсийн бүтээгдэхүүнийг гаргадаг талаар судлах-ын тулд Java Spring Boot фрэймворк ашигладаг компани болох ”Монгол Ай Ди” байгууллагыг сонгон авч мэргэжлийн дадлагаа гүйцэтгэлээ.

Дадлагын эхний долоо хоногт би бараа захиалгийг зохицуулах ”Auftragsverwaltung” систем дээр урвуу инженерчлэлийн аргачлал ашиглан зохиомжийн үлгэр загваруудыг уг системд хэрхэн ашигласан байгааг олж тогтоосон. Үүний дараа МУИС-ийн дипломын ажлыг удирдах системийн шаардлагийн дагуу системийн зохиомжийг гаргаж, уг зохиомж дээр үндэс-лэн системийг Жава технологи ашиглан хэрэгжүүллээ.

Дадлагын сүүлийн долоо хоногт Монгол Ай Ди компанийн хөгжүүлж буй дотоод зээлийн системийн асуудал шийдвэрлэх хэсэг дээр ажилласан. Миний хариуцаж авсан модуль нь хэрэглэгчийн дансны Excel хуулгыг сервер рүү ачаалж, хуулга доторх өгөгдлийг боловсруу-лан системийн өгөгдлийн сантай уялдуулж, хэрэглэгчийн зээлийн мэдээллийг шинэчлэх үүрэгтэй. Уг модуль дээр ажиллахын тулд би Spring Boot, Spring MVC технологийн талаар судалгаа хийж, компанийн хөгжүүлэлтийн арга барилтай танилцаж, өгөгдсөн асуудлыг шийд-вэрлэлээ.

1. БАЙГУУЛЛАГЫН ТАНИЛЦУУЛГА

1.1 Товч танилцуулга

Монгол Ай Ди нь 2015 онд байгуулагдсан Монголын хамгийн анхны албан ёсны төлбөр тооцооны процессорын үйл ажиллагаа эрхлэх зөвшөөрөлтэйгөөр олон улсын стандар -тын дагуу төлбөр тооцоо, карт, мэдээллийн технологи, лоялти зэрэг чиглэлээр цогц үйл ажиллагаа явуулдаг “Финтек” (Fintech) компани бөгөөд банк санхүү болон IT чиглэлийн туршлагатай мэргэжлийн баг бүрэлдэхүүнтэй ба орчин үеийн дэвшилтэт технологийг Монголд хамгийн тэргүүнд нутагшуулан дэлгэрүүлсээр байна.

1.2 Ямар үйлчилгээ үзүүлдэг вэ?

Монгол Ай Ди нь дотоодын болон олон улсын зах зээлд нийцсэн технологиудыг нутагшуулж, блокчэйн, хиймэл оюун ухаан, биометр зэрэг шинэ дэвшилтэт шийдлүүдийг нэвтрүүлэхэд анхаарч ажилладаг. Тус компани нь Монголд анх удаа түлшний төлбөрийн карт, SoftPOS, HCE, DRM зэрэг дэвшилтэт системүүдийг амжилттай нэвтрүүлсэн туршлага -тай. Одоогийн байдлаар зээлийн үйлчилгээ үзүүлдэг.

1.3 Ямар систем дээр голчлон төвлөрдөг вэ?

Монгол Ай Ди нь 2025 оны байдлаар “Mongol ID” систем дээр голчлон төвлөрч байгаа бөгөөд энэ нь хэрэглэгчийн хувийн бүртгэл үүсгэсэнээр олон төрлийн үйлчилгээ, аппликейшн (Minu Chat, RedPoint, Minu Pay, Sonos Audiobooks гэх мэт) ашиглах боломж олгодог нэгтгэн төвлө -рүүлсэн платформ юм. Энэ нь санхүүгийн болон санхүүгийн бус олон талын аппликейшн, үйлчилгээ рүү нийлэмжтэй холбогддог горим дээр бүтээгдсэн бөгөөд хэрэглэгчид дахин бүртгүүлэхгүйгээр үйлчлүүлэхэд зориулагдсан.

2. ЗОРИЛГО БА ЗОРИЛТ

2.1 Зорилго

Энэхүү үйлдвэрлэлийн дадлагын ажлын хүрээнд програм хангамжийн зохиомжийн үлгэр загваруудыг судалж, тэдгээрийн онцлог, хэрэглээ, давуу болон сул талыг тодорхойлж, бодит төсөл болох ”Банкны Excel хуулга боловсруулах модуль”-ийг Spring Boot фреймворк ашиглан хэрэгжүүлнэ.

2.2 Зорилт

Энэхүү зорилгод хүрэхийн тулд дараах зорилтуудыг тавьж ажиллалаа:

- Програм хангамжийн зохиомжийн үлгэр загваруудын талаар судалгаа хийх;
- ...

3. ОНОЛЫН СУДАЛГАА

Програм хангамжийн зохиомж нь програм хангамжийн системийн бүтэц, түүний бүрэлдэхүүн хэсгүүдийн харилцан үйлчлэл, зан төлөвийг тодорхойлох үйл явц юм. Энэ нь програм хангамжийн хөгжүүлэлтийн амжилтын гол хүчин зүйл бөгөөд системийн чанар, уян хатан байдал, засвар үйлчилгээний чадамжид шууд нөлөөлдөг. Зохиомж нь системийн шаардлага, бизнесийн зорилго, техникийн хязгаарлалтыг ойлгох, эдгээрийг зохицуулсан шийдэл гаргахыг шаарддаг.

3.1 Програм хангамжийн зохиомжийн зарчмууд

Програм хангамжийн зохиомжийн зарчмууд нь сайн зохиомжийг бий болгоход чиглэсэн удирдамж, хамгийн сайн туршлагуудын цуглуулга юм. Эдгээр зарчмууд нь програм хангамжийн системийг уян хатан, дахин ашиглах боломжтой, засвар үйлчилгээ хийхэд хялбар болгоход тусалдаг. SOLID зарчмуудыг доор дурдлаа:

- **"Single Responsibility Principle"**: Класс эсвэл модуль нь зөвхөн нэг л шалтгаанаар өөрчлөг- дөх ёстой. Энэ нь кодыг илүү ойлгомжтой, засвар үйлчилгээ хийхэд хялбар болгодог.

- **"Open/Closed Principle"**: Програм хангамжийн элементүүд нь өргөтгөхөд нээлттэй, өөрч- лөхөд хаалттай байх ёстой. Энэ нь шинэ функц нэмэхдээ одоо байгаа кодыг өөрчлөхгүй байхыг шаарддаг.

- **"Liskov Substitution Principle"**: Суперклассын объектуудыг түүний дэд классын объектуудаар орлуулах боломжтой байх ёстой. Энэ нь полиморфизмын үндсэн зарчим бөгөөд кодыг илүү уян хатан болгодог.

- **"Interface Segregation Principle"**: Том интерфэйсүүдийг жижиг, тодорхой интерфэйсүүдэд хуваах ёстой. Ингэснээр клиент хэрэгцээгүй аргуудыг хэрэгжүүлэх шаардлагагүй болно.

- **"Dependency Inversion Principle"**: Өндөр түвшний модульууд нь доод түвшний модулиудаас бус харин илүү хийсвэр түвшнээс хамааралтай байх ёстой. Энэ нь кодыг уян хатан, дахин ашиглах боломжтой болгодог.

3.2 Програм хангамжийн зохиомжийн үлгэр загварууд: Creational, Structural, Behavioral

Зохиомжийн үлгэр загвар нь програм хангамжийн зохиомжийн нийтлэг асуудлуудад батлагдсан, дахин ашиглах боломжтой шийдэл юм. Уг нэршлийг 1994 онд "Gang of Four" (GoF) алдаршуулж, хөгжүүлэгчдийн нийтлэг үгсийн сангийн нэг болсон. Ихэнх үлгэр загварыг маш формал байдлаар тайлбарласан байдаг бөгөөд тухайн нөхцөл байдалд тохируулан хуулбарлаж ашигладаг. Үлгэр загварын тайлбарт ихэвчлэн байдаг хэсгүүд нь:

- **Интент** нь асуудал болон шийдлийн аль алиныг нь товч тайлбарладаг.
- **Хүсэл эрмэлзэл** нь асуудал болон шийдлийг боломжтой болгохыг цааш нь тайлбарладаг.
- **Классуудын бүтэц** нь үлгэр загварын хэсэг бүр, тэдгээр нь хэрхэн уялдаж байгааг харуулдаг.
- Түгээмэл програмчлалын хэлнүүдийн нэг дэх **кодын жишээ** нь үлгэр загварын цаад санааг ойлгоход хялбар болгодог.

3.2.1 Creational үлгэр загварууд

Creational үлгэр загварууд нь уян хатан байдлыг нэмэгдүүлэх, класс болон модулиудыг дахин ашиглахын тулд объект байгуулах ажлыг зохицуулдаг.

Singleton үлгэр загвар

Класст зөвхөн нэг тохиолдол байгаа эсэхийг баталгаажуулж, түүнд хандах глобал хандалтын цэгийг үүсгэнэ. Практикт энэ класс нь private байгуулагчтай бөгөөд цорын ганц заагчийг

буцаах статик аргатай (эсвэл үүнтэй төстэй) гэсэн үг юм. Энэ үлгэр загвар нь систем дэх үйлдлийг зохицуулахад яг нэг объект шаардлагатай үед хэрэг болно. Гол шинж чанар нь:

- Нэг классын заагч буюу нэг глобал объектыг баталгаажуулдаг.
- Глобал хандалтын цэгээр, жишээ нь getInstance() аргаар хангадаг.
- Ихэвчлэн ”lazy initialization”[1] арга замыг ашигладаг бөгөөд тредийн аюулгүй байдлын механизмуудыг агуулж болно.

Дундын нөөц эсвэл тохиргоог удирдахад энэ үлгэр загвар ихэвчлэн ашиглагддаг. Жишээлбэл, Java-ийн үндсэн ангиуд java.lang.Runtime болон java.awt.Desktop нь синглтон үлгэр загвараар хэрэгжүүлэгдсэн байдаг.

Давуу тал: Нөөц эсвэл үйлчилгээнд глобал хэмжээнд хандах хандалтыг хялбаршуулдаг. Олон клиент ханддаг байсан ч зөвхөн нэг объект үүсдэг. Мөн lazy initialization нь зөвхөн эхний хэрэглээнд л заагч үүсгэх замаар нөөцийг хэмнэх боломжтой.

Сул тал: Глобал төлөв үүсгэх учир кодыг тестлэхэд хэцүү болгож, далд хамааралтай байдалд хүргэж болзошгүй. Нэг классад заагчийн хяналтыг зохицуулах замаар Single Responsibility Principle зарчмыг зөрчиж байна. Болгоомжтой хэрэгжүүлээгүй тохиолдолд конкуррент асуудлууд үүсэж, тредтэй холбоотой нэмэлт код шаарддаг.

Factory Method үлгэр загвар

Объект байгуулах интерфэйс эсвэл хийсвэр аргыг тодорхойлдог боловч дэд классуудад аль конкрет классаар объектыг байгуулахыг шийдэх боломжийг олгодог. Үнэн хэрэгтээ объект байгуулах ажлыг үйлдвэрийн дэд классуудад буюу *үйлдвэрүүдэд* томилон илгээдэг. Энэ нь клиент кодыг конкрет бүтээгдэхүүн классаас салгадаг. Гол шинж чанар нь:

[1] ”lazy initialization” гэдэг нь объект байгуулах эсвэл утгыг тооцолох ажлыг програмыг эхлүүлэх үед биш, харин анх удаа хэрэг болох хүртэл хойшлуулдаг оновчлолын арга зам бөгөөд хэзээ ч ашиглагдахгүй объект байгуулахаас зайлсхийж, програмыг эхлүүлэх хугацааг багасгаж, санах ойн ашиглалтыг бууруулдаг.

- Үйлдвэрийн интерфэйсийн ард объект байгуулах ажлыг битүүмжилдэг.
- Дэд классууд нь өөр өөр бүтээгдэхүүн буцаахын тулд үйлдвэрийн аргыг дарж бичдэг.
- Клиент нь ямар конкрет төрлөөр бүтээгдсэнийг мэдэхгүйгээр үйлдвэрийн интерфейсийг ашигладаг.
- Клиент кодыг өөрчлөхгүйгээр шинэ бүтээгдэхүүн нэмэх боломжийг олгодог.

Класс нь аль дэд классыг байгуулах ёстойг урьдчилан таамаглах боломжгүй үед энэ үлгэр загвар ашиглагддаг. Жишээлбэл, GUI фреймворк нь платформд тусгайлан зориулсан UI элементүүдийг үүсгэхийн тулд үйлдвэрүүдийг ашиглаж болно. Мөн Жава стандарт сан болон фреймворкуудад өргөн хэрэглэгддэг, жишээ нь Spring, Struts.

Давуу тал: Үйлдвэрийн шинэ дэд классуудыг нэмснээр шинэ бүтээгдэхүүн нэвтрүүлэхэд хялбар. Клиент нь конкрет бүтээгдэхүүн классаас бус зөвхөн үйлдвэрийн интерфейсээс хамаардаг. Ингэснээр байгуулах логикийг бизнесийн логикоос тусгаарлаж өгдөг.

Сул тал: Үйлдвэрийн дэд классуудад нэмэлт класс үүсгэх шаардлагатай бөгөөд энэ нь кодын хэмжээг нэмэгдүүлдэг. Зарим тохиолдолд бүтээгдхүүний төрөл бүрийн хувилбаруудыг дэмжихийн тулд олон үйлдвэр хэрэгтэй болдог.

Abstract Factory үлгэр загвар

Конкрет классыг тодорхойлохгүйгээр холбоотой эсвэл хамааралтай объектуудын *гэр бүлийг* үүсгэх интерфейсээр хангадаг. Энэ нь үндсэндээ ”үйлдвэрүүдийн үйлдвэр” юм. Хийсвэр үйлдвэр нь бүтээгдэхүүн тус бүрийг бий болгох аргыг тодорхойлдог бөгөөд конкрет үйлдвэрийн дэд классууд нь конкрет хэрэгжүүлэлтээр хангадаг. Гол шинж чанар нь:

- Хамтдаа ажиллахад зориулагдсан хоорондоо холбоотой объектуудын бүлгүүд буюу бүтээгдэхүүнийг үүсгэдэг.
- Клиентад тодорхой конкрет классаас хамааралгүйгээр бүтээгдэхүүний гэр бүлийг буцаана.

- Конкрет үйлдвэр бүр нь гэр бүлийн бүх бүтээгдэхүүнийг бий болгох аргыг хэрэгжүүлдэг.

Аппликешныг олон төрлийн бүтээгдэхүүний аль нэгэнд тохируулах шаардлагатай үед энэ үлгэр загварыг ашигладаг. Сонгодог жишээ бол олон янзын харагдах байдлыг дэмждэг UI хэрэгсэл юм. AbstractWidgetFactory нь товчлуур, текст хайрцаг, цэс үүсгэж болох ба конкрет үйлдвэрүүд Windows эсвэл Mac загварын хувилбаруудыг үйлдвэрлэдэг.

Давуу тал: Хоорондоо холбоотой бүтээгдэхүүнүүдийг нийцэмжтэй байдлаар хангана. Клиент код нь зөвхөн үйлдвэрийн интерфейстэй ажиллах боломжтой. Мөн үйлдвэрийн дэд классуудыг солилцох замаар өөр өөр бүтээгдэхүүний гэр бүл үүсгэдэг.

Сул тал: Маш олон үйлдвэр, бүтээгдэхүүн классаас хамардаг. Гэр бүлд шинэ бүтээгдэхүүн нэмэхийн тулд хийсвэр интерфейсийг өөрчлөх шаардлагатай нь зохиомжийн Open/Closed зарчмыг зөрчдөг. Конкрет хийсвэр үйлдвэрийн класс нь холбогдох бүтээгдэхүүн бүрийг үйлдвэрлэхийн тулд олон үйлдвэрийн аргын дуудлага ашигладаг.

Builder үлгэр загвар

Нарийн төвөгтэй объект байгуулах үйл явцыг алхам алхмаар тусгаарлах боломжийг олгодог. Энэ нь ижил байгуулах үйл явцыг ашиглан өөр өөр дүрслэлийг бий болгох боломжийг олгодог. Гол шинж чанар нь:

- Нарийн төвөгтэй объектуудыг үе шаттайгаар байгуулдаг, жишээ нь олон нэмэлт хэсгүүд эсвэл үүрлэсэн дэд объектуудтай нь хамт.
- *Барилгачин* класс нь эд ангиудыг цуглуулж, эцсийн объектыг build() аргаар угсардаг.
- Захиалагч нь байгуулагчийн интерфэйсээр дамжуулан объект байгуулах үйл явцыг удирддаг. Барилгачин класс нь харин объект үүсгэх ажлыг зохицуулдаг.

Объект нь олон хэсэг эсвэл хослолын сонголтуудыг шаарддаг бөгөөд энэ нь олон тооны байгуулагчуудыг хэт ачаалахад хүргэх нөхцөлд энэ үлгэр загварыг ашигладаг. Жишээ нь, олон сонголттой хоол бүхий хоолны иж бүрдлийг хийх, эсвэл нэмэлт талбар бүхий хэрэглэгчийг

байгуулах зэрэг орно. Энэ нь өөрчлөгдөшгүй объектууд эсвэл тохиргооны хүнд объектуудыг (жишээ нь, өгөгдлийн сангийн холболтын тохиргоо, нарийн төвөгтэй GUI бүрэлдэхүүн хэсгүүд) бүтээхэд түгээмэл байдаг.

Давуу тал: Хэд хэдэн аргын дуудалтыг гинжлэх замаар програмын уншигдахуйц байдлыг нэмэгдүүлдэг. Байгуулагчын бичдэсийг өөрчлөхгүйгээр нэмэлт параметр эсвэл алхамуудыг хялбархан нэмж болно. Барилгачин объект нь өөрөө түр зуурын, өөрчлөгддөг контейнер бөгөөд эцсийн өөрчлөгддөггүй объектыг бүтээх хүртэл бүх тохиргооны мэдээллийг хадгалдаг.

Сул тал: Үйл явцыг хянахын тулд нэмэлт класс үүсгэх шаардлагатай бөгөөд энэ нь кодын хэмжээг нэмэгдүүлдэг. Бүтээгдэхүүн тус бүрт барилгачин класс хэрэгтэй бөгөөд энэ нь кодын хэмжээг нэмэгдүүлдэг. Энгийн объектуудын хувьд хэт их байж болно.

Prototype үлгэр загвар

Энэ үлгэр загвар нь бүтээгдэхүүний заагчийг ашиглан шинэ объект үүсгэх боломжийг олгодог. Энэ нь шинэ объект байгуулахын оронд одоо байгаа объектыг хуулбарлах замаар шинэ объект үүсгэдэг. Ингэхдээ конкрет *прототипийг* хэрэгжүүлдэг ихэвчлэн нэг аргатай clone() интерфейсийг зарладаг. Клиент нь классуудыг шууд үүсгэхийн оронд прототипийг өөрөө хувилахыг асуух замаар шинэ объектуудыг шаарддаг. Гол шинж чанар нь:

- Объектыг хувилах ажлыг тухайн объектод томилон илгээдэг.
- Кодыг конкрет классуудтай холбохоос зайлсхийдэг учир код нь ганц ерөнхий ”prototype” интерфейстэй харьцдаг.
- Объект байгуулах нь үнэтэй эсвэл төвөгтэй үед ашигтай байдаг, өөрөөр хэлбэл одоо байгаа прототипийг хуулбарлах нь илүү хялбар байх болно.

Ижил төстэй эсвэл эхнээс нь байгуулахад үнэтэй объектуудыг бүтээхэд хэрэгтэй.

Давуу тал: Клиент хувилах объектын классыг мэдэх шаардлагагүй ба одоо байгаа прототипүүдийг хувилах замаар дахин давтагдах кодыг арилгах боломжтой. Мөн програм ажиллаж байх

үед шинэ прототип нэмэхэд хялбар.

Сул тал: Тойрог үүсгэн нэг нэгнээ зааж хандсан эсвэл гадаад нөөцөөс хамаарсан нарийн төвөгтэй объектуудыг хувилах нь төвөгтэй эсвэл алдаатай байж болно. Класс бүрт clone() аргыг хэрэг- жүүлэх ёстой. Гүн ба гүехэн[2] хуулалттай холбоотой асуудлуудыг анхааралтай авч үзэх шаардлагатай.

3.2.2 Structural үлгэр загварууд

Structural үлгэр загварууд нь класс болон объектуудыг уян хатан байлгахын зэрэгцээ том бүтэц болгон хэрхэн бүрдүүлэх талаар авч үздэг.

Adapter үлгэр загвар

Нэг интерфэйсийг өөр интерфэйс рүү хөрвүүлэх боломжийг олгодог. Энэ нь өөр интерфэйсийг шаарддаг клиентүүдэд зориулж хуучин эсвэл таарахгүй интерфэйс бүхий объектуудыг ашиглах боломжийг олгодог. Гол шинж чанар нь:

- Клиент болон үйлчилгээг үзүүлэгчийн хоорондох интерфэйсийн зөрүүг арилгах.
- *Адаптер* класс нь үйлчилгээг үзүүлэгчийн интерфэйсийг хэрэгжүүлдэг бөгөөд дараа нь үйлчилгээг үзүүлэгчийн объект руу дуудлага дамжуулдаг.
- Клиент нь зөвхөн адаптерийн интерфэйстэй харьцдаг.

Хуучин кодыг шинэ систем эсвэл шинэ интерфэйс шаарддаг гуравдагч талын номын сан эсвэл API-тэй нэгтгэхэд ашигладаг. Жишээ нь, Java-ийн InputStreamReader класс нь InputStream (байт урсгал)- ийг Reader (тэмдэгт урсгал) интерфэйс рүү хөрвүүлдэг.

[2]Гүехэн хуулах: Объектын дээд түвшний шинж чанаруудыг хуулах боловч эх хувийн аль ч үүрлэсэн объектыг бус түүний заагчийг оноодог. Энэ нь илүү хурдан боловч хуулбар дахь үүрлэсэн өгөгдөлд өөрчлөлт оруулах нь анхны объектод нөлөөлнө гэсэн үг юм. Гүн хуулах: Шинэ объект үүсгэж, бүх үүрлэсэн объектуудыг давталттайгаар хуулбарлаж, хуулбар нь эх хувилбараас бүрэн хамааралгүй эсэхийг баталгаажуулна. Энэ нь нарийн төвөгтэй, өөрчлөгддөг өгөгдлийн бүтцэд илүү аюулгүй боловч илүү удаан бөгөөд санах ой их шаарддаг.

Давуу тал: Хуучин кодыг шинэ системд дахин ашиглах боломжийг олгодог. Клиент болон үйлчилгээг үзүүлэгчийн хоорондох хамаарлыг бууруулдаг. Мөн олон үйлчилгээг үзүүлэгчийн интерфэйсийг нэг адаптерээр нэгтгэх боломжийг олгодог.

Сул тал: Нэмэлт түвшш буюу төвөгтэй байдлыг нэмж оруулдаг. Зарим тохиолдолд адаптер нь үйлчилгээг үзүүлэгчийн интерфэйсийг бүрэн дэмжихгүй байж болно.

Bridge Pattern

Абстракц болон түүний хэрэгжилтийг тусгаарлах боломжийг олгодог. Гол шинж чанар нь:

- Абстракц болон түүний хэрэгжилтийг тусгаарлах.
- Абстракц нь хэрэгжилтийн интерфэйсийг агуулдаг бөгөөд дараа нь хэрэгжилтийн объект руу дуудлага дамжуулдаг.
- Клиент нь зөвхөн абстракцын интерфэйстэй харьцдаг.

Абстракц болон түүний хэрэгжилт нь өөр өөр шалтгаанаар өөрчлөгдөж болзошгүй үед энэ үлгэр загварыг ашигладаг. Жишээ нь, график хэрэглэгчийн интерфэйсийн элементүүдийг (жишээ нь, товчлуур, текст хайрцаг) платформ тус бүрийн хэрэгжилтээс тусгаарлаж болно.

Давуу тал: Абстракц болон түүний хэрэгжилтийг тусгаарлах боломжийг олгодог. Абстракц болон хэрэгжилтийн аль алиныг нь бие даан өөрчлөх боломжийг олгодог. Мөн олон хэрэгжилтийн хувилбаруудыг дэмжихэд хялбар.

Сул тал: Нэмэлт түвшш буюу төвөгтэй байдлыг нэмж оруулдаг. Зарим тохиолдолд абстракц нь хэрэгжилтийн интерфэйсийг бүрэн дэмжихгүй байж болно.

Composite үлгэр загвар

Объектуудыг модны бүтэц болгон зохион байгуулж, нэгж болон бүрэлдэхүүн хэсгүүдийг ижил байдлаар харьцах боломжийг олгодог. Гол шинж чанар нь:

- Композит болон навчны объектууд ижил интерфэйсийг хэрэгжүүлдэг.
- Композит объект нь хүүхэд объектуудын цуглуулгыг агуулж, хүүхэд объектууд дээр үйлдлүүдийг гинжин хэлхээнээр гүйцэтгэдэг.
- Клиент нь композит болон навчны объектуудыг ижил байдлаар харьцдаг.

Ижил төрлийн объектуудын ижил үйлдлийг нэгтгэх шаардлагатай үед энэ үлгэр загварыг ашигладаг. Жишээ нь, график хэрэглэгчийн интерфэйсийн элементүүдийг (жишээ нь, цэс, товчлуур, текст хайрцаг) модны бүтэц болгон зохион байгуулж, хэрэглэгчийн интерфэйсийг ижил байдлаар удирдах боломжийг олгодог.

Давуу тал: Композит болон навчны объектуудыг ижил байдлаар харьцах боломжийг олгодог. Композит бүтэц нь динамикаар өөрчлөгдөж, шинэ хүүхэд объектуудыг нэмэх эсвэл устгах боломжтой. Мөн модны бүтцийн бүх түвшинд үйлдлүүдийг гинжин хэлхээнээр гүйцэтгэх боломжийг олгодог.

Сул тал: Композит бүтэц нь төвөгтэй бөгөөд удирдах, ойлгоход хэцүү байж болно. Зарим тохиолдолд навчны объектуудыг композит объектуудаас ялгах нь төвөгтэй байж болно.

Decorator үлгэр загвар

Объектын үйлдлийг динамикаар нэмэх эсвэл өөрчлөх боломжийг олгодог. Гол шинж чанар нь:

- Үйлдлийг нэмэх эсвэл өөрчлөхийн тулд объектын оронд түүний оронд *декоратор* объект ашигладаг.
- Декоратор класс нь анхны объектын интерфэйсийг хэрэгжүүлдэг бөгөөд дараа нь анхны объект руу дуудлага дамжуулдаг.
- Клиент нь зөвхөн декораторын интерфэйстэй харьцдаг.

Хувьсах эсвэл нэмэлт функцтэй объектуудыг динамикаар бий болгох шаардлагатай үед энэ үлгэр загварыг ашигладаг. Жишээ нь, Java-ийн I/O номын сан нь InputStream болон Reader зэрэг анхны объектуудыг янз бүрийн декораторууд (жишээ нь, BufferedInputStream, DataInputStream) ашиглан нэмэлт үйлдлүүдээр (жишээ нь, буферлалт, өгөгдлийн уншилт) чимэглэж болно.

Давуу тал: Объектын үйлдлийг динамикаар нэмэх эсвэл өөрчлөх боломжийг олгодог. Анхны класс эсвэл модульыг өөрчлөхгүйгээр шинэ функц нэмэх боломжийг олгодог. Мөн олон декораторуудыг гинжин хэлхээнээр холбож, олон төрлийн функцтэй объектуудыг бий болгох боломжийг олгодог.

Сул тал: Нэмэлт түвшш буюу төвөгтэй байдлыг нэмж оруулдаг. Зарим тохиолдолд декоратор нь анхны объектын интерфэйсийг бүрэн дэмжихгүй байж болно.

Facade үлгэр загвар

Нэг буюу хэд хэдэн нарийн төвөгтэй систем, номын сангийн энгийн интерфэйсийг хангадаг. Гол шинж чанар нь:

- Нарийн төвөгтэй систем, номын сангийн ард энгийн интерфэйсийг хангадаг.
- Фасад класс нь нарийн төвөгтэй системийн олон үйлдлийг нэгтгэж, энгийн аргаар дамжуулдаг.
- Клиент нь зөвхөн фасадын интерфэйстэй харьцдаг.

Нарийн төвөгтэй систем, номын санг хялбархан ашиглах шаардлагатай үед энэ үлгэр загварыг ашигладаг. Жишээ нь, Java-ийн JDBC API нь өгөгдлийн сантай харьцахын тулд фасадын интерфэйсийг хангадаг.

Давуу тал: Нарийн төвөгтэй систем, номын санг хялбархан ашиглах боломжийг олгодог . Клиент болон нарийн төвөгтэй системийн хоорондох хамаарлыг бууруулдаг. Мөн нарийн төвөгтэй системийн олон үйлдлийг нэгтгэж, энгийн аргаар дамжуулдаг.

Сул тал: Фасад нь нарийн төвөгтэй системийн бүх функцэд хандах боломжийг хангахгүй

байж болно. Зарим тохиолдолд фасад нь нарийн төвөгтэй системийн интерфэйсийг бүрэн дэмжихгүй байж болно.

Flyweight үлгэр загвар

Их хэмжээний объектуудыг үр ашигтайгаар удирдах боломжийг олгодог. Гол шинж чанар нь:

- Их хэмжээний объектуудыг хуваалцах замаар санах ойг хэмнэх.
- *Flyweight* класс нь хуваалцсан өгөгдөл (жишээ нь, дүрслэл, төлөв) болон хуваалцдаггүй өгөгдөл (жишээ нь, байрлал, төлөв)- ийг агуулдаг.
- Клиент нь зөвхөн flyweight-ийн интерфэйстэй харьцдаг.

Их хэмжээний ижил төстэй объектуудыг удирдах шаардлагатай үед энэ үлгэр загварыг ашигладаг. Жишээ нь, текст засварлагч нь ижил үсэг дүрсүүдийг хуваалцаж, санах ойг хэмнэхийн тулд flyweight үлгэр загварыг ашиглаж болно.

Давуу тал: Их хэмжээний объектуудыг үр ашигтайгаар удирдах боломжийг олгодог. Санах ойг хэмнэхийн тулд ижил төстэй объектуудыг хуваалцдаг. Мөн flyweight объектуудыг дахин ашиглах боломжийг олгодог.

Сул тал: Flyweight объектуудыг зохицуулахын тулд нэмэлт төвөгтэй байдлыг нэмж оруулдаг. Зарим тохиолдолд flyweight нь бүх шаардлагатай өгөгдлийг агуулж чадахгүй байж болно.

Proxy үлгэр загвар

Бусад объект руу хандах хяналтын цэгийг хангадаг. Гол шинж чанар нь:

- Үйлчилгээг үзүүлэгчийн объект руу хандах хяналтын цэгийг хангадаг.
- Прокси класс нь үйлчилгээг үзүүлэгчийн интерфэйсийг хэрэгжүүлдэг бөгөөд дараа нь үйлчилгээг үзүүлэгчийн объект руу дуудлага дамжуулдаг.

- Клиент нь зөвхөн проксийн интерфэйстэй харьцдаг.

Үйлчилгээг үзүүлэгчийн объект руу хандах хяналтыг хэрэгтэй үед энэ үлгэр загварыг ашигладаг. Жишээ нь, Java-ийн RMI (Remote Method Invocation) нь алсын объект руу хандах прокси объектуудыг үүсгэдэг.

Давуу тал: Үйлчилгээг үзүүлэгчийн объект руу хандах хяналтыг хангадаг. Үйлчилгээг үзүүлэгчийн объектын амьдралын мөчлөгийг удирдах боломжийг олгодог. Мөн үйлчилгээг үзүүлэгчийн объект руу хандах өмнөх боловсруулалт (жишээ нь, кэшлэх, аюулгүй байдал)-ийг гүйцэтгэх боломжийг олгодог.

Сул тал: Нэмэлт түвшш буюу төвөгтэй байдлыг нэмж оруулдаг. Зарим тохиолдолд прокси нь үйлчилгээг үзүүлэгчийн интерфэйсийг бүрэн дэмжихгүй байж болно.

3.2.3 Behavioral үлгэр загварууд

Behavioral үлгэр загварууд нь объект хоорондын холбоос, үүргийг тодорхойлох, хяналтын урсгал болон алгоритмыг удирдах талаар тусгасан байдаг.

Observer үлгэр загвар

Нэг объектын төлөв өөрчлөгдөхөд хамааралтай объектуудад автоматаар мэдэгдэх боломжийг олгодог. Гол шинж чанар нь:

- Нэг объект (subject) нь өөрт хамааралтай олон объектууд (observers)- ыг удирддаг.
- Subject объект нь төлөвийн өөрчлөлтийг observers объектуудад мэдэгддэг.
- Observers объектууд нь subject объектын төлөвийн өөрчлөлтөд хариу үйлдэл үзүүлдэг.

Үйл явдлын хяналт, үйл явдлын хариу үйлдэл, эсвэл тараах системд энэ үлгэр загварыг ашигладаг. Жишээ нь, Java-ийн Swing фреймворк нь товчлуур дээр дарах зэрэг үйл явдлуудыг хянахын тулд observer үлгэр загварыг ашигла даг.

Давуу тал: Объект хоорондын сул хамаарлыг хангадаг. Subject болон observers объектуудыг

бие даан өөрчлөх боломжийг олгодог. Мөн олон observers объектуудыг нэг subject объекттой холбох боломжийг олгодог.

Сул тал: Observers объектуудын тоо их байх үед гүйцэтгэлд нөлөөлж болзошгүй.

Strategy үлгэр загвар

Нэг алгоритмыг өөр алгоритмтай солих боломжийг олгодог. Гол шинж чанар нь:

- Алгоритмыг тусгаарлахын тулд хийсвэр *стратеги* интерфэйсийг тодорхойлдог.
- Тус интерфэйсийг хэрэгжүүлдэг олон *конкрет стратеги* классууд байдаг.
- Клиент нь стратеги интерфэйсээр дамжуулан алгоритмыг сонгодог.

Алгоритмыг сонгох эсвэл өөрчлөх шаардлагатай үед энэ үлгэр загварыг ашигладаг. Жишээ нь, Java-ийн Collections.sort() аргыг нь янз бүрийн эрэмбэлэх стратегиудыг (жишээ нь, өсөх, буурах) дэмжихийн тулд стратеги үлгэр загварыг ашигладаг.

Давуу тал: Алгоритмыг бие даан өөрчлөх боломжийг олгодог. Клиент нь зөвхөн стратеги интерфэйстэй харьцдаг. Мөн шинэ стратеги классуудыг хялбархан нэмж болно.

Сул тал: Нэмэлт төвөгтэй байдлыг нэмж оруулдаг. Зарим тохиолдолд стратеги нь бүх шаардлагатай өгөгдлийг агуулж чадахгүй байж болно.

Command үлгэр загвар

Үйлдлийг объект болгон төлөөлөх боломжийг олгодог. Гол шинж чанар нь:

- Үйлдлийг гүйцэтгэхийн тулд *команд* объект ашигладаг.
- Команд класс нь үйлдлийг гүй цэтгэхийн тулд *гүйлгэх* аргыг хэрэгжүүлдэг.
- Клиент нь зөвхөн коммандын интерфэйстэй харьцдаг.

Командуудыг хадгалах, дараах байдлаар гүйцэтгэх, эсвэл буцаах шаардлагатай үед энэ үлгэр загварыг ашигладаг. Жишээ нь, Java-ийн Swing фреймворк нь товчлуур дээр дарах зэрэг

үйлдлүүдийг гүйцэтгэхийн тулд комманд үлгэр загварыг ашигладаг.

Давуу тал: Үйлдлийг объект болгон төлөөлөх боломжийг олгодог. Коммандуудыг хадгалах, дараах байдлаар гүйцэтгэх, эсвэл буцаах боломжийг олгодог. Мөн шинэ комманд классуудыг хялбархан нэмж болно.

Сул тал: Нэмэлт төвөгтэй байдлыг нэмж о руулдаг. Зарим тохиолдолд комманд нь бүх шаардлагатай өгөгдлийг агуулж чадахгүй байж болно.

Iterator үлгэр загвар

3.3 ...

...

4. ДАДЛАГЫН БЭЛТГЭЛ АЖИЛ

4.1 Урвуу инженерчлэл: ”Auftragsverwaltung” систем дэх зохиомжийн үлгэр загварууд

Энэ хэсэгт Жава технологийг ашиглан хэрэгжүүлсэн бараа захиалгийг зохицуулах Герман нэршлийн конвенцтэй ”Auftragsverwaltung” систем дээр урвуу инженерчлэлийн аргачлалаар програм хангамжийн зохиомжийн үлгэр загваруудыг уг системд хэрхэн ашигласан байгааг олж тогтооно.

4.1.1 Системийн статик загвар

Системийн классын диаграмыг гаргахын тулд Eclipse IDE дээрх PlantUML[1] програм хангамжийн багажыг ашигласан. Гэвч энэ багаж нь классууд хоорондын холбоосуудыг, тухайлбал бүрдмэл, нийлмэл зэрэг холбоог оновчтой гаргаж чадаагүй. Юуны түрүүнд энэ системийн нэршлийн конвенц нь Герман хэл дээр хийгдсэн учир тодорхой үгсийн Герман–Англи нэршлийн конвенцийг гаргах шаардлага үүссэн. Жишээ нь, ”Auftragsverwaltung” нь ”OrderManagement” буюу захиалгын удирдлага, ”Auftrag” нь Order” буюу захиалга, ”Kunde” нь ”Customer” буюу харилцагч, ”auftragLoeschen” нь ”deleteOrder” буюу Order (Aufrag) классын устгагч гэх мэт. Хавсралт В хэсгээс эдгээр орчуулгыг харж болно.

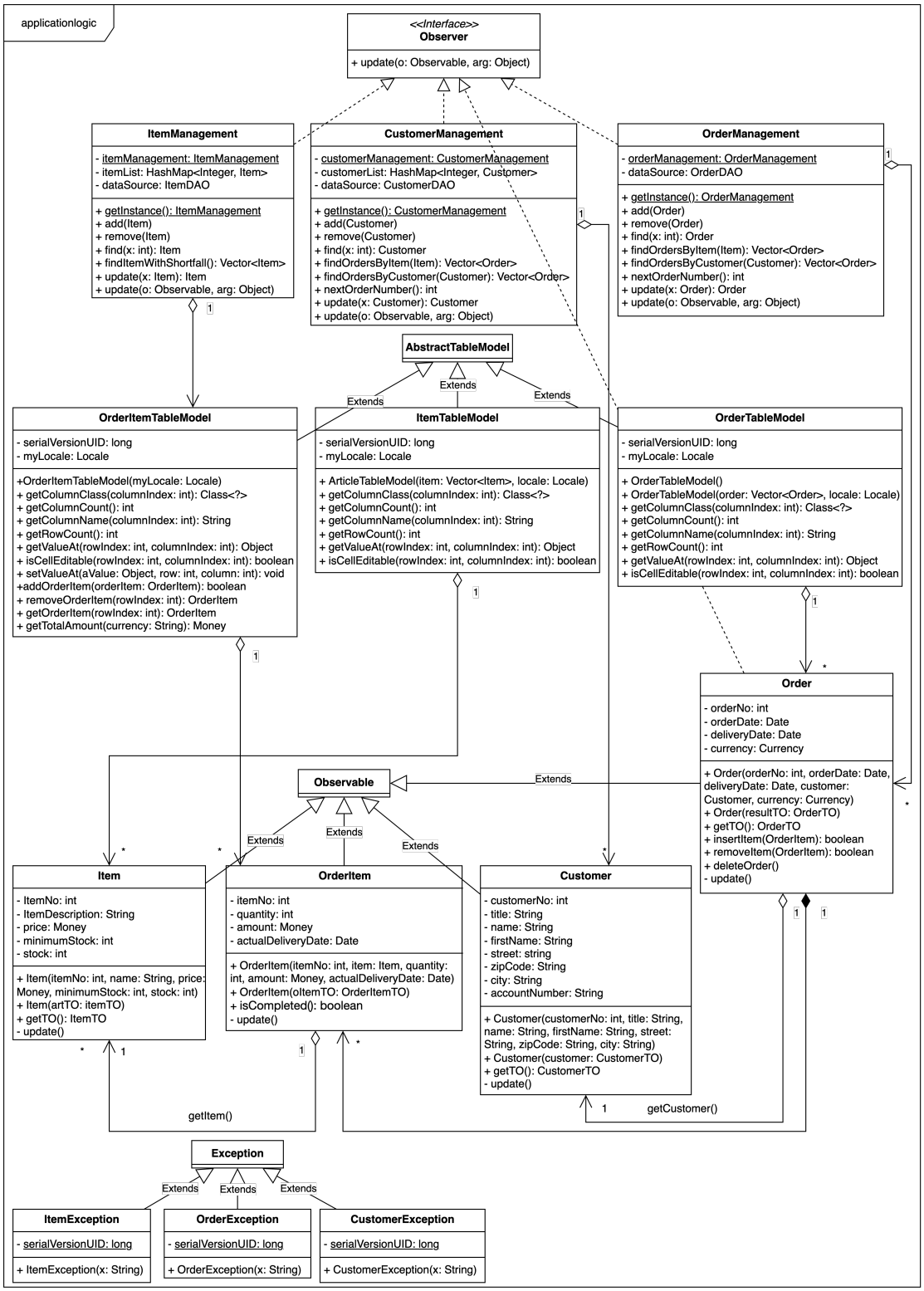
Холбоосуудын төрлийг тодорхойлохын тулд класс тус бүрийн устгагчийн хэрэгжүүлэлтийг ажигласан. Жишээ нь, Order (Aufrag), OrderItem (Aufragsposition) классуудын хувьд Order классын объектийг устгах үед OrderItem классын объектуудаас бүрдсэн векторoop entfernenPos аргын тусламжтай гүйж устгаж байна. Харин Customer (Kunde) классын объектыг устгалгүй үлдээсэн байна. Иймд Order, OrderItem классууд хоорондоо нийлмэл холбоотой бол Order, Customer классууд хоорондоо бүрдмэл холбоотой гэж дүгнэлээ.

[1] https://plantuml.com/eclipse

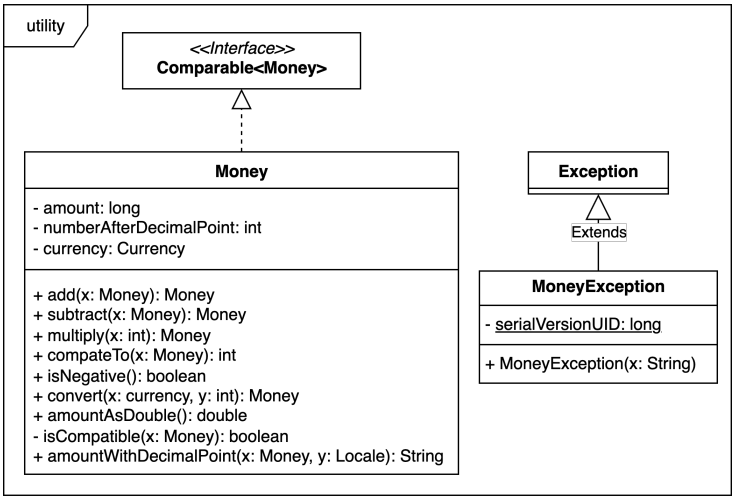
```
public boolean entfernenPos(Auftragsposition apos)
{
    boolean rc = apositionen.remove(apos);
    if (rc)
        this.aktualisieren();
    return rc;
}

public void auftragLoeschen()
{
    Iterator<Auftragsposition> positionen =
            apositionen.iterator();
    while (positionen.hasNext())
    {
        entfernenPos(positionen.next());
    }
}
```

Код 4.1: Order классын устгагч auftragLoeschen

Үүнтэй адилаар ”Applicationlogic” багцын бүх классын нэрсийг жагсааж, тэдгээрийн хоорон -дын холбоосуудыг илрүүлсэн. Дараа нь, холбоосуудын төрлийг тодорхойлж, UML классын диаграмыг гаргасан. Үүний үр дүнг 4.1 зурагт үзүүлэв. Мөн Item (Artikel) класс нь гадаад ”Utility” багцаас (зураг 4.2) Money (Geld) классыг ашиглаж байгаа тул энэ холбоосыг харуу -лаагүй болно.



Зураг 4.1: ”Applicationlogic” багцын классын диаграм



Зураг 4.2: ”Utility” багцын классын диаграм

4.1.2 Системийн зохиомж

Энэ хэсэгт ”Auftragsverwaltung” системд илрүүлсэн гол зохиомжийн үлгэр загваруудын зорилго, хэрэглээ, бүтцийн элемент, өмнө тулгарч болох асуудал, үр дүн, жишээ код болон систем доторх бодит хэрэгжүүлэлтийг дэлгэрэнгүй авч үзнэ.

”Iterator” үлгэр загвар

Order классын бүх талбар, аргуудаар статик уншлага хийн `apositionen.iterator()`, `while (positionen.hasNext())`, `positionen.next()` мөрүүдийг тэмдэглэж үлгэр загварт заавал байх элементүүд кодод ямар хэлбэрээр илэрсэнг харлаа. Энд apositionen нь *бүрдэл* болж, түүний `iterator()` аргыг дуудаж байгаа нь ”Iterator” интерфэйсийг ашиглаж буйг илтгэнэ. `auftragLoeschen()` болон `getAuftragssumme()` мэт аргуудын давталтын логикыг уншиж, хэрхэн давталт явдаг, давталтын үед ямар үйлдэл хийгдэхийг тодруулсан. Давталтын туршид бүрдлүүдтэй хэрхэн харьцаж байгааг анхааран харж, ”concurrent modification”[2] зэрэг

[2]Конкуррент програмчлалд өгөгдлийн бүтцийг өөр процесс эсвэл тредээр давтаж байх үед өөрчилсөн тохиолдолд ”concurrent modification” үүсдэг. Энэ нь үнэн байхаа больсон өгөгдлийн бүтцээр гүйснээс үүдсэн урьдчилан таамаглах боломжгүй өгөгдлийн эвдрэл эсвэл ”runtime error” алдааг үүсгэж болно.

асуудал үүсэх эрсдлийг тооцсон. Дээрх ажиглалтаас үлгэр загварын шинж тэмдгүүд илэрсэн тул тухайн кодонд ”Iterator” үлгэр загвар ашиглагдсан гэж тодорхойлсон. Энд apositionen нь `Vector<Auftragsposition>` бөгөөд давталт хийхдээ `apositionen.iterator()` ашиглагдаж байна. `getAuftragssumme()` нь `while(positionen.hasNext()){...positionen.next()...}` маягаар стандарт давталтыг ашиглан бүх Amount (Betrag) буюу үнийн дүнг олж байна. Энэ нь ”Iterator” интерфэйсийн классик хэрэглээ юм.

```
1  public Geld getAuftragssumme()
2  {
3    Iterator<Auftragsposition> positionen =
4        apositionen.iterator();
5    Geld ergebnis = new Geld(0, 0, waehrung);
6    if (positionen.hasNext())
7      ergebnis = new Geld(positionen.next().getBetrag());
8    while (positionen.hasNext())
9    {
10     ergebnis.addieren(positionen.next().getBetrag());
11   }
12   return ergebnis;
13 }
```

Код 4.2: Order классын арга getAuftragssumme

”Singleton” үлгэр загвар

OrderManagement классын статик талбар дээр шууд `new Auftragsverwaltung()` дуудаж байгаа нь ”eager initialization”[3] бөгөөд тредүүдэд аюулгүй байдаг боловч хэзээ ч ашиглагдахгүй объект байгуулагдах эрсдэлтэй. Мөн `DAOFactory.getInstance()` гэх мэт глобал хандалтын

[3]”Eager initialization” нь програмчлалын стратеги бөгөөд объект эсвэл нөөцийг анх ашиглахыг хүсэх хүртэл хүлээх биш, агуулагдсан класс нь ачаалагдсан даруйд үүсгэгддэг.

цэгүүдийг ажиглалаа. Класс нь өөрөө ”Observer”-ыг хэрэгжүүлж, менежментийн үүрэг гүйцэтгэж байгаа ч байгуулагчийг private гэж огт заагаагүй тул шинэ Auftragsverwaltung объектыг байгуулж параллел объект үүсгэх эрсдэлтэй. `hinzufuegen()` дотор `auf.addObserver(this)` гэж байгаа нь Singleton объект өөрөө Observable объектын өөрчлөлтийг хүлээн авч DAO-руу өөрчлөлт илгээж байна. Энэ бүх ажиглалтуудаас жинхэнэ ”Singleton” үлгэр загварыг ашиглаагүй гэж дүгнэлээ. Харин өөр нэг үлгэр загварыг олсон нь ”Observer” юм. Үүнийг дараагийн хэсэгт тайлбарлав.

```
1  private static Auftragsverwaltung eineAuftragsverwaltung = new
       Auftragsverwaltung();
2
3  public void hinzufuegen(Auftrag auf)
4      throws AuftragException
5  {
6    int auftragsnr = auf.getAuftragsnr();
7    try {
8      if (datenquelle.read(auftragsnr) != null)
9        throw new AuftragException(
10           "Auftrag␣mit␣dieser␣Nummer␣ist␣schon␣vorhanden");
11     datenquelle.create(auf.getTO());
12     auf.addObserver(this);
13   } catch (Exception ex)
14   {
15     throw new AuftragException("Auftrag␣" + auftragsnr
16         + "␣konnte␣nicht␣gespeichert␣werden!");
17   }
18 }
```

Код 4.3: OrderManagement классын арга hinzufuegen

”Observer” үлгэр загвар

OrderManagement классын `hinzufuegen()` арга дотор `auf.addObserver(this)` гэж байгаа нь Singleton объект өөрөө Observable объектын өөрчлөлтийг хүлээн авч DAO-руу өөрчлөлт илгээж байна. Үүнийг ажигласнаар ”Observer” үлгэр загвар ашиглагдсан гэж дүгнэлээ. Домай объект болох Customer, Order, OrderItem, Item зэрэг классууд нь Observable-ээс удамшиж, өөрийн төлөв өөрчлөгдөх мөчид `notifyObservers()`-ыг дуудаж бүх бүртгэлтэй Observer-уудад мэдээлж байна. CustomerManagement, OrderManagement, ItemManagement зэрэг класс нь Observer интерфейсийг хэрэгжүүлэн, `update()` аргаар домайн объектоос ирсэн өөрчлөлтийг хүлээн авч байна. Ингэснээр системийн бүх чухал домэйн объектууд үзэгдлийн төв болж, менежер бүр эдгээр үзэгдлийг сонсож, өөрийн цуглуулгад харгалзах домэйн объектыг дахин оруулах/шинэчлэх нь төвлөрсөн удирдлага үүсгэхээс гадна домэйн объект, менежерүү- дийн хооронд нягт уялдаа холбоог бий болгож, нийцэмжтэй байдлыг хадгалж байна.

DAO (Data Access Object) үлгэр загвар

Intent: Бизнес логик болон өгөгдлийн сан (database) хооронд харилцааг салгаж, өгөгдлийн хандалтын кодыг төвлөрүүлдэг. Ингэснээр өгөгдлийн сангийн төрөл, схем өөрчлөгдөхөд бусад давхаргаас хамааралгүйгээр шинэчлэлт хийх боломжтой. Applicability: Өгөгдлийн сан, файлын систем, REST үйлчилгээ болон бусад I/O үйлдлүүдтэй тухайн давхаргын кодыг багцлах шаардлагатай үед. Олон бизнес обьект өгөгдлийн үйлдэл хийх үед давталт, дубликацыг багасгах шаардлагатай бол. Participants: DAO Interface: SqlDAO<T> — нийтлэг өгөгдөл унших, бичих аргатай. Concrete DAO: ArtikelDAO, KundeDAO, AuftragDAO — хүссэн хүснэгтийн өргөтгөл. Model: Artikel, Kunde, Auftrag классууд. Collaboration: Business логик DAO interface-ээр дамжин өгөгдөл авах/хадгална. Concrete DAO-ууд өгөгдлийн сангийн SQL query-үүдийг гүйцэтгэнэ. Consequences: Давхаргын тусгаарлалт сайжирна. Тестлэхэд mock DAO ашиглах замаар өгөгдлийн сангүйгээр unit тест бичиж болно. Өгөгдлийн сангийн холболт өөрчлөгдөхөд DAO давхаргыг л шинэчлэхэд хангалттай. Жишээ код:

5. СИСТЕМИЙН ШИНЖИЛГЭЭ

6. СИСТЕМИЙН ЗОХИОМЖ

7. АШИГЛАСАН ТЕХНОЛОГИ

7.1 Git

Linus Trovalds буюу Linux Kernel-г хөгжүүлсэн хүн Kernel-ийнхээ эх кодыг удирдах зорилгоор уг технологийг анх санаачилж, хэрэгжүүлсэн байдаг. Гол зорилго нь Version Control System буюу хувилбар удирдах системийг бүтээх ба ингэснээр хөгжүүлэлтийн явцад бүх өөрчлөлтөө хадгалах, багаар ажиллах боломжийг бүрдүүлж өгсөн юм. Дадлага хийх хугацаандаа байгууллагынхаа сонгосон Gitlab.com платформыг ашиглаж, GIT-н үндсэн үйл ажиллагааны талаар илүү сайн ойлголттой боллоо.

Үндсэн ажиллагааг нь тайлбарлавал GIT ашиглаж буй хөгжүүлэгч/хэрэглэгч бүр өөрийн төхөөрөмж дээр үндсэн repository-тай яг ижил repository-г үүсгэнэ. Ингэснээр сүлжээнд холбогдсон эсэхээс үл хамааран дурын өөрчлөлт, хөгжүүлэлтээ хийх боломжтой ба уг ажил дууссан үед бичсэн өөрчлөлтүүдээ commit хийж, remote repository руу push үйлдлийг хийнэ.

7.2 React library

Фэйсбүүк компани дотооддоо ашиглаж байсан технологио 2013 онд танилцуулсан нь програмчлалын Javascript хэлийг ашиглаж хийсэн Front-end library болох React[1] технологи юм. Declarative UI хөгжүүлэлтийн аргыг хамгийн анх дэлгэрүүлж, өргөн хэрэглээнд нэвтрүүлж чадсан тул Declarative UI-н гол төлөөлөгч гэж явдаг. Уг технологийг ашиглахын тулд үндсэн хэдэн ойлголтууд авах хэрэгтэй. Үүнд component ба түүний lifecycle, javascript-н өргөжүүлсэн хувилбар болох jsx, мөн хамгийн чухал зүйл болох Virtual DOM нар багтана.

Declarative UI гэдэг нь хэрэглэгчийн интерфэйсийн кодыг бичихдээ юу зурагдах буюу render хийх үеийн интерфэйсийг бүгдийг урьдчилан тодорхойлдог. Imperative програмчлалаас ялгаатай нь хязгаартай нөхцөлд яг юу хийхийг хатуугаар зааж өгөхгүйгээр тухайн state-с

[1] Reactjs official site `https://reactjs.org`

хамааруулж хэрэглэгчийн хүссэн зүйлийг гаргаж өгөх боломжтой.

React нь component-based буюу DOM дээр хэвлэж байгаа бүх зүйлс component байна гэсэн дүрмийг баримталдаг. Component үүсгэж бичихийн давуу тал нь нэг бичсэн кодоо олон дахин бичигдэхээс зайлсхийж, дахин ашиглах боломжийг олгодог. Тус бур өөрсдийн гэсэн дотоод төлөвтэй мөн гаднаас утга хүлээн авах чадвартай. Үүнийг бид Props гэж нэрлэдэг. Мөн component нь stateless, stateful гэж хоёр хуваагддаг ба stateful component нь өөрийн гэсэн төлөвтэй, түүнийгээ удирддаг, class болон hook ашигласан функцууд байна. React-н давуу тал нь state эсвэл props-н өөрчлөлтийг үргэлж хянаж байдаг тул өөрчлөлт орж ирэхэд бүтэн хуудсыг зурах бус зөвхөн тухайн өөрчлөгдсөн component-г л дахин зурдаг. Ингэснээр энгийн вэбүүдээс илүү хурдтай ажилладаг.

JSX нь Javascript Extended гэсэн үгний товчлол бөгөөд энгийнээр javascript дотор HTML-н тагуудыг бичиж өгөх мөн кодыг илүү богино болгож хүссэн үр дүндээ хүрэх боломжийг олгодог. Үүний цаана Babel гэсэн transcompiler-г ашиглаж дундын хөрвүүлэлтийг хийдэг ба хэдийгээр HTML таг бичиж байгаа харагддаг ч код дунд цэвэр HTML-г огтоос бичиж өгдөггүй гэсэн үг юм.

```
export function Container = ({children}) => {
  return (
    <div className="container">
      {children}
    </div>
  )
}
```

Код 7.1: JSX ашиглаж ”container” класстай html элемент буцаах компонент

Жинхэнэ DOM дээр богино хугацаанд олон өөрчлөлт хийхэд удах асуудал гарсан тул React маань Virtual DOM гэсэн abstraction давхарга үүсгэж өөрчлөлтүүдээ Virtual DOM дээрээ хадгалаад нэгдсэн нэг өөрчлөлтийг жинхэнэ DOM руугаа дамжуулдаг.

7.3 Next.js

Next.js[2] нь React library дээр суурилж хөгжүүлсэн нээлттэй эхийн фрэймворк бөгөөд Vercel компани 2016 онд албан ёсны танилцуулгаа хийж олон нийтэд зарласан юм. React нь зөвхөн хэрэглэгчийн интерфэйсийг зурах үүрэгтэй сан ба бусад вэб хөгжүүлэлтэд хэрэгтэй хуудас хооронд шилжих гэх мэт үйлдлийг react-router болон бусад маш олон нэмэлт сангаас сонголт хийж шийдэх шаардлагатай байсан нь төслийн эхлэх явцыг удаашруулах хандлагатай байдаг. Харин Next.js ашигласнаар нэг ч тохиргоо хийлгүйгээр төслийг эхлүүлж шууд код бичих боломжийг бүрдүүлдэг. Цаана нь хийгдсэн тохиргоо нь нийт вэбсайтуудын 90 хувийн шаардлагыг хангаж чаддаг гэж үздэг нь уг фрэймворкын сүүлийн жилүүдэд эрэлттэй болж буй шалтгаануудыг нэг билээ. Дадлага хийсэн компани маань төслүүдээ Next.js дээр хийдэг тул уг технологийг зайлшгүй сурах шаардлага гарсан юм.

Next.js давуу талуудаас дурьдвал:

- Image Optimization буюу их хэмжээтэй зураг оруулахад автоматаар зургийн чанарыг алдагдалгүйгээр хэмжээг багасгаж өгдөг
- Zero config буюу нэг ч тохиргоо хийлгүйгээр төслөө эхлүүлэх боломж
- Static Site Generator болон Server Side Render хийх
- Typescript болон Fast Refresh дэмждэг
- File-system Routing буюу “pages” гэсэн хавтас дотор үүссэн файлуудаас хамаарч вэбийн замууд тодорхойлогддог мөн dynamic routing ашиглах боломжтой
- API Routes буюу өөр дээрээ nodejs сервер ашиглаж API endpoint гаргах. Ингэснээр тусдаа сервер ашиглах шаардлага үүсэхгүй
- SEO буюу хайлтын системийн оновчлолыг SSR ашиглаж тохируулж өгөх гэх мэт маш олон давуу талуудтай

[2]Next.js official site `https://nextjs.org`

Мөн ердөө ганц “build” коммандаар статик болон динамик вэбийг гарган авч ямар нэгэн вэб сервер /apache, nginx гэх мэт/ ашиглалгүйгээр сервер дээрээ шууд байршуулах боломжтой юм.

7.4 Material-ui сан

Material-ui нь 2014 онд Google-н дизайнер Матиас Дуартегийн санаачилсан Material Design гэх design language дээр суурилж хийгдсэн нээлттэй эхийн төсөл юм. Ашигласан технологи нь React тул бэлэн component-уудыг дуудаж төсөл дээрээ шууд ашиглах боломжтой. Уг санг дадлагын хугацаан дахь сүүлийн долоо хоног дээрээ ажилласан төсөл дээр хэрэглэсэн ба дизайн дээр нэмэлт хөгжүүлэлт хийлгүй хариуцаж авсан component-ийнхоо зөвхөн логик үйлдлүүд дээр анхаарах боломжийг олгосноор хөгжүүлэлтийн процессийг маш ихээр хурдасгаж өгсөн.

Уг сан нь css-in-js технологи болох JSS-г ашигладаг. Хэрэв интерфэйс загвар дээр нэмэлт хөгжүүлэлт хийх шаардлага гарвал makeStyle() гэсэн custom hook ашиглаж CSS кодоо бичнэ. Сүүлийн beta хувилбар дээрээ JSS-г хасаж dynamic style дээрээ emotion болон styled-component-г хэрэглэж эхэлсэн байгаа. Жишээ болгон нэг компонентийн загварыг сольж үзүүллээ.

```
import {Grid, Card} from '@material-ui/core'

function Card({ children }) {
  return (
    <Grid item xs={12} md={4}>
      <Card header="  ␣ ?">
       {children}
      </Card>
    </Grid>
  );
}
```

Код 7.2: Material-ui сангийн компонентыг ашиглаж буй байдал

8. ХЭРЭГЖҮҮЛЭЛТ

8.1 Аймаг сумдын мэдээлэл авдаг форм

Уг формыг шаардлагын дагуу алхам алхмаар хөгжүүлснээр React-н анхан шатны туршлагатай болох зорилготой байсан ба цаашид ажилласан төсөл дээр хэрэглэж буй зарим сан болох Material-ui, react-select-н ажиллагааг ойлгох, тодорхой хэмжээнд практик мэдлэгийг цуглуулж чадсан.

Сурах ур чадвар:

- Асуудлаа тодорхойлж бага багаар шийдвэрлэх чадварт суралцах
- Git ашиглах чадвараа нэмэгдүүлэх
- Нэг төсөл дээр өөр өөр технологи ашиглах

Формын шаардлага:

- Next.js ашигласан байна
- Functional component ашиглаж, тухайн component-н дотоод төлвийг ашиглах
- Эцэг сонголтыг өөрчлөхөд хүү сонголтуудын утга цэвэрлэгддэг байх
- useEffect hook ашиглах
- Дараагийн шатанд форм дээрээ Material-ui нэвтрүүлэх
- Дараагийн шатанд useReducer ашиглах
- Дараагийн шатанд react-select санг нэвтрүүлэх

8.1.1 useEffect болон useState ашиглан формын мэдээллийг шаардлагын дагуу авах

Эхний ээлжинд ямар нэгэн загваргүй зөвхөн формын зөв ажиллагаа буюу логик үйлдлүүд дээр анхаарах хэрэгтэй байсан ба хамгийн түрүүнд хийх шаардлагатай зүйл нь аймаг сумдын датаг next.js дээрээ үүсгэсэн api-аасаа авч дэлгэцэнд харуулах байсан юм.

```
const fetchData = (url) => {
  return axios
    .get(`http://localhost:3000/api/${url}`)
    .then((res) => {
      const results = res.data;
      return results;
    })
    .catch((err) => {
      console.error(err);
    });
};
```

Код 8.1: Next.js дээр бичсэн серверээс датагаа татаж авах

Доор харагдаж буй хэсэгт компонентийн логик үйлдлүүд харагдаж байна. useForm() hook ашиглаж select-н утга өөрчлөгдсөн эсэхийг барьж авах, functional component ашиглан state дотор хэрэглэгчийн сонгосон аймаг, сум, хороог хадгална.

```
export default function Home() {
  const [values, handleChange] = useForm();
  const [data, setData] = useState({
    cities: [],
    districts: [],
    wards: [],
  });
```

```
8
9   useEffect(() => {
10    fetchData(`cities`)
11      .then((res) => {
12        setData({ ...data, cities: res });
13      })
14      .catch((err) => {
15        console.error(err);
16      });
17  }, []);
18
19  const register = (e) => {
20    e.preventDefault();
21    console.log(values);
22  };
23
24  const handleSelect = (id, type) => {
25    if (type == "city") {
26      fetchData(`cities/${id}`)
27        .then((res) => {
28          setData({ ...data, districts: res, wards: [] }); //set
                districts and clear wards data
29        })
30        .catch((err) => {
31          console.error(err);
32        });
33    } else if (type == "district") {
34      //get wards
```

```
35      fetchData(`cities/${values.city}/${id}`)
36        .then((res) => {
37          setData({ ...data, wards: res });
38        })
39        .catch((err) => {
40          console.error(err);
41        });
42    }
43  };
44
45  ...
```

Код 8.2: Component-н үндсэн логик үйлдлүүд

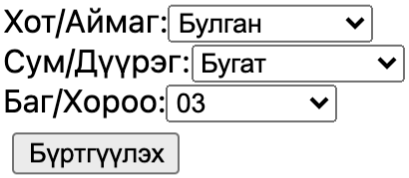
State дотор хэрэглэгчийн сонгосон мэдээллийг зөвөөр хадгалах шаардлагатай байсан. Эхний байдлаар доор харагдаж байгаагаар хадгалсан ч үүссэн асуудлууд нь замбаараагүй, эцэг сонголтыг сонгоход хүү сонголтуудыг цэвэрлэхэд төвөгтэй байв.

```
1  const [cities, setCities] = useState([]);
2  const [districts, setDistricts] = useState([]);
3  const [wards, setWards] = useState([]);
4  const [selectedCity, setSelectedCity] = useState();
5  const [selectedDistrict, setSelectedDistrict] = useState();
6  const [selectedWard, setSelectedWard] = useState();
```

Код 8.3: Дотоод төлвийн эхний хувилбар

Иймд state-ээ дараах байдлаар хадгаллаа.

```
1  const [data, setData] = useState({
2    cities: [],
3    districts: [],
```



Зураг 8.1: Формын эхний хувилбар

```
    wards: [],
});
```

Код 8.4: Дотоод төлвийн сайжруулсан хувилбар

Эхний шаардлагын дагуу формыг хэрэгжүүлсний дараах вэб дээр харагдах байдал

8.1.2 useState-н оронд useReducer hook ашиглах

useReducer нь useState hook-н өөр нэг хувилбар бөгөөд өөр дээрээ state болон action гэсэн параметрүүдийг хүлээн авдаг. State нь өгөгдөл хадгалах бол action нь тухайн state дээр ямар үйлдлүүдийг хийх шаардлагатайг хүлээж авдаг.

Хамгийн түрүүнд Action-ынхаа төрлүүдийг зарлана.

```
const SET_CITY = "city";
const SET_DISTRICT = "district";
const SET_WARD = "ward";
```

Код 8.5: Хийх үйлдлүүдийн төрлийг зарлах

Үүний дараа хийх үйлдлүүдээ switch case дотор тодорхойлно. Switch case ашигласнаар олон дахин функц зарлах шаардлагагүйгээр Action-н төрлөөс хамаарч тухайн үйлдлийг ажиллуулна.

```
const reducer = (state, action) => {
switch (action.type) {
  case SET_CITY:
    return {
      city: action.index,
      district: null,
      ward: null,
    };
  case SET_DISTRICT:
    return {
      ...state,
      district: action.index,
      ward: null,
    };
  case SET_WARD:
```

```
      return {
        ...state,
        ward: action.index,
      };
    default:
      return state;
  }
};
```

Код 8.6: Хийх үйлдлүүдийг тодорхойлох

useReducer ашигласнаар өгсөн шаардлагуудын нэг болох эцэг сонголтыг солиход хүү сонголтууд хоосрох ёстой гэснийг маш хялбар байдлаар шийдэх боломжтой болов. Ердөө тухайн сонголтын доор байгаа state-үүдийн утгыг анхны утгаар солисон.

```
  ...
    case SET_DISTRICT:
      return {
        ...state,
        district: action.index,
        ward: null,
      };
  ...
```

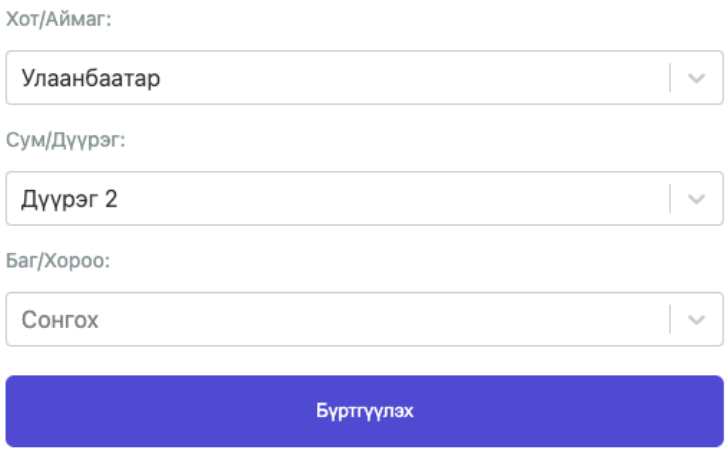
Код 8.7: Хүү сонголтуудыг цэвэрлэх

Одоо форм дээрээ React-Select санг ашиглаж компонент доторх кодоо бичих шаардлагатай. Энэ хэсэг нь DOM дээр рендерлэгдэнэ.

```
  ...
  <form className={styles.grid} onSubmit={register}>
    <label>
```

```
4       <p>Country:</p>
5       <Select
6         value={address.map((i, index) => ({ ...i, index }))[state.city
              ]}
7         onChange={(e) => handleChange(e.index, SET_CITY)}
8         options={address.map((i, index) => ({ ...i, index }))}
9         getOptionLabel={(option) => option.name}
10        getOptionValue={(option) => option.index}
11        placeholder="Choose␣country"
12      />
13    </label>
14  ...
```

Код 8.8: React-Select ашигласан байдал



Зураг 8.2: Формын эцсийн байдлаар харагдаж буй байдал

React-Select болон Material-UI ашигласны дараа формын маань харагдах байдал. Удирдагчийн өгсөн шаардлагуудын дагуу амжилттай хөгжүүлж дууслаа.

8.2 Toast компонент

Уг компонентийн гол үүрэг нь Мерчант төсөл дээр ашиглаж буй бүх хүсэлтүүдийн хариуг хэрэглэгч дээр харуулах үүрэгтэй. Мөн хөгжүүлж дууссаны дараа хэрэглэгчийн интерфэйсийн автоматжуулсан тест хийж байж production дээр орох боломжтой.

Сурах ур чадвар:

- State management-н гол ойлголт болох Context-н талаар мэдлэгтэй болно
- Өөрийн шаардлагад нийцсэн custom hook бичиж сурах
- Хэрхэн бичсэн компонент дээрээ UI автоматжуулсан тест бичих мэдлэг

Формын шаардлага:

- Material-UI-н Snackbar ашиглах
- Олон Toast зэрэг гаргадаг байх
- Toast нь үүсгэх, цэвэрлэх, устгах үйлдлүүдтэй байх
- Toast-н цаг дуусахад автоматаар цэвэрлэдэг байх
- UI автоматжуулсан тестийг давсан байх

8.2.1 Context үүсгэх

React-н өгөгдлийн урсгал нэг чиглэлд буюу зөвхөн эцгээс хүү компонент рүү утга дамжуулах чадвартай байдаг. Иймд нэг state-ээ нэгэндээ хамааралгүй олон өөр газарт ашиглагдаж байгаа компонентод ашиглахын тулд маш замбаараагүй дамжуулалт хийх шаардлага гардаг. Харин үүний нэг шийдэл нь Context гэх ойлголт бөгөөд бусад програмчлалын хэл дээр байдаг Global хувьсагч шиг ашиглах боломжтой.

Хамгийн эхлээд context-оо үүсгэж, түүний хийх үйлдлүүдийг зарлаж өгөх хэрэгтэй. Өмнөх хэрэгжүүлэлт дээр ашигласан useReducer-н мэдлэг хэрэг болов.

```
...
export const ACTION_TOAST = "TOAST";
export const ACTION_RESET = "RESET";
export const ACTION_CLEAR = "CLEAR";

export const INITIAL_STATE: { toasts: ToastType[] } = {
  toasts: [],
};

type Actions =
  | (ActionType<typeof ACTION_TOAST> & { toast: ToastType })
  | ActionType<typeof ACTION_RESET>
  | (ActionType<typeof ACTION_CLEAR> & { id: string });

export const ToastContext = createContext(null);
ToastContext.displayName = "ToastContext";
export const ToastControlContext = createContext(null);
ToastControlContext.displayName = "ToastControlContext";
...
```

Код 8.9: Context үүсгэх

Toast-н анхны утгыг зарласан байдал

```
...
export const INITIAL_STATE: { toasts: ToastType[] } = {
  toasts: [],
};
...
```

Код 8.10: Toast context-н анхны утгыг зарласан байдал

Мөн манайх төсөл дээрээ Typescript ашигладаг тул мэдээж Toast-н интерфэйсийг зааж өгөх хэрэгтэй. Ингэснээр Toast ашиглаж буй хөгжүүлэгч тухайн компонент ямар төрөлтэй ямар props-уудыг хүлээн авах шаардлагатайг хялбараар харах боломжтой. Мөн хөгжүүлэлтийн явцад гарах алдаа багасна.

```
...
export interface ToastType {
  id?: string;
  message: string;
  type: "success" | "info" | "warning" | "error";
  duration?: number;
}
...
```

Код 8.11: Toast-н интерфэйс тодорхойлох

Одоо шаардлагын дагуу Toast үүсгэх, цэвэрлэх, устгах үйлдэл хийх функцын кодыг бичсэн

```
...
function toast(state: typeof INITIAL_STATE, toast: ToastType): typeof
    state {
  return {
    ...state,
    toasts: [...state.toasts, toast],
  };
}
...
```

Код 8.12: Toast үүсгэх функц

```
...
function clear(state: typeof INITIAL_STATE, toastId: string): typeof
   state {
  return {
    ...state,
    toasts: state.toasts.filter((toast) => toast.id !== toastId),
  };
}
...
```

Код 8.13: Toast-г context-оос цэвэрлэх

Олон Toast зэрэг гарах боломжтой тул үүсгэсэн Toast бүр дээрээ uniq ID үүсгэж, түүний дараа тухайн ID-аар нь хайж олон цэвэрлэх боломжтой болно.

```
...
function generateToastId() {
  return Math.random().toString(36).substr(2, 9);
}
...
```

Код 8.14: Uniq ID гаргах

Тухайн үйлдлийг дуудахад дээр тодорхойлж өгсөн тус тусын үүрэгтэй функцууд ажиллана. Бүх Toast-г context дотроосоо устгахдаа case дээр анхны зарласан хоосон массивыг буцааж оноож байгаа.

```
...
function reducer(state = INITIAL_STATE, action: Actions): typeof
   state {
  switch (action.type) {
```

```
      case ACTION_TOAST:
        return toast(state, action.toast);
      case ACTION_RESET:
        return INITIAL_STATE;
      case ACTION_CLEAR:
        return clear(state, action.id);
      default:
        return state;
    }
  }
  ...
```

Код 8.15: useReducer дээр ажиллах үйлдлүүдийг заах

8.2.2 Custom Hook бичих

Hook бичиж өгснөөр Toast-г бүрэн ашиглах нөхцөл нь бүрдэнэ. useToast() болон useToastControl() гэсэн хоёр төрлийн hook бичиж өгсөн.

Тухайн context-г төсөл дотор буюу index.jsx файлд wrap хийж өгөөгүй нөхцөлд console дээр ашиглах боломжгүй гэсэн алдааг гаргаж өгөх шаардлагатай.

```
  ...
  export function useToastControl() {
    const dispatch = useContext(ToastControlContext);
    if (!dispatch) throw new TypeError("Please␣use␣within␣ToastProvider
        ");

    const reset = useCallback(() => dispatch({ type: ACTION_RESET }), [
        dispatch]);
  ...
```

Код 8.16: Context-г буруу ашигласан үед алдаа гаргах

Toast-г шинээр үүсгэх шаардлагатай үед уг код ажиллана. Хэрэв Toast-оо үүсгэхдээ дэлгэцэнд харагдах цагийг зааж өгөөгүй бол автоматаар 4000 ms-н дараа тухайн Toast санах ой болон хэрэглэгч дээр харагдах хэсгээс цэвэрлэгдэнэ.

```
...
const toast = useCallback(
  (toast: ToastType) => {
    const toastId = generateToastId();

    dispatch({ toast: { ...toast, id: toastId }, type: ACTION_TOAST
       });

    setTimeout(() => {
      dispatch({ id: toastId, type: ACTION_CLEAR });
    }, toast.duration || 4000);
  },
  [dispatch]
);
...
```

Код 8.17: Toast үүсгэх үүрэгтэй hook

Үүний дараа Toast компонентийнхоо интерфэйсийг Material-UI сан ашиглаж шийдэв.

```
...
return (
  <Snackbar
    anchorOrigin={{ horizontal: "center", vertical: "bottom" }}
    open={open}
```

```
      autoHideDuration={duration}
      onClose={onClose}
      action={action}
    >
      <Alert onClose={onClose} severity={type} sx={{ width: "100%" }}>
        {message}
      </Alert>
    </Snackbar>
  );
  ...
```

Код 8.18: Material-UI ашиглаж интерфэйсийг үүсгэх

Hook бичихгүйгээр шийдэх нь Toast-г ашиглахын тулд бүтэн компонент код бичих асуудалтай байсан ба custom hook-р шийдсэнээр хөгжүүлэгч ердөө ганц функцийг л дуудаж ажиллуулахад хангалттай. Ингэснээр хөгжүүлэлтийн хурданд ч эерэгээр нөлөөлнө.

```
  ...
  .then(
        (res) =>
          res.data &&
          toast({
            message: "Created product",
            type: "success",
          })
      )
  ...
```

Код 8.19: Toast-г дуудаж ашиглаж буй байдал

8.3 UI автоматжуулсан тест

Бичсэн компонентдоо тест код бичсэнээр ажиллагааг нь баталгаажуулах, алдааг илрүүлэх, хүнээр тест хийлгүүлэх шаардлагагүй болдог. Тестээ бичихдээ React багийн гишүүн Kent C. Dodds-н Jest сан дээр нэмэлт хөгжүүлэлт хийж гаргасан React Testing Library ашиглав.

Хуурамч цаг ашиглаж, тест хийх stage-н хурдыг нэмэгдүүлнэ. Ингэснээр заавал Toast дээр тавигдсан 4 секундыг хүлээх хэрэггүй.

```
...
jest.useFakeTimers();
...
```

Код 8.20: FakeTimer ашиглах

Хамгийн эхэнд хийгдэх тест бол DOM дээр зурагдаж байгаа эсэх, явуулсан string-г хэвлэж байгаа эсэх болон ”ХААХ” товчлуур дээр дарагдахад цэвэрлэгдэж байгаа эсэхийг шалгах юм.

```
...
const message = "Hello, it's toast";
render(<Toast message={message} type="success" />);
...
```

Код 8.21: Toast компонентийг DOM дээр зурна

```
...
expect(screen.getByText(message)).toBeInTheDocument();
...
```

Код 8.22: DOM дээр зурагдсан эсэхийг шалгана

```
...
userEvent.click(
  screen.getByRole("button", {
```

```
      name: /close/i,
    })
  );

  await waitForElementToBeRemoved(() => screen.getByText(message));

  expect(screen.queryByText(message)).not.toBeInTheDocument();
  ...
```

Код 8.23: Хаах товчлуур дээр дарахад устсан эсэхийг шалгах

Үүний дараачаар компонентийг хоёр удаа дуудаж ажиллуулаад интерфэйс дээр харагдах нийт хугацааг өөр өөрөөр зааж өгнө. Хөгжүүлэгч дурын хугацаа зааж өгсөн үед уг хугацааны үед ажиллаж байгаа эсэхийг шалгана.

```
  ...
  test("duration", async () => {
    const cssAnimation = 300;
    const toasts = [
      {
        duration: 4000,
        message: "Toast␣1",
      },
      {
        duration: 5000,
        message: "Toast␣2",
      },
  ];

  toasts.forEach((toast) =>
```

```
16      render(
17        <Toast
18          message={toast.message}
19          type="success"
20          duration={toast.duration}
21        />
22      )
23  );
24  ...
```

Код 8.24: Олон Toast зэрэг гаргаж дурын хугацааг зааж өгөх

```
1   ...
2   expect(screen.getByText("Toast␣1")).toBeInTheDocument();
3     await waitForElementToBeRemoved(() => screen.getByText("Toast␣1"),
        {
4       timeout: toasts[0].duration + cssAnimation,
5     });
6   expect(screen.queryByText("Toast␣1")).not.toBeInTheDocument();
7   ...
```

Код 8.25: Зааж өгсөн хугацааны дараа цэвэрлэгдэж байгаа эсэхийг шалгах

Хамгийн сүүлд нь миний бичсэн Toast дөрвөн өөр төрлийг хүлээж аваад тухайн төрлөөс нь хамаарч өөр загвартай харуулдаг тул уг төрлийн дагуу зурагдаж байгаа эсэхийг шалгах юм.

Toast-н хүлээж авах type-н жагсаалт:

- success
- info

- warning
- error

```
...
test("check type", () => {
  const types = [
    {
      class: "filledSuccess",
      message: "Success toast",
      type: "success",
    },
    { class: "filledError", message: "Error toast", type: "error" },
    { class: "filledInfo", message: "Info toast", type: "info" },
    {
      class: "filledWarning",
      message: "Warning toast",
      type: "warning",
    },
  ];

types.forEach((opt) => {
  const { container } = render(
    <Toast
      message={opt.message}
      type={opt.type as "success" | "info" | "warning" | "error"}
    />
  );
  const alert = container.firstChild;
```

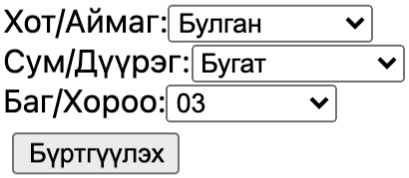
```
26
27      expect(alert.firstChild).toHaveClass(`MuiAlert-${opt.class}`);
28      cleanup();
29      });
30    });
31    ...
```

Код 8.26: Өгсөн type-н дагуу зурагдаж байгаа эсэхийг шалгах

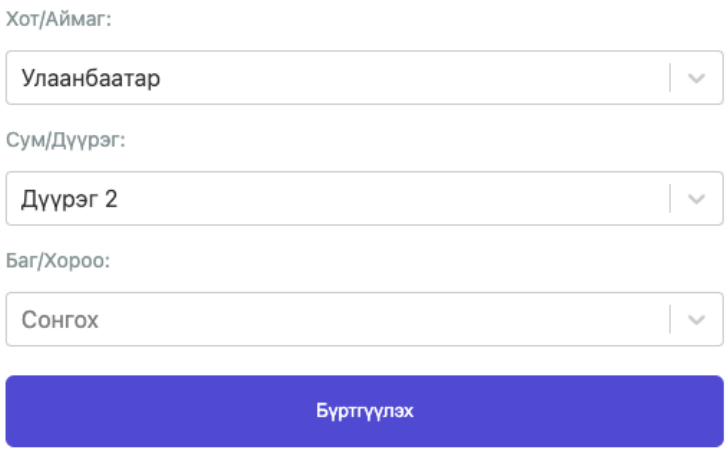
9. ДҮГНЭЛТ

9.1 Үр дүн

9.1.1 Аймаг сумдын мэдээлэл авдаг форм

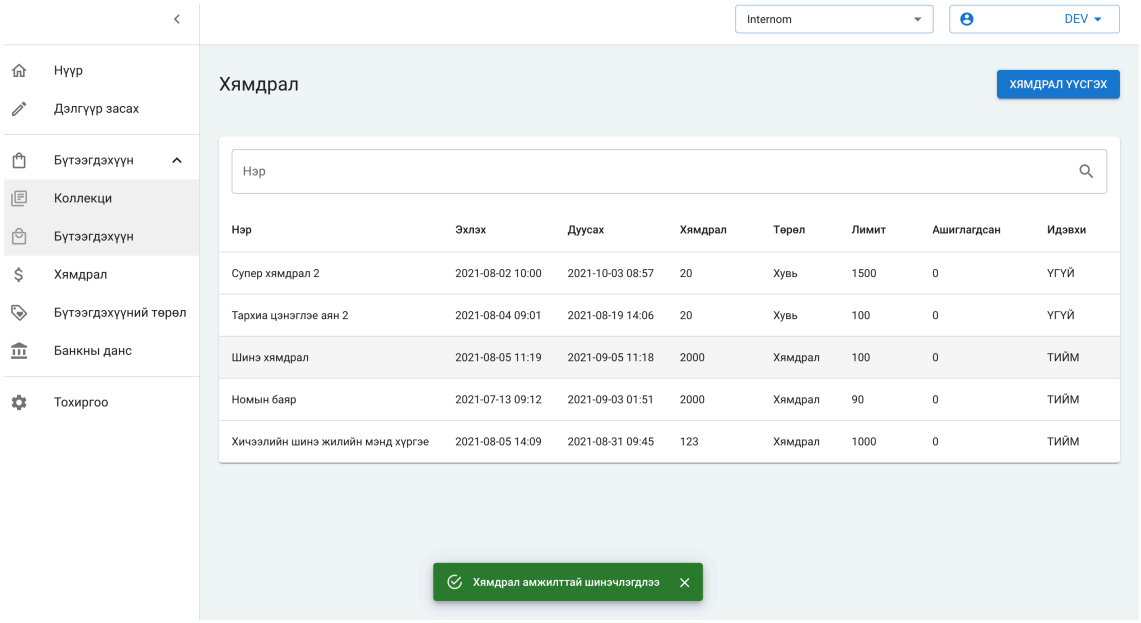


Зураг 9.1: Формын эхний хувилбар

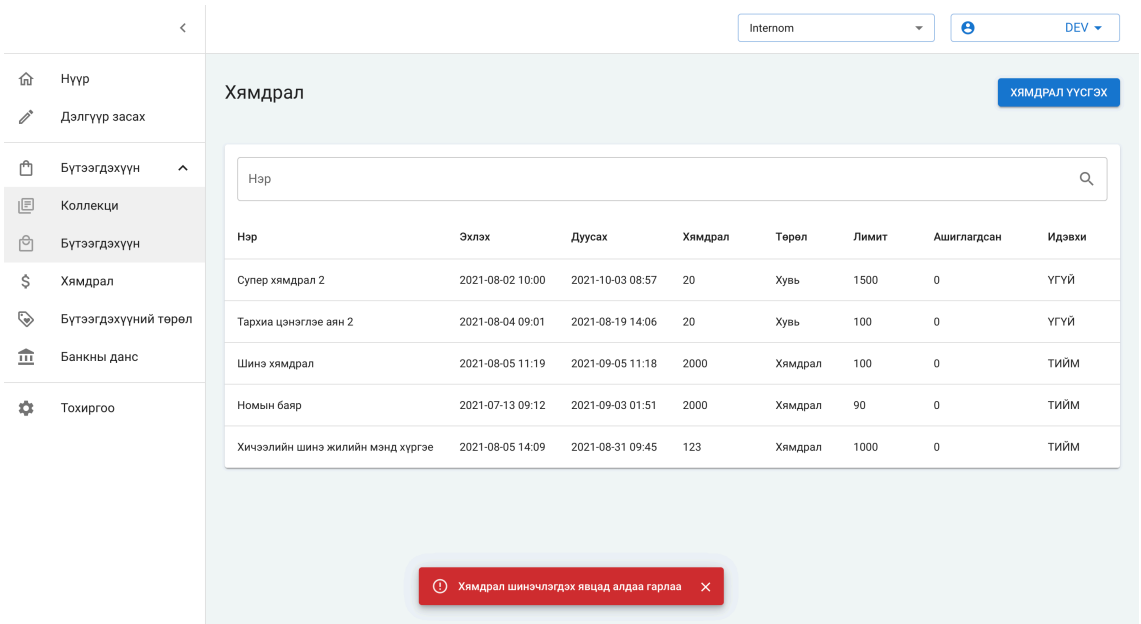


Зураг 9.2: Material ui болон React Select ашигласан байдал

9.1.2 Toast компонент



Зураг 9.3: Хүсэлт амжилттай болсон үед харагдах Toast



Зураг 9.4: Хүсэлт амжилтгүй болсон үед харагдах Toast

9.2 Үр дүнгийн тайлан

Миний бие Хуур Мюзик Групп ХХК-д 21 хоногийн хугацаатай мэргэжлийн дадлагыг амжилттай гүйцэтгэж дуусгалаа. Уг хугацаанд хичээлийн хүрээнд үзсэн онолын ойлголтуудыг практик дээр туршиж, хэрэгжүүлсэн ба хөгжүүлэлт голчилсон технологийн компанийн ерөнхий үйл ажиллагаа, баг хооронд зохицон ажиллах чадвар, хөгжүүлэлтийн шинэ арга барилуудыг амжилттай эзэмшсэн гэж дүгнэж байна.

Continuous Integration/Continuous Deployment, GIT дээрх Feature Branch, ашиглаж буй програмчлалын хэлнийхээ давуу талыг судлан уг хэлээрээ сэтгэж бичих, том асуудлыг олон болгон хувааж багаар, алхам дэс дараатай асуудлыг шийдвэрлэх мөн ашиглаж буй сан, технологийнхоо гарын авлага буюу documentation-тай илүү сайн танилцаж уг технологийнхоо цаана нь буй концептийг хялбараар ойлгох гэх мэт чадваруудыг эзэмшсэн. Үүнийгээ цаашид илүү хөгжүүлж мэргэшсэн фронт-энд хөгжүүлэгч болохоор зорьж байна. Дадлага хийсэн компани маань хэрэгжүүлж буй төсөлдөө үргэлж технологийн шинэ туршилтын хувилбаруудыг төвөгшөөлгүйгээр хэрэглэж, түүнийхээ алдааг илрүүлж, ажлын бус цагаараа хамтдаа шийдлийг хайж олон улсын нээлттэй эхийн төсөлд гар бие оролцдог нь бусад компаниудаас онцлог. Үүний үр дүнд манай дадлагын удирдагч болох С. Дөлмандах нь React-Native-н core contributor болж, 2019 онд болсон React-Native EU гэх олон улсын хөгжүүлэгчдийн эвентэд илтгэл тавьж байсан удаатай. Би цаашид өөрийн чөлөөт цагаа ашиглан дотоодын болон олон улсын нээлттэй эхийн төсөлд хувь нэмрээ оруулж гадны чадварлаг хөгжүүлэгчдийн арга барил, код бичих туршлага, тухайн асуудлыг хэрхэн шийдсэн гэх мэт үнэтэй мэдлэгүүдийг хуримтлуулж бусад хүмүүст мөн нээлттэй эхийн төсөлд оролцохын давуу талуудыг танилцуулж уриалахаар төлөвлөсөн байгаа.

Дадлагын эхэн үед React болон Next.js технологийн талаар судлах, түүнийгээ хэрэгжүүлэх, хөгжүүлэлтийн арга барилуудтай танилцах зорилготой байсан ба цаашид мэдээлэл технологийн ямар чиглэлээр мэргэшиж, түүндээ хүрэхийн тулд хэрхэн чадварлаг болох ёстойг ойлгосон тул зорилгодоо бүрэн хүрсэн гэдэгт итгэлтэй байна.

Bibliography

[1] Declarative програмчлал болон Imperative програмчлалын ялгаа
https://codeburst.io/declarative-vs-imperative-programming-a8a7c93d9ad2

[2] Material-ui Beta v5 хувилбарын Card ашиглах заавар
https://next.material-ui.com/api/card/

А. КОНВЕНЦ

Table A.1: Auftragsverwaltung систем дээрх Герман-Англи нэршлийн конвенц

№	Герман	Англи
1	auftragsverwaltung	order management
2	artikel	item
3	kunde	customer
4	benutzer	user
5	geld	money
6	betrag	amount
7	anzNachKomma	number after decimal point
8	waehrung	currency
9	addieren	add
10	subtrahieren	subtract
11	multiplizieren	multiply
12	umrechnen	convert
13	kompatibel	compatible
14	betragMitKomma	amount with decimal point
15	artikelnr	Item No.
16	artikelbezeichnung	Item Description
17	pries	Price
18	mindestbestand	Minimum Stock
19	bestand	Stock
20	aktualisieren	Update
21	lieferdatum	delivery date
22	apositionen	positions
23	einfuegen	insert
24	entfernen	remove
25	loeschen	delete
26	anrede	Title
27	vorname	First name
28	strasse	Street
29	plz	Zip code
30	ort	City
31	debitorennr	Account receivable number
32	eine artikel verwaltung	an article management
33	artikelliste	article list
34	datenquelle	data source
35	hinzufuegen	add
36	entfernen	remove
37	finden	find
38	findeArtikelMitUnterdeckung	find article with shortfall
39	posNr	PosNo
40	menge	Quantity

үргэлжнэ

Table A.1 – *үргэлжлэл*

№	**Герман**	**Англи**
41	tatsLieferdatum	Actual Delivery Date
42	istAbgeschlossen	isCompleted
43	findeAuftraegeZuArtikel	Find orders by item
44	findeAuftraegeZuKunde	Find orders by customer
45	naechsteAuftragsnr	Next order number
46	kennwortHash	password hash
47	fabrik	factory
48	verbindung	connection
49	meinlocale	myLocale
50	berechtigungssystem	authorization system

B. КОДЫН ХЭРЭГЖҮҮЛЭЛТ

B.1 Форм

B.1.1 useReducer болон React-Select сан ашигласан байдал

```
import { useState, useReducer } from "react";
import Head from "next/head";
import styles from "../styles/Home.module.css";
import Select from "react-select";
import address from "../data/address";

const SET_CITY = "city";
const SET_DISTRICT = "district";
const SET_WARD = "ward";

const reducer = (state, action) => {
  switch (action.type) {
    case SET_CITY:
      return {
        city: action.index,
        district: null,
        ward: null,
      };
    case SET_DISTRICT:
      return {
        ...state,
        district: action.index,
        ward: null,
      };
    case SET_WARD:
      return {
        ...state,
        ward: action.index,
      };
    default:
      return state;
  }
};

export default function Home() {
  const [state, dispatch] = useReducer(reducer, {
    city: null,
    district: null,
    ward: null,
  });

  const handleChange = (index, type) => {
    dispatch({ type: type, index });
  };
```

```
45
46  const register = (e) => {
47    e.preventDefault();
48    console.log(state);
49  };
50
51  return (
52    <div className={styles.container}>
53      <Head>
54        <title>Create Next App</title>
55      </Head>
56
57      <main className={styles.main}>
58        <div>
59          <form className={styles.grid} onSubmit={register}>
60            <label>
61              <p    >/:</p>
62              <Select
63                value={address.map((i, index) => ({ ...i, index }))[
                     state.city]}
64                onChange={(e) => handleChange(e.index, SET_CITY)}
65                options={address.map((i, index) => ({ ...i, index }))}
66                getOptionLabel={(option) => option.name}
67                getOptionValue={(option) => option.index}
68                placeholder="   "
69              />
70            </label>
71
72            <label>
73              <p    >/:</p>
74              <Select
75                value={
76                  state.district != null
77                    ? address[state.city].districts.map((i, index) =>
                         ({
78                          ...i,
79                          index,
80                        }))[state.district]
81                    : []
82                }
83                onChange={(e) => handleChange(e.index, SET_DISTRICT)}
84                options={
85                  state.city != null
86                    ? address[state.city].districts.map((i, index) =>
                         ({
87                          ...i,
88                          index,
89                        }))
90                    : []
91                }
92                getOptionLabel={(option) => option.name}
93                getOptionValue={(option) => option.index}
```

```
                placeholder="   "
              />
            </label>

            <label>
              <p    >/:</p>
              <Select
                value={
                  state.ward != null
                    ? address[state.city].districts[state.district].
                        wards.map(
                          (i, index) => ({ ...i, index })
                        )[state.ward]
                    : []
                }
                onChange={(e) => handleChange(e.index, SET_WARD)}
                options={
                  state.district != null
                    ? address[state.city].districts[state.district].
                        wards.map(
                          (i, index) => ({ ...i, index })
                        )
                    : []
                }
                getOptionLabel={(option) => option.name}
                getOptionValue={(option) => option.index}
                placeholder="   "
              />
            </label>
            <button className={styles.registerBtn    }></button>
          </form>
        </div>
      </main>
    </div>
  );
}
```

B.1.2 Functional component дээр state ашигласан байдал

```
import { useEffect, useState } from "react";
import Head from "next/head";
import styles from "../styles/Home.module.css";
import axios from "axios";
import useForm from "../utils/useForm";

const fetchData = (url) => {
  //get data from next.js api
  return axios
    .get(`http://localhost:3000/api/${url}`)
    .then((res) => {
      const results = res.data;
```

```
      return results;
    })
    .catch((err) => {
      console.error(err);
    });
};

export default function Home() {
  const [values, handleChange] = useForm();
  const [data, setData] = useState({
    cities: [],
    districts: [],
    wards: [],
  });

  useEffect(() => {
    fetchData(`cities`)
      .then((res) => {
        setData({ ...data, cities: res });
      })
      .catch((err) => {
        console.error(err);
      });
  }, []);

  const register = (e) => {
    e.preventDefault();
    console.log(values);
  };

  const handleSelect = (id, type) => {
    if (type == "city") {
      fetchData(`cities/${id}`)
        .then((res) => {
          setData({ ...data, districts: res, wards: [] }); //set
              districts and clear wards data
        })
        .catch((err) => {
          console.error(err);
        });
    } else if (type == "district") {
      //get wards
      fetchData(`cities/${values.city}/${id}`)
        .then((res) => {
          setData({ ...data, wards: res });
        })
        .catch((err) => {
          console.error(err);
        });
    }
  };

```

```
  const OptionItems = (props) => {
    const options = props.items.map((item) => {
      return (
        <option key={item.id} value={item.id}>
          {item.name}
        </option>
      );
    });

    return options;
  };

  return (
    <div className={styles.container}>
      <Head>
        <title>Create Next App</title>
      </Head>

      <main className={styles.main}>
        <div>
          <form className={styles.grid} onSubmit={register}>
            <label>
              /:
              <select
                value={values.city}
                defaultValue="default"
                name="city"
                onChange={(e) => {
                  handleChange(e.target.name, e.target.value);
                  handleSelect(e.target.value, "city");
                }}
              >
                <option value="default" hidden>
                  /
                </option>

                {data.cities.length > 0 ? (
                  <OptionItems items={data.cities} />
                ) : null}

                {/* {data.cities.map((item) => {
                  return (
                    <option key={item.id} value={item.name}>
                      {item.name}
                    </option>
                  );
                })} */}
              </select>
            </label>

            <label>
              /:
```

```
                <select
                  value={values.district}
                  defaultValue="default"
                  name="district"
                  onChange={(e) => {
                    handleChange(e.target.name, e.target.value);
                    handleSelect(e.target.value, "district");
                  }}
                >
                  <option value="default" hidden>
                    /
                  </option>
                  {data.districts.length > 0 ? (
                    <OptionItems items={data.districts} />
                  ) : null}
                </select>
              </label>

              <label>
                /:
                <select
                  value={values.ward}
                  defaultValue="default"
                  name="ward"
                  onChange={(e) => {
                    handleChange(e.target.name, e.target.value);
                  }}
                >
                  <option value="default" hidden>
                    /
                  </option>
                  {data.wards.length > 0 ? (
                    <OptionItems items={data.wards} />
                  ) : null}
                </select>
              </label>
              <button style={{ margin: "5px"      }}></button>
            </form>
          </div>
        </main>
      </div>
    );
}
```

B.2 Toast компонент

B.2.1 Toast/context.tsx - Context үүсгэх

```
import { createContext, useReducer } from "react";

```

```
import { ActionType } from "types";

export interface ToastType {
  id?: string;
  message: string;
  type: "success" | "info" | "warning" | "error";
  duration?: number;
}

export const ACTION_TOAST = "TOAST";
export const ACTION_RESET = "RESET";
export const ACTION_CLEAR = "CLEAR";

export const INITIAL_STATE: { toasts: ToastType[] } = {
  toasts: [],
};

type Actions =
  | (ActionType<typeof ACTION_TOAST> & { toast: ToastType })
  | ActionType<typeof ACTION_RESET>
  | (ActionType<typeof ACTION_CLEAR> & { id: string });

export const ToastContext = createContext(null);
ToastContext.displayName = "ToastContext";
export const ToastControlContext = createContext(null);
ToastControlContext.displayName = "ToastControlContext";

function toast(state: typeof INITIAL_STATE, toast: ToastType): typeof
   state {
  return {
    ...state,
    toasts: [...state.toasts, toast],
  };
}

function clear(state: typeof INITIAL_STATE, toastId: string): typeof
   state {
  return {
    ...state,
    toasts: state.toasts.filter((toast) => toast.id !== toastId),
  };
}

function reducer(state = INITIAL_STATE, action: Actions): typeof state
   {
  switch (action.type) {
    case ACTION_TOAST:
      return toast(state, action.toast);
    case ACTION_RESET:
      return INITIAL_STATE;
    case ACTION_CLEAR:
      return clear(state, action.id);
```

```
    default:
      return state;
  }
}

export const ToastProvider = ({ children }) => {
  const [state, dispatch] = useReducer(reducer, INITIAL_STATE);

  return (
    <ToastContext.Provider value={state}>
      <ToastControlContext.Provider value={dispatch}>
        {children}
      </ToastControlContext.Provider>
    </ToastContext.Provider>
  );
};
```

B.2.2 Toast/hooks.ts - Custom hook бичих

```
import { useCallback, useContext, useMemo } from "react";

import {
  ACTION_CLEAR,
  ACTION_TOAST,
  ACTION_RESET,
  ToastContext,
  ToastControlContext,
  ToastType,
} from "./context";

function generateToastId() {
  return Math.random().toString(36).substr(2, 9);
}

export function useToast() {
  const state = useContext(ToastContext);
  if (!state) throw new TypeError("Please␣use␣within␣ToastProvider");
  return state;
}

export function useToastControl() {
  const dispatch = useContext(ToastControlContext);
  if (!dispatch) throw new TypeError("Please␣use␣within␣ToastProvider")
     ;

  const reset = useCallback(() => dispatch({ type: ACTION_RESET }), [
     dispatch]);

  const toast = useCallback(
    (toast: ToastType) => {
      const toastId = generateToastId();
```

```
31
32        dispatch({ toast: { ...toast, id: toastId }, type: ACTION_TOAST
              });
33
34        setTimeout(() => {
35          dispatch({ id: toastId, type: ACTION_CLEAR });
36        }, toast.duration || 4000);
37      },
38      [dispatch]
39    );
40
41    return useMemo(() => {
42      return { reset, toast };
43    }, [toast, reset]);
44  }
```

B.2.3 Toast/index.ts

```
1  export * from "./Toast";
2  export * from "./context";
3  export * from "./hooks";
```

B.2.4 Toast/Toast.tsx - Үндсэн Toast компонент

```
1  import { useState, forwardRef } from "react";
2
3  import { IconButton, Snackbar } from "@material-ui/core";
4  import MuiAlert, { AlertProps } from "@material-ui/core/Alert";
5  import { Close as CloseIcon } from "@material-ui/icons";
6
7  import { ToastType } from "./context";
8  import { useToast } from "./hooks";
9
10 const Alert = forwardRef<HTMLDivElement, AlertProps>(function Alert(
11   props,
12   ref
13 ) {
14   return <MuiAlert elevation={6} ref={ref} variant="filled" {...props}
         />;
15 });
16
17 export function Toast({ message, type, duration = 4000 }: ToastType) {
18   const [open, setOpen] = useState(true);
19
20   const onClose = () => setOpen(false);
21
22   const action = (
23     <IconButton
24       size="small"
```

```
      aria-label="close"
      color="inherit"
      onClick={onClose}
    >
      <CloseIcon fontSize="small" />
    </IconButton>
  );

  return (
    <Snackbar
      anchorOrigin={{ horizontal: "center", vertical: "bottom" }}
      open={open}
      autoHideDuration={duration}
      onClose={onClose}
      action={action}
    >
      <Alert onClose={onClose} severity={type} sx={{ width: "100%" }}>
        {message}
      </Alert>
    </Snackbar>
  );
}

export function ToastContainer() {
  const { toasts } = useToast();
  return toasts.map((toast) => <Toast key={toast.id} {...toast} />);
}
```

B.3 Toast компонент дээр интерфэйсийн автоматжуулсан тест бичсэн байдал

B.3.1 tests/Toast.test.tsx

```
import {
  cleanup,
  render,
  screen,
  waitForElementToBeRemoved,
} from "@testing-library/react";
import userEvent from "@testing-library/user-event";

import { Toast } from "../../../components/Toast";

jest.useFakeTimers();
describe("Test: Toast component", () => {
  test("show message and delete toast after close button ", async () =>
      {
    const message = "Hello, it's toast";

    render(<Toast message={message} type="success" />);
    expect(screen.getByText(message)).toBeInTheDocument();

    userEvent.click(
      screen.getByRole("button", {
        name: /close/i,
      })
    );

    await waitForElementToBeRemoved(() => screen.getByText(message));
    expect(screen.queryByText(message)).not.toBeInTheDocument();
  });

  test("duration", async () => {
    const cssAnimation = 300;
    const toasts = [
      {
        duration: 4000,
        message: "Toast 1",
      },
      {
        duration: 5000,
        message: "Toast 2",
      },
    ];

    toasts.forEach((toast) =>
      render(
        <Toast
          message={toast.message}
```

```
46            type="success"
47            duration={toast.duration}
48          />
49        )
50      );
51
52      expect(screen.getByText("Toast␣1")).toBeInTheDocument();
53      await waitForElementToBeRemoved(() => screen.getByText("Toast␣1"),
           {
54        timeout: toasts[0].duration + cssAnimation,
55      });
56      expect(screen.queryByText("Toast␣1")).not.toBeInTheDocument();
57
58      expect(screen.getByText("Toast␣2")).toBeInTheDocument();
59      await waitForElementToBeRemoved(() => screen.getByText("Toast␣2"),
           {
60        timeout: 1000 + cssAnimation,
61      });
62      expect(screen.queryByText("Toast␣2")).not.toBeInTheDocument();
63    });
64
65    test("check␣type", () => {
66      const types = [
67        {
68          class: "filledSuccess",
69          message: "Success␣toast",
70          type: "success",
71        },
72        { class: "filledError", message: "Error␣toast", type: "error" },
73        { class: "filledInfo", message: "Info␣toast", type: "info" },
74        {
75          class: "filledWarning",
76          message: "Warning␣toast",
77          type: "warning",
78        },
79      ];
80
81      types.forEach((opt) => {
82        const { container } = render(
83          <Toast
84            message={opt.message}
85            type={opt.type as "success" | "info" | "warning" | "error"}
86          />
87        );
88        const alert = container.firstChild;
89
90        expect(alert.firstChild).toHaveClass(`MuiAlert-${opt.class}`);
91        cleanup();
92      });
93    });
94  });
```